Strong Convex Relaxations and Tailored Conic Solvers with Applications in Robotics

by

Alexandre Amice

B.S.E. University of Pennsylvania (2020)
M.S.E. University of Pennsylvania (2020)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2026

Authored by: Alexandre Amice
Department of Electrical Engineering and Computer Science
May 15, 2026

Certified by: Pablo A. Parrilo
Joseph F. and Nancy P. Keithley Professor in Electrical Engineering, Thesis Supervisor

Certified by: Russ Tedrake
Toyota Professor of Electrical Engineering and Computer Science,
Aeronautics and Astronautics, and Mechanical Engineering, Thesis Supervisor

Accepted by: Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

THESIS COMMITTEE

Thesis Supervisors

Pablo A. Parrilo
Joseph F. and Nancy P. Keithley Professor in Electrical Engineering
Department of Electrical Engineering and Computer Science

Russ Tedrake
Toyota Professor of Electrical Engineering and Computer Science,
Aeronautics and Astronautics, and Mechanical Engineering
Department of Electrical Engineering and Computer Science
Department of Aeronautics and Astronautics
Department of Mechanical Engineering

Thesis Readers

Robert M. Freund
Theresa Seley Professor in Management Science
Sloan School of Management

Bartolomeo Stellato
Assistant Professor
Department of Operations Research and Financial Engineering, Princeton University

Strong Convex Relaxations and Tailored Conic Solvers with Applications in Robotics

by

Alexandre Amice

Submitted to the Department of Electrical Engineering and Computer Science
on May 15, 2026 in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

ABSTRACT

Many challenging robotics and control problems are well modeled as polynomial optimization programs (POP). Although POPs are difficult to solve in general, the moment/sums-of-squares (SOS) hierarchy provides a systematic way to construct convex relaxations of increasing strength. These relaxations have shown broad practical value in robotics, but their effectiveness depends on the full computational pipeline: accurately modeling the original problem, designing a powerful yet tractable relaxation, and solving the resulting convex optimization problems efficiently. This thesis considers a problem or tools for each stage of that pipeline.

In part one of this thesis, we collect the many preliminary technical tools that will be used throughout our work.

In part two of this thesis, we consider the modeling and relaxation parts of the pipeline. First, we develop a polynomial optimization program for certifying that a region of a robot's configuration space is collision-free. This allows us to use SOS programming to certify non-collision of polynomial trajectories, certify the safety of full-dimensional regions of configuration space, and build inner approximations of the collision-free configuration space to enable rapid collision-free motion planning. Our method is the first efficient method for certifying full-dimensional volumes of a robot's configuration space are collision free, and effective modeling choices enables our approach to scale to real, high-dimensional robotics systems.

Next, we turn our attention to designing effective relaxations. SOS relaxations are sensitive to how the original polynomial programs are formulated. Moreover, the real-time requirements of many robotics pipelines requires us to balance the strength of the convex relaxation against the solve time. Given a POP with equality constraints, we demonstrate a procedure for rapidly finding additional equalities capable of strengthening the SOS relaxation. Our method relies only on simple linear algebra, is fast enough to run as a lightweight preprocessing step before solving the SOS relaxation, and recovers many of the known strengthened formulations of POP programs in the robotics literature.

In part three of this thesis, we develop infrastructure and methods for solving convex programs produced by the relaxation procedure. We begin by introducing `CCosmo`, a family of `C++` implementation of a popular alternating direction method of multipliers algorithm from the literature for solving convex optimization programs. We provide an implementation

for convex programs with quadratic costs, add support for a matrix standard form commonly appearing in SOS relaxations, and a GPU implementation optimized for solving large batches of small convex optimization programs.

Next, we focus on solving linear systems of matrix equations. Solving linear systems is a fundamental subroutine in nearly all convex optimization algorithms, and matrix equations arise naturally when solving SOS relaxations. We first provide a new direct method for solving positive semidefinite, generalized Sylvester equations. This algorithm forms the basis of our matrix standard form solver in `CCosmo`. We also develop a sparse elimination method for solving a linear matrix equation which arises when solving a particular structured least-squares problem. This problem arises naturally in the design of controllers for networked linear systems. The results of this part may be of independent interest.

Our part on solving culminates in a scalable method for large Graph of Convex Sets (GCS) programs. Introduced in [1], GCS considers an optimization problem over a graph where every vertex and edge is paired with a convex program. GCS extends the powerful modeling framework of graph optimization programs such as shortest path and travelling salesman to handle continuous decisions while navigating the graph, and has become increasingly popular in robotics. A parsimonious application of the SOS methodology generates a highly effective convex relaxation of GCS that inherits substantial structure from the graph.

We design a decomposition-based solver, named `VEGA`, for GCS relaxations: decomposing the problem into its natural constituent parts: a program at every vertex, a program at every edge, and a classical graph optimization program. At every iteration, each of these programs will be solved in parallel and driven to consensus over the iterations, resulting in a sequence of closely related convex optimization programs with quadratic costs over the iterations. `VEGA` is built on top of `CCosmo`, leveraging the warm-start capabilities to amortize the solve times of the subproblems of the `VEGA` iterations and matrix-form solver to more efficiently solve some of the programs.

In summary, this thesis develops contributions to the entire convex relaxation pipeline for problems in robotics; from the high level modeling of problems, to their effective relaxation, and finally to their actual solution.

Thesis supervisor: Pablo A. Parrilo
Title: Joseph F. and Nancy P. Keithley Professor in Electrical Engineering

Thesis supervisor: Russ Tedrake
Title: Toyota Professor of Electrical Engineering and Computer Science,
Aeronautics and Astronautics, and Mechanical Engineering

Acknowledgments

My PhD at MIT has been the most exciting and rewarding experience of my life. I consider myself incredibly lucky to have been afforded the opportunity to write this thesis. More than anything, I am grateful to the many mentors, collaborators, friends, and family members whose support made it possible.

I would first like to thank my advisors, Pablo Parrilo and Russ Tedrake. I could not have asked for a better pair of co-advisors. Working with Pablo has been one of the most intellectually humbling experiences of my life. Pablo makes the world of math seem smaller, seeing connections everywhere and bringing a clarity to the subject that is difficult to match. He combines this with a rare gift for teaching and explaining, and with a humility that I have also tried to learn from. His insights have been instrumental in guiding my understanding of the field, and he has been an extraordinary role model both professionally and personally.

Throughout my PhD, Russ has epitomized what it means to be a curious engineer. Russ has a remarkable knack for finding hard problems at the frontier of the field, distilling them to their core difficulty, and finding just the right approach to make progress. He taught me how to enjoy the process of research. His optimism carried me through several research slumps and helped push forward results I would not have believed were possible.

Striving to live up to Pablo's and Russ's high standards and scientific ideals has been the most challenging and rewarding part of graduate school, and I am grateful to have had the opportunity.

I am also grateful to my committee members, Bartolomeo Stellato and Robert Freund. Their insights helped shape many elements of this thesis, and I am fortunate to have had such engaged committee members.

Next, I would like to thank my closest collaborators throughout my time at MIT. Hongkai Dai acted as my first non-advisor mentor in graduate school and taught me many of the basic skills I rely on today: how to write a paper, write research code, and design experiments. He also taught me the satisfaction of writing high-performance code and the impact of turning a theoretical project into a practically useful one. Tobia Marcucci showed me the value of thinking deeply, rigorously, and mathematically about robotics problems, even as the field continues to move at an increasingly frenetic pace. I am grateful to Lujie Yang, with whom I navigated the early years of graduate school during the pandemic, and who was my first collaborator. Bernhard Paus Graesdal has been a generous collaborator and friend through the end of the PhD, always willing to make time for a hard problem or a careful discussion. Finally, I would like to thank Peter Werner, who has served the dual role of my closest collaborator and one of my closest friends for the past five years. No one has spent more time listening to my research ideas, helping me prepare for presentations, and pushing

through long nights to meet deadlines, and I am grateful for his many contributions, big and small, to my work.

Beyond my closest collaborators, I would like to thank all the members of the Robot Locomotion Group and the Parrilo lab who I have been fortunate enough to overlap with. All of you contributed to making the lab the fun and intellectually stimulating environment I always hoped graduate school would be.

I would also like to thank my family and closest friends, who provided so much support outside the rigors of scientific research. I am grateful to Alec Yen, Ethan Fahnestock, and Peter Werner, with whom I enjoyed many long hikes and rock climbs in New Hampshire. I thank my sister, Charlotte, and my brother, Quentin, for their persistent encouragement.

This thesis is dedicated to my parents, Anne-Chantal and Olivier Amice, and my fiancée, Jenna Steichen. My parents gave me every opportunity imaginable growing up and were a constant source of support throughout the PhD. I would not be where I am today without their sacrifices and the many lessons they taught me growing up. Jenna has been the best partner I could possibly have asked for throughout my PhD, and I look forward to this next stage together.

This PhD was made possible by generous financial support[1]. The work described in this dissertation was supported in part by Amazon[2], the National Science Foundation[3], the Robotics and AI Institute[4], and by the MIT Quest for Intelligence[5].

[1]Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of any funding source or agency.

[2]Dynamic Manipulation Award No. 2D-19158851 from June 1–August 31, 2022 and June 1–August 31, 2024

[3]National Institute for Foundations of Data Science Grant No. DMS–2022448, from January 16–May 31, 2024

[4]Graphs of Convex Sets for Mobile Manipulators, from September 1, 2023–January 15, 2024

[5]September 1, 2022–January 15, 2023

Contents

List of Figures 15
List of Tables 17

I Background **25**

1 Elementary Background **27**
1.1 Indexing notation . . . 27
1.2 Sets and Vector Spaces . . . 27
1.3 Elementary Convex Analysis . . . 28
1.3.1 Support Function, Dual, and Recession Cones . . . 30
1.4 Tensor Products and Matrix Products . . . 31
1.4.1 Symmetric Tensors . . . 32
1.4.2 Matrix Products . . . 33
1.4.3 Tensor Products of Cones, Separable Matrices, and Positive Maps . . . 34
1.5 Polynomials . . . 36
1.5.1 Polynomials as Maps on Symmetric Tensors . . . 38
1.5.2 Linear Subsets of $\mathbb{R}[x]$ . . . 39
1.5.3 Conic Subsets of $\mathbb{R}[x]$ . . . 39
1.5.4 Semialgebraic Sets . . . 42
1.6 Conic Programs . . . 42
1.7 Graphs . . . 43
1.8 Summary of Notation . . . 43

2 Convex Relaxations: Sums-of-Squares and Tensor Product Relaxations **47**
2.1 Sums-Of-Squares Programming . . . 47
2.1.1 Certificates of Nonnegativity . . . 49
2.1.2 The SOS Hierarchy For Polynomial Optimization . . . 53
2.1.3 Complexity of the SOS Hierarchy . . . 54
2.2 Tensor Product Relaxations and the Moment Hierarchy . . . 55
2.2.1 Moment Relaxations . . . 57
2.2.2 Duality . . . 60
2.2.3 General Tensor Product Relaxations . . . 61
2.3 Deferred Proofs . . . 65
2.3.1 Proof of Example 2 . . . 65

2.3.2 Proof of Degree Bound in Example 3 . . . 66

3 Graphs of Convex Sets 67
3.1 Graph Problems . . . 67
3.1.1 Vertex Local Constraints . . . 69
3.2 The GCS Extension . . . 69
3.3 The GCS Convex Relaxation . . . 71
3.4 Connection to Sums Of Squares . . . 74

4 Alternating Direction Method of Multipliers 75
4.1 Acceleration Via Restarted Anchoring . . . 76

II Effective Sums-of-Squares Programming in Robotics 79

5 Certifying Collision-Free Subsets of Configuration Space 81
5.1 Related Work . . . 82
5.2 Problem Statement . . . 84
5.3 Mathematical Preliminaries . . . 85
5.3.1 Separating Convex Bodies . . . 85
5.3.2 Rational Forward Kinematics . . . 87
5.4 Certifying a Region is in $\mathcal{T}^{\text{free}}$ . . . 90
5.4.1 Method via Parametrized Hyperplane Certificates of Non-Collision . . . 90
5.4.2 Method via Polynomial Infeasibility Certificates . . . 93
5.4.3 Power of the Certification Programs . . . 94
5.4.4 Specialization to Trajectories . . . 95
5.5 Growing Regions in $\mathcal{T}^{\text{free}}$ . . . 96
5.6 Results . . . 99
5.6.1 Simple Robots . . . 100
5.6.2 KUKA IIWA robot . . . 105
5.6.3 UR3e Robot . . . 107
5.6.4 Fast Certification of Trajectories . . . 109
5.7 Covering Free Space in Convex Regions . . . 114
5.7.1 Related Works . . . 114
5.7.2 Convex Covers, Visibility, and Cliques . . . 116
5.7.3 Algorithm . . . 117
5.7.4 Results . . . 120
5.7.5 Limitations of Approximating Convex Sets with Cliques . . . 121
5.8 Conclusions . . . 123
5.9 Deferred Proofs . . . 123
5.9.1 The Certification Programs are Necessary and Sufficient . . . 123

6 Practical Strengthening of Semidefinite Relaxations 127
6.1 Introduction and Related Work . . . 128
6.2 Simulation Lemma: When Can a New Equality Help? . . . 130

6.3 Generating New Quadratic Equalities for QCQPs . . . 133
6.3.1 Warmup: Linearly Constrained Quadratic Program . . . 133
6.3.2 Explicit Low Degree Construction . . . 135
6.3.3 High Degree Search . . . 137
6.4 Results . . . 139
6.4.1 Ideal of Rotated Vectors . . . 139
6.4.2 Cloned-Quaternion Pose Estimation . . . 140
6.4.3 Generating New Equalities: The General POP Case . . . 141
6.5 Conclusion . . . 142

III Tailored Conic Solvers 143

7 `CCosmo`: A General and Customizable First-Order Convex Solver 145
7.1 Related Work . . . 146
7.2 Problem Formulation . . . 147
7.2.1 Optimality and Infeasibility Conditions . . . 148
7.3 General Solution Method . . . 149
7.3.1 Form of $\mathcal{D}_x$ and $\mathcal{D}_s$ . . . 149
7.3.2 Affine Subspace Projection . . . 150
7.3.3 Cone Projection . . . 152
7.3.4 Termination Criterion . . . 153
7.3.5 Summary of the Base Algorithm . . . 154
7.4 Enhancements . . . 155
7.4.1 Using Acceleration . . . 155
7.4.2 Adaptive Penalty Function Selection . . . 156
7.5 Results . . . 160
7.5.1 Standard Form Benchmark . . . 160
7.5.2 Fixed Contact-Schedule Centroidal Locomotion MPC . . . 161
7.5.3 Enhancements Ablation . . . 162
7.5.4 Matrix Standard Form Results . . . 170
7.6 `CCosmo` on the GPU . . . 171
7.6.1 A Primer on GPU Architectures . . . 174
7.6.2 `Cuda` Implementation . . . 175
7.6.3 Results . . . 176
7.7 Conclusion . . . 179
7.8 Deferred Proofs . . . 180
7.8.1 Duality in Convex Quadratic Programs . . . 180
7.8.2 Proof of Cone Projection . . . 181
7.8.3 Proof of Proposition 4 . . . 181

8 Solving Tensor-Structured Linear Systems 183
8.1 Positive Semidefinite Generalized Sylvester Equations . . . 183
8.1.1 Related Work . . . 184
8.1.2 Conditions for Solvability . . . 185

8.1.3 Simultaneous Diagonalizations of Positive Semidefinite Matrices . . . 186
8.1.4 Solving the Equation . . . 189
8.1.5 Results . . . 192
8.2 Solving Poset Structure Least Squares Problems . . . 198
8.2.1 Preliminaries: Posets and Graphs of Linear Systems . . . 200
8.2.2 Problem Formulation . . . 206
8.2.3 Main Results . . . 206
8.2.4 Results . . . 212
8.2.5 Conclusion . . . 214

9 Solving Graph of Convex Sets Instances **217**
9.1 An ADMM Split for GCS . . . 217
9.2 Implementation of the ADMM Steps . . . 221
9.2.1 Selection of the Penalty Functions . . . 221
9.2.2 Solving the Vertex Programs . . . 221
9.2.3 Solving the Edge and Graph Programs . . . 222
9.2.4 Inexact Substeps . . . 223
9.2.5 Termination Criterion . . . 225
9.3 Test Instances: the `GcsCc` Library and `GcsBench` Instances . . . 225
9.4 Results . . . 227
9.4.1 Centralized Solver Results . . . 228
9.4.2 Regularized Problems . . . 229
9.4.3 Performance of `VEGA` . . . 231
9.5 Conclusion . . . 237

References 243

A Appendices for Chapter 5 **265**
A.1 Algebraic Kinematics . . . 265
A.2 Semialgebraic Descriptions of Set Membership for Common Convex Bodies . 267
A.3 Description of Set Membership for Common Convex Bodies . . . 267
A.3.1 Semialgebraic Description under Rational Forward Kinematics . . . 268
A.4 Parametrized Hyperplane Separation Condition for the Cylinder . . . 269
A.5 Practical aspects . . . 270
A.5.1 Choosing the Reference Frame . . . 271
A.5.2 Basis Selection . . . 271
A.5.3 Parallelization . . . 274

B Appendices for Chapter 7 **275**
B.1 Supported Sets . . . 275
B.2 Locomotion MPC . . . 277
B.3 Breakdown of CPU/GPU Transfer Times for Section 7.6 . . . 278

C Appendix for Chapter 9 **281**
C.1 `GcsBench` Instances . . . 281
C.1.1 `Maze` . . . 288
C.1.2 `Helicopter` . . . 288
C.1.3 `IiwaShelf` . . . 288
C.1.4 `Bimanual` . . . 288
C.1.5 `SdpPushing` . . . 289
C.1.6 `ContactGraph` . . . 289
C.2 Shortest Path in a GCS on `GcsBench` . . . 289
C.2.1 Regularized Problems . . . 290
C.3 Detailed `VEGA` Results on `GcsBench` . . . 293
C.3.1 Original Problems . . . 293
C.3.2 Original Problems in Half-Step Mode . . . 294
C.3.3 Edge-Regularized Problems . . . 296
C.3.4 Edge-Regularized Problems in Half-Step Mode . . . 297

List of Figures

1 Model-Relax-Solve Workflow . . . 20

2.1 Prestel Example Feasible Region . . . 51

3.1 Classic Graph Problems . . . 67
3.2 GCS Shortest Path Extension . . . 70

5.1 Separation and Intersection Alternatives . . . 86
5.2 Double-Pendulum Forward Kinematics . . . 89
5.3 Parametrized Hyperplane Collision Certificate . . . 91
5.4 Certified Polytope Face Pushback . . . 98
5.5 Pendulum-on-Rail Robot . . . 100
5.6 Pendulum-on-Rail Region-Growth Progress . . . 102
5.7 Pinball Flipper Free-Space Cover . . . 103
5.8 Pinball Flipper Region-Growth Progress . . . 104
5.9 Pinball Flipper Certified Regions . . . 105
5.10 Seven-DOF IIWA Shelf Certificate . . . 105
5.11 IIWA Shelf Certified Region Growth . . . 106
5.12 Bimanual IIWA Certified Regions . . . 107
5.13 UR3e Shelf Certified Postures . . . 108
5.14 Dual-UR3e Certified Postures . . . 109
5.15 Trajectory Certification Environments . . . 110
5.16 Bimanual IIWA RRT Certification Timing . . . 111
5.17 Visibility Clique Cover Algorithm Sketch . . . 115
5.18 IRIS Directional Region Growth . . . 119
5.20 Maximum-Clique Hole Counterexample . . . 122
5.21 Finite-Sample Maximum-Clique Comparison . . . 122

6.1 TEASER++ Implied-Equality Ablation . . . 141

7.1 Maros–Meszaros QP Performance Profile . . . 160
7.2 Mittelmann SOCP Performance Profile . . . 160
7.3 Locomotion MPC Timing Distribution . . . 162
7.4 Penalty-Adaptation Iteration Ratios . . . 164
7.5 Penalty-Adaptation Time Ratios . . . 165
7.6 Acceleration Iteration Ratios . . . 166
7.7 Acceleration Time Ratios . . . 167

7.8 Composed-Enhancement Iteration Ratios . . . 168
7.9 Composed-Enhancement Time Ratios . . . 169
7.10 GCS Vertex-Family Performance Profiles . . . 171
7.11 GCS Vertex Linear-Solve Timing Breakdown . . . 172
7.12 Dense-Cut Matrix-Form Stress Test . . . 173
7.13 Large-Batch CUDA Timing Breakdown . . . 177
7.14 Maze CUDA Timing Breakdown . . . 179

8.1 Dense Matrix Equation Runtime Comparison . . . 193
8.2 Random Sparse Matrix Equation Comparison . . . 195
8.3 Banded Sparse Matrix Equation Comparison . . . 196
8.5 Dense SPD Diagonalization Timing . . . 198
8.6 Poset, Block Matrix, and Interval Poset . . . 200
8.7 Multitree and Non-Multitree Poset Examples . . . 203
8.8 Claw Graph Fill-In Under Two Orders . . . 205
8.9 Common-Descendant Sparsification Step . . . 207
8.10 Common-Descendant Adjacency Matrix . . . 208
8.11 Eliminated-System Graph Evolution . . . 210
8.12 Representative Multitree Benchmark Families . . . 212
8.13 Eliminated-System Nonzero Profiles . . . 213
8.14 Elimination Front-Size Profiles . . . 214
8.15 Poset Least-Squares Solve Times . . . 215

9.1 GCS Relaxation Variable Overlap . . . 218
9.2 Flow-Variable Consensus Split . . . 219
9.3 Spatial Edge-Variable Consensus Split . . . 220
9.4 Representative GCSBench Instance Families . . . 226
9.5 GCSBench Centralized Solver Performance Profile . . . 228
9.6 Edge-Regularized Centralized Solver Performance Profile . . . 230
9.7 Ratio of `VEGA` times to `CCosmo` times on `GcsBench` . . . 232
9.8 Ratio of idealized `VEGA` times to `CCosmo` times on `GcsBench` . . . 233
9.9 Ratio of outer iterations for `VEGA` and `CCosmo` . . . 233
9.10 Ratio of `VEGA` times to `CCosmo` times on edge-regularized `GcsBench` . . . 235
9.11 Ratio of idealized `VEGA` times to `CCosmo` times on edge-regularized `GcsBench` 236
9.12 Effect of edge regularization on `VEGA` solve time . . . 236

A.1 Cylinder Hyperplane-Separation Geometry . . . 269

B.1 Locomotion MPC Appendix Timing Traces . . . 278

List of Tables

1.1 Notation Summary . . . 43

5.1 Convex-Body Separation Parameterizations . . . 113
5.2 Visibility Clique Cover Benchmark Comparison . . . 120

6.1 Orthogonal Rotation Implied Equalities . . . 139
6.2 Special-Orthogonal Rotation Implied Equalities . . . 139
6.3 Pose-Estimation Implied Equalities . . . 140

7.1 Maros–Meszaros QP Solver Summary . . . 160
7.2 Mittelmann SOCP Solver Summary . . . 160
7.3 Maros–Meszaros Penalty-Ablation Summary . . . 163
7.4 KM/Halpern Acceleration-Ablation Summary . . . 166
7.5 Composition-Ablation Summary . . . 167
7.6 GCS Vertex Matrix-Program Structure . . . 170
7.7 GCS Vertex Matrix-Form Timing Comparison . . . 172
7.8 CUDA Local-Program Batch Performance . . . 176
7.9 Contact-Graph Local-Program Families . . . 178
7.10 Regularized Maze Local-Program Families . . . 178
7.11 Maze Local-Program Families . . . 179

8.1 Positive-Definite Pair Diagonalization . . . 189
8.2 Mixed-Definite Pair Diagonalization . . . 189
8.3 Semidefinite Pair Diagonalization . . . 190

9.1 GCSBench Centralized Solver Success Summary . . . 229
9.2 Edge-Regularization Runtime Comparison . . . 230
9.3 Edge-Regularization Accuracy Comparison . . . 231
9.4 Regularized GCS Solver Time Comparison . . . 234
9.5 Regularized Half-Step Time Comparison . . . 234

A.1 Algebraic Joint Parameterization . . . 266
A.2 SOS Conditions for Body Separation . . . 267
A.3 Rigidly Moving Convex-Body Membership . . . 268

B.1 Supported Cone and Set Blocks . . . 276
B.2 Shao CUDA Timing Decomposition . . . 279

B.3 Shao CUDA Readback Diagnostics 279
B.4 Tobia CUDA Timing Decomposition 280
B.5 Tobia CUDA Readback Diagnostics 280

C.2 GCSBench Graph-Object Instance Information 284
C.4 GCSBench Conic Relaxation Information 287
C.5 GCSBench Centralized Solve Times 290
C.6 GCSBench Centralized Cost and Status 291
C.7 Regularized GCSBench Centralized Solve Times 291
C.8 Regularized GCSBench Solve-Time Ratios 292
C.9 Regularized GCSBench Cost and Status 292
C.10 Iteration counts for original problems. 293
C.11 Solution quality on original problems. 294
C.12 Half-step iteration counts for original problems. 295
C.13 Half-step solution quality on original problems. 295
C.14 Regularized GCS Solver Iterations and Times 296
C.15 Regularized GCS Solver Solution Quality 297
C.16 Regularized Half-Step Iterations and Times 298
C.17 Regularized Half-Step Solution Quality 298

Introduction

Mathematical programming provides a unifying framework for expressing decision-making tasks as the minimization of an objective subject to constraints. It has emerged as a dominant paradigm for expressing problems in a wide range of engineering and computational fields including machine learning, finance, control, and robotics. Research into optimization methods that accurately solve these mathematical programs (also known as optimization models), has been a central topic in mathematics for centuries, associated with algorithms such as Newton's root-finding method [2], Cauchy's gradient descent method [3], and Lagrange's constrained formulation of mechanics [4]. A modern revolution of these algorithms began close to a century ago with the intense study of linear programming [5–8] and has remained an active area of research due to the introduction of interior point methods in the 1980-1990s [9–13]. While attractive as a general paradigm, in practice there is frequently a trade-off between mathematical programs that accurately model the task at hand and programs that can be solved either in theory or in practice.

Take, for example, a robot in a city that needs to find the shortest path between its current location and the nearest charging station. This problem can be modeled quite effectively as a shortest-path problem on a graph and can be solved in milliseconds on even enormous graphs using classic algorithms such as Dijkstra's, Bellman-Ford, or push-relabel [14–17]. In contrast, consider the same robot, in the same city, tasked with visiting as many intersections as possible before needing to return to the nearest charging station. This problem is readily modeled as finding the longest path in a graph, which is known to be NP-complete and therefore intractably difficult in general [6] [18].

In robotics, polynomial optimization programs (POP) appear quite naturally:

$$\begin{aligned} &\inf\ p(x) \\ &\text{subject to } g_i(x) \geq 0,\ i = 1, \ldots, N \\ &\qquad\qquad h_j(x) = 0,\ j = 1, \ldots, M \end{aligned} \tag{POP}$$

where p, g_i, and h_j are multivariate polynomials and $x \in \mathbb{R}^n$; kinematic relationships, dynamic constraints, contact models, and trajectory representations frequently give rise to polynomial expressions. Unfortunately, this class includes as special cases many hard problems such as mixed-integer linear programs[7] [18] and multivariate polynomial root-finding [19].

Despite the known theoretical difficulty of this class, a large body of research has shown that designing and solving *convex* approximations of (POP) can be an effective heuristic

[6] Unless one believes that $P = NP$

[7] The binary constraint $x \in \{0, 1\}$ is readily encoded as $x(1 - x) = 0$

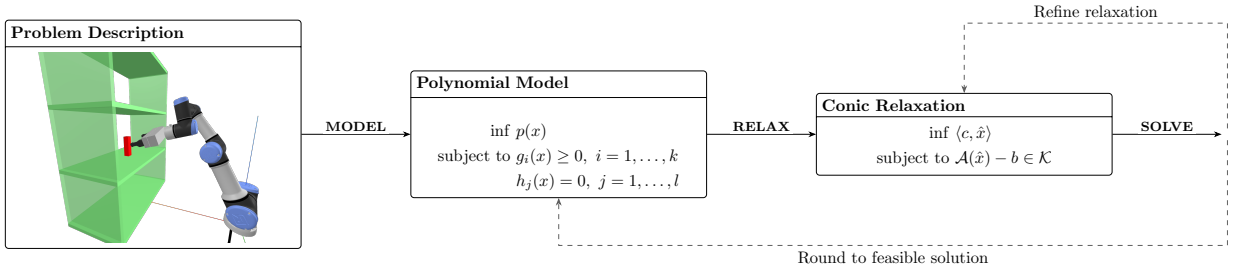


Figure 1: Using the Sums-of-Squares procedure to solve challenging problems involves three steps. First, the problem must be converted into a polynomial model. Next, we form the convex relaxation of the problem. While theoretically this is fully algorithmic, in practice a trade off between theoretical performance and computational efficiency must be made. Third, we must actually solve the convex optimization programs. Finally, the solution of this convex relaxation can either be rounded to a feasible solution or the convex relaxation can be strengthened and the loop repeated. This thesis explores the Model, Relax, and Solve steps of this procedure.

for producing solutions of satisfactory quality. Convex or conic optimization programs are a particularly important subclass of mathematical programs of the form

$$\begin{aligned} &\inf\ c^T x \\ &\text{subject to } Ax - b \in \mathcal{K} \end{aligned} \tag{Conic}$$

where $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, and $\mathcal{K} \subseteq \mathbb{R}^m$ is a *convex cone*. Such programs can be solved efficiently to global optimality at scale [9,20–23].

The Sums-of-Squares (SOS) hierarchy provides a principled mechanism for constructing convex relaxations of (POP) whose solutions converge to the true optimum under mild conditions [24–27]. Even low levels of the hierarchy have proven a useful tool in robotics, being used for region of attraction estimation, safety verification, perception, and trajectory optimization [28–35].

The process of using the SOS hierarchy to solve a problem is presented in Figure 1. The journey begins by first converting the problem at hand into a polynomial optimization program. This step can be non-trivial but if successful can be a worthwhile endeavor as it enables the use of a very powerful computational tool. Once the polynomial model has been constructed, the SOS hierarchy prescribes how to form its convex relaxation. However, this prescription is very expensive, frequently leading to programs with impractical solve times. Effectively trading off between theoretical strength and practical performance is the second step of the process.

With the convex model in hand, we can now turn our attention to solving the program. While general-purpose solution methods exist, SOS relaxations contain both substantial structure and substantial degeneracies [36]. In the pursuit of greater performance, it can be worthwhile to pursue tailored solvers for the particular convex relaxation at hand.

Finally, given a relaxed solution we have two possibilities. The first is that the relaxed solution is satisfactory in some manner; it is either the globally optimal solution or it can be rounded to a satisfactory approximate solution of our original program. If this is not the

case, we can use the solution to further tighten the convex relaxation, continuing the loop until we are satisfied with our answer. When solving mixed-integer programs, this latter step takes the form of the familiar branch-and-cut algorithm.

In this thesis, we study the practical use of convex relaxations for polynomial optimization problems in robotics by examining three parts of the pipeline in Figure 1: modeling the original problem, designing an effective relaxation, and solving the resulting convex program. The contributions are organized around the observation that the usefulness of the SOS hierarchy depends not only on its theoretical convergence guarantees, but also on the quality of the polynomial model, the strength and size of the relaxation, and the numerical methods used to solve it.

In Part I of this thesis, we collect the necessary background upon which the results of this thesis are based. We begin by collecting our notation and elementary background in Chapter 1. In Chapter 2, we review two different ways relaxing polynomial optimization programs: the Sums-of-Squares method and Tensor Product (or Moment) Relaxations (TPR). These are connected to each other by convex duality. We also review a slight generalization of TPR beyond polynomial programs.

In Chapter 3, we review the Graph Of Convex Sets (GCS) framework introduced in [1,37,38]. GCS considers an optimization problem over a graph where every vertex and edge is paired with a convex program. GCS extends the powerful modelling framework of graph optimization programs such as shortest path and travelling salesman to handle continuous decisions while navigating the graph, and has become increasingly popular in robotics. The authors of [37,38] develop a sparse TPR of the GCS problem that is highly effective in practice. We review this relaxation in the language of TPRs. Effective use of GCS motivates some developments in our first two chapters of contributions. Efficiently solving this TPR program motivates the second half of our contributions in Part III.

Finally, Chapter 4 briefly reviews the alternating direction method of multipliers (ADMM). ADMM is a very general, classic algorithm introduced by [39] for solving composite convex optimization that has seen an explosion of popularity in recent years. The simplicity of ADMM makes it an appealing base algorithm to specialize to different structures of optimization programs, and we use these properties to design efficient and scalable solvers.

After covering this essential background, we move on to our contributions. Part II focuses on the modeling and relaxation steps of the framework in Figure 1. In Chapter 5, we develop a polynomial optimization formulation for certifying collision-free subsets of a robot's configuration space. This formulation allows us to use SOS methods to provide the first efficient method for certifying full-dimensional regions of a robot's configuration contains no collisions. Specializing our method to trajectories further improves the practicality of our method, allowing us to rigorously certify non-collision of polynomial motion plans in just under one second. We additionally provide a method for growing large, convex, collision-free regions in a polynomial reparametrization of the configuration space. Armed with a method for growing convex regions, we conclude Chapter 5 with a heuristic method for automatically choosing the seed location of these convex regions to provide an efficient convex cover of the collision-free configuration space.

Chapter 6 in Part II studies the design of stronger SOS relaxations. Because SOS relaxations can be highly sensitive to the formulation of the underlying polynomial program, we introduce a lightweight preprocessing method that uses linear algebra to discover additional

redundant equality constraints for the original POP. Importantly, our method only returns the subset of new equalities which do not change the original POP and also can plausibly strengthen the SOS relaxation. We demonstrate that our method efficiently recovers known strengthened relaxations from the robotics literature, finding the strengthened formulation in about 1% of the time it takes to solve the SOS relaxation. This shows that the procedure can strengthen the resulting relaxation while preserving a computational profile suitable for robotics pipelines that require time-sensitive solves.

Part III develops solver infrastructure for the convex programs produced by the convex relaxation procedure. In Chapter 7, we introduce `CCosmo`, a family of `C++` implementations of a popular ADMM algorithm for solving general convex optimization programs introduced in [23,40]. `CCosmo` supports convex programs with quadratic costs, includes a specialized implementation for a matrix standard form that appears naturally in SOS and TPR relaxations, and provides a GPU implementation for solving large batches of small convex programs. When solving sparse programs in standard form, our implementation is on-par with other popular, open-source solvers including `OSQP`, `SCS`, and `COSMO.jl`. When solving matrix-structured programs, `CCosmo` is signficantly faster, and the batched implementation can solve large batches over $8\times$ faster.

Next, in Chapter 8 we study algorithms for solving matrix linear systems, a core numerical subroutine in many convex optimization algorithms.

We first give a direct method for solving positive semidefinite generalized Sylvester equations. Solving such systems is the bottleneck in the matrix-standard form implementation of `CCosmo` and may be of independent interest. Our method scales with the largest dimension in the unknown matrix rather than scaling in the product of the dimensions as a standard solver would. The resulting solver is between $10 - 1000\times$ faster than a standard solver, and is between $5 - 10$ times faster than a matrix solver that does not account for the PSD structure of the equations.

Next, we consider a specific linear system that arises in the context solving a sparse, matrix least-squares problem. The sparsity constraint in the problem is given combinatorially as a partially ordered set (poset). Such problems arise naturally in the context of designing controllers for networked linear systems but may also arise in other contexts. We describe a natural elimination order for solving the linear system that is related to the poset describing its sparsity pattern. The elimination order goes beyond the typical theory of sparse linear algebra, resulting in an elimination order that exhibits both low-fill and successively sparsifies the system. The resulting solver is $2-5\times$ faster than a generic sparse solver under a heuristic fill-reducing order.

Part III culminates in Chapter 9, which describes a decomposition based solver for the convex relaxation of GCS programs [1,37]. GCS programs are a practically relevant class of TPRs, and the convex relaxation inherits substantial structure from both the original graph-based formulation of the program and the actual relaxation procedure itself. Similar to other TPRs, GCS relaxations can grow very large very quickly, becoming intractable to solve as a general convex program.

To overcome this limitation, in Chapter 9 we introduce `VEGA`, a decomposition-based first-order solver that separates the relaxation into the natural pieces of a GCS: vertex programs, edge programs, and a classical graph optimization problem. `VEGA` solves these pieces in parallel while driving them to consensus. Each step of `VEGA` will amount to solving a conic

program with a quadratic cost; the vertex programs are naturally in matrix standard form and the edge and graph programs are naturally in standard-form. To exploit this diverse structure and to amortize the cost of solving the subproblems via warm-starting, `VEGA` is built on top of `CCosmo` described in Chapter 7, which in turn is built on top of the developments in Chapter 8.

In addition to `VEGA`, Chapter 9 also introduces `GcsCc`, a `C++` based library with `Python` bindings for modeling GCS programs. `GcsCc` supports specifying programs directly in standard-form, making it trivial to serialize and deserialize instances. This allows us to distribute a family of diverse GCS instances from the robotics literature as the `GcsBench` benchmark.

`VEGA` is tested on the `GcsBench` problems. Compared to generic convex optimization solvers, `VEGA` is slower on the smaller `GcsBench` instances, comparable on the moderately sized instances, and is the only solver capable of making any progress and indeed solving the larger instances. Therefore, `VEGA` is built on top of `CCosmo`, which can adapt to the structure of the programs and al

Taken together, this thesis contributes methods across the full convex-relaxation pipeline, from polynomial modeling and relaxation design to scalable numerical solution methods for robotics optimization.

Part I

Background

Chapter 1

Elementary Background

In this chapter, we collect some elementary concepts from a range of fields primarily to organize our notation. The sections are organized from most elementary to least elementary, with references to more thorough treatment of the subject noted in each section.

Throughout, we will use the standard notation of $\mathbb{N}$ for the natural numbers (including 0), $\mathbb{Z}$ for the integers, $\mathbb{Q}$ for the rationals, $\mathbb{R}$ for the reals, and $\mathbb{C}$ for the complex numbers.

1.1 Indexing notation

Throughout, we use the notation

$$[n] = \{1, 2, \ldots, n\}.$$

The size of an index set $\mathcal{J} \subseteq [n]$ with m elements is denoted by $|\mathcal{J}| = m$. Index sets are assumed *ordered* unless otherwise stated.

Given a vector $x \in \mathbb{R}^n$ and an index set $\mathcal{J} \coloneqq \{i_1, \ldots, i_m\} \subseteq [n]$, we denote by $x_{\mathcal{J}} \coloneqq [x_{i_1}, \ldots, x_{i_m}]$ or $x[\mathcal{J}] \coloneqq x_{\mathcal{J}}$ depending on the context. Given a matrix $A \in \mathbb{R}^{m \times n}$ and index sets $\mathcal{J} \subseteq [m]$ and $\mathcal{K} \subseteq [n]$, we denote the corresponding submatrix by $A_{\mathcal{J},\mathcal{K}}$ or $A[\mathcal{J}, \mathcal{K}]$ depending on the context.

1.2 Sets and Vector Spaces

Given sets $\mathcal{S}$ and $\mathcal{T}$ we denote

- The interior of $\mathcal{S}$ as $\text{int}(\mathcal{S})$.
- The closure of $\mathcal{S}$ as $\text{cl}(\mathcal{S})$.
- The linear combination (Minkowski sum) of sets $\alpha\mathcal{S} + \beta\mathcal{T} \coloneqq \{z \mid z = \alpha x + \beta y,\ x \in \mathcal{S},\ y \in \mathcal{T}\}$

Throughout this thesis, we use the standard notions of rings, modules, fields and vector spaces.

Given a vector space $\mathbb{V}$ over some field $\mathbb{F}$, we denote the dual space as $\mathbb{V}^*$. When we make reference to the dual space, we assume an unambiguous duality pairing $\langle \cdot, \cdot \rangle : \mathbb{V}^* \times \mathbb{V}$.

Given a linear map between two vector spaces $\mathcal{A} : \mathbb{V} \to \mathbb{W}$, we denote its adjoint $\mathcal{A}^* : \mathbb{W}^* \to \mathbb{V}^*$ which is the unique linear map satisfying

$$\langle y, \mathcal{A}(x) \rangle = \langle \mathcal{A}^*(y), x \rangle.$$

The majority of real vector spaces $\mathbb{V}$ encountered in this thesis have finite dimension n. In this case, we frequently conflate $\mathbb{V}$ with the canonically isomorphic vector space $\mathbb{R}^n$.

Given a matrix $X \in \mathbb{R}^{m \times n}$ we denote by $\mathbf{vec}(X)$ the vertical concatenation of the columns of X into a single vector $x \in \mathbb{R}^{mn}$ and conversely, we'll denote $\mathbf{mat}_{mn}(x)$ as the column-major unrolling of x into a matrix X.

We denote

- The set of diagonal matrices $\mathbb{D}^n$.
- The set of symmetric matrices $\mathbb{S}^n$.

We use $\|v\|$ to denote the norm of a vector $v \in \mathbb{V}$. For $v \in \mathbb{R}^n$, common norms we use are

- The ℓ^2-norm: $\|v\|_2 = \sqrt{v^T v}$
- The ℓ^∞-norm: $\|v\|_\infty = \max |v_i|$
- The Q-norm: $\|v\|_Q = \sqrt{v^T Q v}$ where Q is a positive definite matrix. If Q is only semidefinite then this is only a seminorm.

For matrices $V \in \mathbb{R}^{n \times n}$ we use the norms:

- Frobenius norm: $\|V\|_F = \|\mathbf{vec}(V)\|_2$
- Spectral norm: $\|V\|_2 = \sigma_{\max}(V)$ where $\sigma_{\max}$ denotes the maximum singular value.

When the choice of norm is clear, we drop the subscript. For further review, the reader is directed to any standard elementary linear algebra book, e.g. [41].

1.3 Elementary Convex Analysis

Convexity plays a central role throughout this thesis, and so we begin by introducing the elementary concepts of this field. The standard references for most of these notions and results can be found in the classic text [20], with [42–44] being popular, modern references.

Definition 1 (Affine Set)**.** *A set $\mathcal{A}$ is an* affine set *if the entire line passing between two points in $\mathcal{A}$ is contained in $\mathcal{A}$*

$$x,\ y \in \mathcal{A}, \lambda \in \mathbb{R} \implies \lambda x + (1 - \lambda) y \in \mathcal{A}.$$

Alternatively, an affine set is a set satisfying

$$\mathcal{A} = \lambda \mathcal{A} + (1 - \lambda) \mathcal{A},$$

for every $\lambda \in \mathbb{R}$.

A similar idea is that of a convex set, which requires only that the segment between the two points be contained in the set.

Definition 2 (Convex Set)**.** *A set $\mathcal{C}$ is* convex *if the line segment between two points in $\mathcal{C}$ is also contained in $\mathcal{C}$.*

$$x, y \in \mathcal{C},\ \lambda \in [0,1] \implies \lambda x + (1-\lambda)y \in \mathcal{C}.$$

Alternatively, a convex set is a set satisfying

$$\mathcal{C} = \lambda\mathcal{C} + (1-\lambda)\mathcal{C},$$

for every $\lambda \in [0,1]$.

The most important example of a convex set we encounter in this thesis is a cone.

Definition 3 (Cone)**.** *A set $\mathcal{K}$ is a* cone *if it is invariant under nonnegative scaling.*

$$x \in \mathcal{K},\ \lambda \geq 0 \implies \lambda x \in \mathcal{K}.$$

Alternatively, $\mathcal{K}$ is a cone if it satisfies

$$\lambda\mathcal{K} \subseteq \mathcal{K},$$

for all $\lambda \geq 0$. A cone is called pointed *if $\mathcal{K} \cap -\mathcal{K} = \{0\}$ and* solid *if int$(\mathcal{K}) \neq \emptyset$. A closed, pointed, solid, and convex cone is called a* proper cone

The most important elementary sets and cones are

- The vector space $\mathbb{R}^n$. This is a cone, but not a proper cone.
- The zero set $\mathbb{0}^n \coloneqq \{0 \in \mathbb{R}^n\}$. This is a cone, but not a proper cone.
- The nonnegative orthant $\mathbb{R}^n_+ \coloneqq \{x \in \mathbb{R}^n \mid x \geq 0\}$. This is a proper cone. We denote its interior of positive vectors $\mathbb{R}^n_{++} \coloneqq \{x \in \mathbb{R}^n \mid x > 0\}$.
- The Lorentz Cone $\mathbb{L}^n \coloneqq \{(t,x) \mid t \in \mathbb{R},\ x \in \mathbb{R}^{n-1},\ t \geq \|x\|_2\}$. This is a proper cone.
- The positive semidefinite cone $\mathbb{S}^n_+ \coloneqq \{X \in \mathbb{S}^n \mid x^T X x \geq 0,\ \forall x \in \mathbb{R}^n\}$. This is a proper cone. We denote its interior, the positive definite matrices, as $\mathbb{S}^n_{++} \coloneqq \{X \in \mathbb{S}^n \mid x^T X x > 0,\ \forall x \neq 0 \in \mathbb{R}^n\}$. We denote membership in this cone as $X \succeq 0$ and its interior as $X \succ 0$.
- The nonnegative diagonal matrices $\mathbb{D}^n_+ \coloneqq \{X \in \mathbb{D}^n \mid X_{ii} \geq 0,\ i \in [n]\}$ and its interior $\mathbb{D}^n_{++} \coloneqq \{X \in \mathbb{D}^n \mid X_{ii} > 0,\ i \in [n]\}$. Both are subsets of $\mathbb{S}^n_+$.

Based on these definitions, we can define natural notions of affine, convex, and conic combinations of points.

Definition 4 (Affine, Convex, and Conic Combinations)**.** *Given a set of points $x_1, \ldots, x_k \in \mathbb{R}^n$ and scalars $\lambda_1, \ldots, \lambda_k$ the linear combination*

$$\sum_{i=1}^{k} \lambda_i x_i$$

is called an

1. *Affine combination if $\sum_{i=1}^k \lambda_i = 1$*
2. *Conic combination if $\lambda_i \geq 0$*
3. *Convex combination if $\lambda_i \geq 0$ and $\sum_{i=1}^k \lambda_i = 1$.*

These definitions let us define the affine, convex, and conic hulls of a set

Definition 5 (Affine, Convex, and Conic Hull)**.** *The affine, convex, and conic hulls of a set $\mathcal{S}$ are the smallest affine, convex, and conic sets containing $\mathcal{S}$. They can be written as*

1. $\textbf{\textit{aff}}(\mathcal{S}) \coloneqq \{y = \sum_i \lambda_i x_i \mid \ x_i \in \mathcal{S},\ \sum_{i=1} \lambda_i = 1\}$.
2. $\textbf{\textit{conic}}(\mathcal{S}) \coloneqq \{y = \sum_i \lambda_i x_i \mid \ x_i \in \mathcal{S},\ \lambda_i \geq 0\}$.
3. $\textbf{\textit{conv}}(\mathcal{S}) \coloneqq \{y = \sum_i \lambda_i x_i \mid \ x_i \in \mathcal{S},\ \lambda_i \geq 0,\ \sum_i \lambda_i = 1\}$.

1.3.1 Support Function, Dual, and Recession Cones

The following sets are important in several cases, particularly when we discuss convex programming.

The recession cone of a set $\mathcal{S}$ is the set of unbounded directions (with zero).

Definition 6 (Recession Cone)**.** *The recession cone of a set $\mathcal{S}$ is the set of all vectors defining unbounded directions in $\mathcal{S}$.*

$$\mathcal{S}^\infty \coloneqq \{y \in \mathbb{R}^n \mid x + \lambda y \in \mathcal{S},\ \forall x \in \mathcal{S},\ \lambda \geq 0\}.$$

We always have that $0 \in \mathcal{S}^\infty$, and so we call this the trivial recession-direction.

Compact sets have no non-trivial recession directions, while proper cones are their own recession cone.

Finally, we define the dual cone

Definition 7 (Dual Cone)**.** *Given a set $\mathcal{S} \subseteq \mathbb{V}$ and duality pairing $\langle \cdot, \cdot \rangle$, we define the dual cone*

$$\mathcal{S}^* \coloneqq \{y \in \mathbb{V}^* \mid \langle y, x \rangle \geq 0,\ \forall x \in \mathcal{S}\}.$$

As the name suggests, the dual is always a cone. Moreover, one can show that $(\mathcal{S}^*)^* = \text{cl}(\textbf{conic}(\mathcal{S}))$.

Finally, we define the support function of a convex set.

Definition 8 (Support Function)**.** *Given a convex set $\mathcal{S} \subseteq \mathbb{V}$ and $y \in \mathbb{V}^*$, the support function is*

$$\sigma_{\mathcal{S}}(y) = \sup_{x \in \mathcal{S}} \langle y, x \rangle.$$

1.4 Tensor Products and Matrix Products

We make frequent use of the tensor product in this thesis, so we briefly collect some properties and define some useful matrix products as well. This material can be found in e.g. [45, §14] or in more abstract form in [46, §10.4]

We begin by recalling the definition of multilinear maps.

Definition 9 (Multilinear and Symmetric Multilinear Maps)**.** *Let $\mathbb{V}_1, \ldots, \mathbb{V}_k, \mathbb{W}$ be vector spaces. A map*

$$\mathcal{A} : \mathbb{V}_1 \times \cdots \times \mathbb{V}_k \to \mathbb{W}$$

is k-multilinear *if it is linear in each argument while the other arguments are fixed. If $\mathbb{V}_1 = \cdots = \mathbb{V}_k = \mathbb{V}$, then $\mathcal{A}$ is called* symmetric *when*

$$\mathcal{A}(x_1, \ldots, x_k) = \mathcal{A}(x_{\pi(1)}, \ldots, x_{\pi(k)}),$$

for every permutation π of $[k]$.

A map which is 2-multilinear is called bilinear. We next recall the abstract definition of the tensor product of two vector spaces.

Definition 10 (Tensor Product (Abstract))**.** *The* tensor product *of two vector spaces $\mathbb{V}$ and $\mathbb{W}$ (over the same field) is a vector space $\mathbb{V} \otimes \mathbb{W}$ equipped with an associated bilinear map $\phi : (v, w) \to v \otimes w$ such that for every bilinear map $h : \mathbb{V} \times \mathbb{W} \to \mathbb{I}$ to a vector space $\mathbb{I}$ there is a unique linear map $\tilde{h} : \mathbb{V} \otimes \mathbb{W} \to \mathbb{I}$ such that $h(v, w) = (\tilde{h} \circ \phi)(v, w)$.*

An equivalent, more concrete definition of the tensor product can be stated in terms of bases.

Definition 11 (Tensor Product Space (Bases))**.** *Let $\mathbb{V}$ and $\mathbb{W}$ be finite-dimensional vector spaces over the same field, with ordered bases $v_1, \ldots, v_m$ and $w_1, \ldots, w_n$ respectively. The* tensor product space *$\mathbb{V} \otimes \mathbb{W}$ is the mn-dimensional vector space formed by all formal linear combination of the basis elements $v_i \otimes w_j$.*

For any two vectors $x \coloneqq \sum_{i=1}^{m} x_i v_i \in \mathbb{V}$ and $y \coloneqq \sum_{j=1}^{n} y_j w_j \in \mathbb{W}$, the tensor product is an element in $\mathbb{V} \otimes \mathbb{W}$ given by the bilinear expression:

$$x \otimes y \coloneqq \left(\sum_{i=1}^{m} x_i v_i \right) \otimes \left(\sum_{j=1}^{n} y_j w_j \right) = \sum_{i=1}^{m} \sum_{j=1}^{n} (x_i y_j)(v_i \otimes w_j).$$

Given dual vector spaces $\mathbb{V}^*$ and $\mathbb{W}^*$, the natural dual to $\mathbb{V} \otimes \mathbb{W}$ is given by the tensor product of the dual spaces $(\mathbb{V} \otimes \mathbb{W})^* \cong \mathbb{V}^* \otimes \mathbb{W}^*$. It inherits natural induced duality pairing.

Definition 12 (Induced Duality Pairing on Tensors)**.** *Let $v_1 \otimes w_1 \in \mathbb{V}^* \otimes \mathbb{W}^*$ and $v_2 \otimes w_2 \in \mathbb{V} \otimes \mathbb{W}$ and let $(\mathbb{V}, \mathbb{V}^*)$ and $(\mathbb{W}, \mathbb{W}^*)$ have duality pairing $\langle \cdot, \cdot \rangle_{\mathbb{V}}$ and $\langle \cdot, \cdot \rangle_{\mathbb{W}}$. The operation*

$$\langle v_1 \otimes w_1, v_2 \otimes w_2 \rangle_{\mathbb{V} \otimes \mathbb{W}} \coloneqq \langle v_1, v_2 \rangle_{\mathbb{V}} \langle w_1, w_2 \rangle_{\mathbb{W}}$$

extended linearly to all elements $y \in \mathbb{V}^* \otimes \mathbb{W}^*$ and $x \in \mathbb{V} \otimes \mathbb{W}$ is the induced duality pairing between $(\mathbb{V} \otimes \mathbb{W}, \mathbb{V}^* \otimes \mathbb{W}^*)$

Finally, we define the concept of tensor products of linear maps

Definition 13 (Tensor Product of Linear Maps)**.** *Let $f : \mathbb{V} \to \mathbb{U}_1$ and $g : \mathbb{W} \to \mathbb{U}_2$ be linear maps. The tensor product $f \otimes g : \mathbb{V} \otimes \mathbb{W} \to \mathbb{U}_1 \otimes \mathbb{U}_2$ is the unique linear map satisfying*

$$(f \otimes g)(v \otimes w) = f(v) \otimes g(w).$$

The tensor product of k vector spaces is defined analogously by considering k-multilinear map rather than bilinear maps The tensor product is the natural linearization of multilinearity. Every k-multilinear map

$$\mathcal{A} : \mathbb{V}_1 \times \cdots \times \mathbb{V}_k \to \mathbb{W}$$

is uniquely associated with a linear map $\tilde{\mathcal{A}}$ on the tensor product

$$\tilde{\mathcal{A}} : \mathbb{V}_1 \otimes \cdots \otimes \mathbb{V}_k \to \mathbb{W}, \qquad \tilde{\mathcal{A}}(x_1 \otimes \cdots \otimes x_k) = \mathcal{A}(x_1, \ldots, x_k).$$

We denote the k-fold tensor product of the same vector space with itself as

$$\mathbb{V}^{\otimes k} \coloneqq \mathbb{V} \underbrace{\otimes \ldots \otimes}_{k \text{ times}} \mathbb{V} = \mathbf{span}\left\{x_1 \otimes \ldots \otimes x_k \mid x_i \in \mathbb{V}\right\}.$$

1.4.1 Symmetric Tensors

Frequently, we are interested in taking the k-fold tensor product of a vector with itself:

$$x^{\otimes k} \coloneqq x \underbrace{\otimes \ldots \otimes}_{k \text{ times}} x.$$

We call $x^{\otimes k}$ a *symmetric tensor*.

This operation generates a subspace of $\mathbb{V}^{\otimes k}$ known as the symmetric tensor product space.

Definition 14 (Symmetric Tensor Product Space)**.** *Let $\mathbb{V}$ be a vector space. The k-fold* symmetric tensor product *of $\mathbb{V}$ is defined as*

$$\boldsymbol{Sym}^k(\mathbb{V}) \coloneqq \left\{ y = \sum_i \lambda_i x_i^{\otimes k} \mid x_i \in \mathbb{V}, \lambda_i \in \mathbb{R} \right\}.$$

Equivalently, $\mathbf{Sym}^k(\mathbb{V})$ is the subspace of tensors invariant under permutations of tensor factors. That is, $y \in \mathbf{Sym}^k(\mathbb{V})$ if and only if

$$P_\pi y = y, \quad \forall \pi \text{ permutation of } [k],$$

where P_π is the linear map defined on pure tensors by

$$P_\pi(x_1 \otimes \cdots \otimes x_k) \coloneqq x_{\pi(1)} \otimes \cdots \otimes x_{\pi(k)}.$$

Given two tensors $T_1 \in \mathbb{V}^{\otimes i}$ and $T_2 \in \mathbb{V}^{\otimes j}$, we denote by $T_1 \otimes_{\text{sym}} T_2 \in \mathbf{Sym}^{i+j}(\mathbb{V})$ the (normalized) symmetric part of the tensor generated by $T_1 \otimes T_2$.

1.4.2 Matrix Products

As every linear map can naturally be written as a matrix, it is natural to construct tensor products of linear maps from matrices. The Kronecker product allows us to do this.

Definition 15 (Kronecker Product)**.** *Let $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{p\times q}$. Then*

$$A \otimes B \coloneqq \begin{bmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \dots & a_{mn}B \end{bmatrix}$$

is a $mp \times nq$ matrix.

The Kronecker product and the tensor product share the same symbol, as a basis for the tensor product between two vector spaces can be assembled by taking all pairwise Kronecker products of bases between the two vector spaces. Similarly, the tensor product of two linear maps in a particular basis is given by the tensor product of their matrix representations. Therefore, we freely conflate the Kronecker product vectors with the tensor product of vector spaces, and the Kronecker product of matrices with the tensor product of the linear maps of two vector spaces.

The Kronecker product has the following properties [47, §4.2].

Theorem 1 (Kronecker Product Properties)**.** *Let A, B, C, D all be matrices associated to linear operators.*

1. ***Bilinearity:*** *$(\alpha A) \otimes B = \alpha(A \otimes B)$ and $A \otimes (\beta B) = \beta(A \otimes B)$.*
2. ***Associativity:*** *$(A \otimes B) \otimes C = A \otimes (B \otimes C)$.*
3. ***Distributivity:*** *$A\otimes(B+C) = (A\otimes B)+(A\otimes C)$ and $(A+B)\otimes C = (A\otimes C)+(B\otimes C)$.*
4. ***Permutative Commutativity:*** *In general $A\otimes B \neq B\otimes A$, but there exist permutation matrices P and Q called the perfect shuffle matrices such that $A \otimes B = P(B \otimes A)Q$. If A and B are square, then $P = Q^T$.*
5. ***Mixed Product Property:*** *$(A \otimes B)(C \otimes D) = AC \otimes BD$ provided the matrices are conformal.*
6. ***Distributivity of Operations:*** *The following operations distribute*

$$(A \otimes B)^* = A^* \otimes B^*,$$
$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1},$$

   where the inverse identity holds assuming the inverses exist.

7. ***Vectorization Identities:*** *The following hold:*

$$\boldsymbol{vec}(AXB) = (B^T \otimes A)\,\boldsymbol{vec}(X), \tag{1.1}$$
$$\boldsymbol{vec}(xy^T) = y \otimes x. \tag{1.2}$$

1.4.3 Tensor Products of Cones, Separable Matrices, and Positive Maps

In the prior section we defined tensor products of vector spaces. We also find it useful to define the tensor product of two cones. In what follows, let $\mathcal{K}_1 \subseteq \mathbb{R}^m$ and $\mathcal{K}_2 \subseteq \mathbb{R}^n$.

Definition 16 ((Minimal) Tensor Product of Cones [48])**.** *Let $\mathcal{K}_1$ and $\mathcal{K}_2$ be two convex cones. We define their* tensor product *as:*

$$\mathcal{K}_1 \otimes \mathcal{K}_2 \coloneqq \boldsymbol{conv}\{x_1 \otimes x_2 \mid x_1 \in \mathcal{K}_1, x_2 \in \mathcal{K}_2\}.$$

This allows us to define the notion of separable matrix.

Definition 17 (Separable Matrix)**.** *A matrix $X \in \mathbb{R}^{m\times n}$ is said to be $\mathcal{K}_1 \otimes \mathcal{K}_2$ separable if $X \in \mathcal{K}_1 \otimes \mathcal{K}_2$. Concretely, X is separable if it can be written as:*

$$X = \sum_i \sum_j \lambda_{ij}(y_i z_j^T),$$

with $\lambda_{ij} \geq 0,\ y_i \in \mathcal{K}_1,\ z_j \in \mathcal{K}_2$. When the cones are clear, we simply say that X is separable.

A vector $x \in \mathbb{R}^{mn}$ is separable in the same manner if

$$x = \sum_i \sum_j \lambda_{ij}(z_j \otimes y_i).$$

A related notion of the separable matrices are the positive maps.

Definition 18 (Positive Maps)**.** *A linear map $\mathcal{L} : \mathbb{R}^m \to \mathbb{R}^n$ is called $(\mathcal{K}_1, \mathcal{K}_2)$-positive if $\mathcal{L}(\mathcal{K}_1) \subseteq \mathcal{K}_2$. The set of $(\mathcal{K}_1, \mathcal{K}_2)$ positive maps is a convex cone and is denoted by $\mathfrak{P}(\mathcal{K}_1, \mathcal{K}_2)$. When the cones $\mathcal{K}_1$ and $\mathcal{K}_2$ are clear from context, we drop all mention of them and simply write $\mathfrak{P}$ and call the maps "positive".*

The two notions are duals of each other.

Theorem 2 (Duality of Separable Matrices and Positive Maps)**.** *Let $\mathcal{K}_1 \subseteq \mathbb{R}^m$ and $\mathcal{K}_2 \subseteq \mathbb{R}^n$ be two convex cones. Then*

$$(\mathcal{K}_1 \otimes \mathcal{K}_2)^* = \mathfrak{P}(\mathcal{K}_1, \mathcal{K}_2^*).$$

Proof 1.1. Let $\mathcal{L} : \mathbb{R}^m \to \mathbb{R}^n$, $x \in \mathcal{K}_1$, and $y \in \mathcal{K}_2$. We use the standard duality pairings $\langle z_1, z_2\rangle = z_2^T z_1$ for $z_1, z_2 \in \mathbb{R}^l$ and $\langle Z_1, Z_2\rangle = \mathbf{tr}(Z_2^T Z_1)$ when $Z_1 \in \mathbb{R}^{l\times k}$ and $Z_2 \in \mathbb{R}^{l\times k}$. After choosing a basis for $\mathbb{R}^n$ and $\mathbb{R}^m$, there is a unique matrix $L \in \mathbb{R}^{n\times m}$ satisfying $Lx = \mathcal{L}(x)$.

We therefore have the following chain

$$\begin{aligned}
\mathcal{L} \in \mathfrak{P}(\mathcal{K}_1, \mathcal{K}_2^*) &\iff \mathcal{L}(x) \in \mathcal{K}_2^*,\ \forall x \in \mathcal{K}_1 \\
&\iff \langle \mathcal{L}(x), y\rangle \geq 0, \forall x \in \mathcal{K}_1, y \in \mathcal{K}_2 \\
&\iff \langle Lx, y\rangle \geq 0, \forall x \in \mathcal{K}_1, y \in \mathcal{K}_2 \\
&\iff \langle L, yx^T\rangle \geq 0, \forall x \in \mathcal{K}_1, y \in \mathcal{K}_2 \\
&\iff \langle \mathbf{vec}(L), x \otimes y\rangle \geq 0, \forall x \in \mathcal{K}_1, y \in \mathcal{K}_2 \\
&\iff \mathcal{L} \in (\mathcal{K}_1 \otimes \mathcal{K}_2)^*.
\end{aligned}$$

□

The $\mathcal{K}_1 \otimes \mathcal{K}_2$ is technically the *minimal* tensor product in the literature. For convex cones, we also define the *maximal tensor product* [48].

Definition 19 (Maximal Tensor Products)**.** *The* maximal tensor product *of cones $\mathcal{K}_1$ and $\mathcal{K}_2$ is defined as*

$$\mathcal{K}_1 \otimes_{max} \mathcal{K}_2 \coloneqq (\mathcal{K}_1^* \otimes \mathcal{K}_2^*)^*,$$

where the tensor product on the right-hand side is the minimal tensor product from above.

The minimal and maximal tensor product of cones are in general different, however they are known to coincide if at least one of the cones is isomorphic to the positive orthant.

Theorem 3 ([48, Theorem A])**.** *For finite-dimensional proper cones $\mathcal{K}_1$ and $\mathcal{K}_2$,*

$$\mathcal{K}_1 \otimes \mathcal{K}_2 = \mathcal{K}_1 \otimes_{max} \mathcal{K}_2,$$

if and only if at least one of $\mathcal{K}_1$ and $\mathcal{K}_2$ is isomorphic to $\mathbb{R}^k_+$ where k is of appropriate dimension.

Remark 1 (Positive Maps and Maximal Tensor Products)**.** *Maximal tensor product, minimal tensor products, are very closely related. Applying Theorem 2 to $\mathcal{K}_1^*$ and $\mathcal{K}_2^*$ gives*

$$\mathcal{K}_1 \otimes_{max} \mathcal{K}_2 = \mathfrak{P}(\mathcal{K}_1^*, \mathcal{K}_2^{**}) = \mathfrak{P}(\mathcal{K}_1^*, cl(\mathcal{K}_2)).$$

Thus, positive maps can be viewed either as duals of minimal tensor products or, equivalently, as maximal tensor products after dualizing the first cone.

In general, describing the tensor product of two cones can be challenging. For example, checking membership in $\mathbb{S}^m_+ \otimes \mathbb{S}^n_+$ is known to be NP-hard [49]. However, in the case that one of the cones $\mathcal{K}_i$ is either $\mathbb{R}^n_+$ or $\mathbb{O}^n$, the tensor product is easy to describe.

Theorem 4. *Let $\mathcal{K}$ be a convex cone in $\mathbb{R}^n$. Then $X \in \mathcal{K} \otimes \mathbb{R}^m_+$ if and only if every column X_i is in $\mathcal{K}$. Formally $\mathcal{K} \otimes \mathbb{R}^m_+ \cong \bigoplus_{i=1}^m \mathcal{K}$*

Proof 1.2. If $X \in \mathcal{K} \otimes \mathbb{R}^m_+$ then $X = \sum_k \lambda_k z_k y_k^T$ with $z_k \in \mathcal{K}$, $y_k \in \mathbb{R}^m_+$, and $\lambda_k \geq 0$. Therefore, every column of X is a nonnegative combination of elements of $\mathcal{K}$. Conversely if every column of X is in $\mathcal{K}$ then $X = \sum_{j=1}^m X_j e_j^T$. The second factor is clearly in $\mathbb{R}_+$. The first factor is in $\mathcal{K}$ by assumption. □

The same theorem can be shown for $\mathbb{O}^n$. The proof is almost the same.

Theorem 5. *Let $\mathcal{K}$ be a convex cone in $\mathbb{R}^n$. Then $X \in \mathcal{K} \otimes \mathbb{O}^m$ if and only if $X = 0$.*

Another tensor product of cones that can be described exactly is $\mathbb{L}^n \otimes \mathbb{L}^m$. This is derived from a non-trivial result in [50,51].

Theorem 6. *Let* $\min(n,m) \geq 3$. *Then* $X \in \mathbb{L}^m \otimes \mathbb{L}^n$ *if and only if* $\exists Y \in \mathbb{S}^{(n-1)} \otimes \mathbb{S}^{(m-1)}$ *such that*

$$X = (\mathcal{W}_n^* \otimes \mathcal{W}_m^*)(Y), \tag{1.3a}$$

$$Y \succeq 0, \tag{1.3b}$$

$$Y \in \mathbb{S}^{(n-1)} \otimes \mathbb{S}^{(m-1)}, , \tag{1.3c}$$

where

$$\mathcal{W}_r^*(Y)_i = \begin{cases} \boldsymbol{tr}(Y) & i = 0 \\ y_{00} - \sum_{i=1}^{r-2} y_{ii} & i = 1 \\ 2y_{0(i-1)} & i = 2, \dots r-1. \end{cases} \tag{1.4}$$

We note that $\mathbb{S}^{(n-1)} \otimes \mathbb{S}^{(m-1)}$ *is a* linear *subspace of* $\mathbb{S}^{(n-1)(m-1)}$ *and so* (1.3c) *can be represented as a linear equation.*

Proof 1.3. Let $\mathcal{W}_r : \mathbb{R}^r \to \mathbb{S}^{r-1}$ be given by

$$\mathcal{W}_r : \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{r-1} \end{bmatrix} \mapsto \begin{bmatrix} x_0 + x_1 & x_2 & x_3 & \dots & x_{r-1} \\ x_2 & x_0 - x_1 & 0 & \dots & 0 \\ x_3 & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ x_{r-1} & 0 & \dots & 0 & x_0 - x_1 \end{bmatrix}.$$

The adjoint of $\mathcal{W}_r$ is given in (1.4).

Now by Theorem 5.6 of [50], a matrix $L \in \mathfrak{P}(\mathbb{L}^m, \mathbb{L}^n)$ if and only if $\exists Z \in (\mathbb{S}^{n-1})^\perp \otimes (\mathbb{S}^{m-1})^\perp$ such that

$$(\mathcal{W}_n \otimes \mathcal{W}_m)(L) + Z \succeq 0.$$

Notice that $(\mathbb{S}^{r-1})^\perp$ is the set of skew-symmetric matrices, but that $Z \in (\mathbb{S}^{n-1})^\perp \otimes (\mathbb{S}^{m-1})^\perp$ is a symmetric matrix.

Appealing to the self-duality of $\mathbb{L}$, Theorem 2 states that taking the dual of this positive semidefinite description of $\mathfrak{P}(\mathbb{L}^m, \mathbb{L}^n)$ gives our result. □

1.5 Polynomials

We briefly review some basic concepts related to polynomials. A more thorough review is available in [52, Appendix A.4].

Let $x = (x_1, \dots, x_n) \in \mathbb{R}^n$ and $\alpha \in \{0, 1, 2, \dots\}^n$ be a multi-index. We write

$$x^\alpha := x_1^{\alpha_1} \cdots x_n^{\alpha_n}, \qquad |\alpha| := \sum_{i=1}^n \alpha_i.$$

We call the expression x^α the α-standard basis element for polynomials in n-variables.

A polynomial is a finite linear combination of the form

$$p(x) = \sum_{\alpha} p_\alpha x^\alpha.$$

The variables x are called *indeterminates* and the real numbers p_α are the *coefficients*.

We denote by $\mathbb{R}[x]$ the set of n-variable polynomials in x, and by $\mathbb{R}[x]_{\leq d}$ the polynomials of degree at most d. The set $\mathbb{R}[x]$ is both an infinite-dimensional vector space and a ring; it is an algebra over a the field $\mathbb{R}$.

Theorem 7 (Dimension of $\mathbb{R}[x]_{\leq d}$)**.** *Polynomials in n-variables of degree at most d form a vector space of dimension $\binom{n+d}{d}$ and therefore are isomorphic to $\mathbb{R}^{\binom{n+d}{d}}$.*

We denote the standard basis monomials of degree at most d as

$$[x]_{\leq d} \coloneqq [1, x_1, \ldots, x_n, x_1^2, x_1 x_2, \ldots, x_n^d]^T.$$

When the variable x is clear from context, we drop it, and write e.g. $p(x)$ as p. Moreover, due to the canonical isomorphism between polynomials and the vector space over its coefficients, we frequently conflate the polynomial $p(x)$ and its vector of coefficients p.

For multivariate polynomials, there are two notions of degree.

Definition 20 (Total Degree of a Polynomial)**.** *The* total degree *of a polynomial $p(x) = \sum_\alpha p_\alpha x^\alpha$ is the maximum sum of the exponents of x.*

$$\deg(p) \coloneqq \max_{\alpha} \sum_{i=1}^{n} \alpha_i.$$

We shorten the term total degree to degree.

This contrasts with what is known as the term degree.

Definition 21 (Term Degree of a Polynomial)**.** *The* term degree *of a polynomial $p(x) = \sum_\alpha p_\alpha x^\alpha$ is an n-tuple denoting the maximum degree of the i-th indeterminate*

$$tdeg(p) \coloneqq \left(\max_{\alpha} \alpha_1, \ldots, \max_{\alpha} \alpha_n \right).$$

Remark 2. *Notice that if $tdeg(p)_i = d_i$, then p is a degree d_i-polynomial in indeterminate x_i.*

It is frequently easier to work with polynomials whose nonzero terms all have the same total degree. These polynomials are called homogeneous polynomials or forms.

Definition 22 (Homogeneous Polynomial)**.** *A polynomial $p(x) = \sum_\alpha p_\alpha x^\alpha$ is called* homogeneous *or a* form *of degree d if every nonzero term has total degree d. Equivalently,*

$$p(x) = \sum_{|\alpha|=d} p_\alpha x^\alpha.$$

We denote the vector space of homogeneous polynomials of degree d by $\mathbb{R}[x]_d$. The homogeneous monomial basis of degree d is denoted:

$$[x]_d \coloneqq \begin{bmatrix} x_1^d & x_1^{d-1}x_2 & \dots & x_n^d \end{bmatrix}.$$

Notice that

$$\mathbb{R}[x]_{\leq d} = \bigoplus_{k=0}^{d} \mathbb{R}[x]_k. \tag{1.5}$$

Given a polynomial $p(x)$, it is always possible to homogenize it by increasing the number of variables by one

Definition 23 (Homogenization of a Polynomial)**.** *Let $p \in \mathbb{R}[x]$ have degree d. The* homogenization *of p is the homogeneous polynomial*

$$\tilde{p}(t, x) = t^d p(x/t),$$

which is a polynomial of degree d in $n+1$ variables.

1.5.1 Polynomials as Maps on Symmetric Tensors

A familiar type of polynomial is a homogeneous quadratic polynomial. These can be defined as $p(x) = x^T P x$. Using the vectorization identity in (1.1) we can write $p(x) = \mathbf{vec}(P)^T(x \otimes x)$. Since $x \otimes x$ is symmetric, we can simplify this to $p(x) = \hat{p}^T x^{\otimes 2}$ by simply matching the terms which are the same.

Example 1. *We consider the quadratic form:*

$$\begin{aligned} p(x) &= \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 2 & 3/2 \\ 3/2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\ &= \begin{bmatrix} 2 & 3/2 & 3/2 & 1 \end{bmatrix} \begin{bmatrix} x_1^2 & x_1x_2 & x_1x_2 & x_2^2 \end{bmatrix}^T \\ &= \begin{bmatrix} 2 & 3 & 1 \end{bmatrix} \begin{bmatrix} x_1^2 & x_1x_2 & x_2^2 \end{bmatrix}^T. \end{aligned}$$

This process can be generalized to any homogeneous polynomial. In short $[x]_d \cong x^{\otimes d}$ and therefore any homogeneous polynomial $p(x)$ can be written as:

$$p(x) = p^T[x]_d = \langle P, x^{\otimes d} \rangle, \tag{1.6}$$

where $P \in \mathbf{Sym}^d(\mathbb{R}^n)$. Formally:

$$\mathbb{R}[x]_d \cong \mathbf{Sym}^d(\mathbb{R}^n).$$

For polynomials of degree at most d, we can use (1.5) to write the isomorphism

$$\mathbb{R}[x]_{\leq d} \cong \bigoplus_{k=0}^{d} \mathbf{Sym}^k(\mathbb{R}^n). \tag{1.7}$$

The conversion between a polynomial and its symmetric tensor is purely mechanical and an explicit construction can be found in e.g. [53, §3.1].

1.5.2 Linear Subsets of $\mathbb{R}[x]$

Next, we move on to some canonical subsets of the vector space $\mathbb{R}[x]$. Polynomial ideals are the canonical affine sets in the vector space $\mathbb{R}[x]$

Definition 24 (Polynomial Ideal)**.** *Let $\{h_1, \ldots, h_M\}$ be a set of real multivariate polynomials. The (real)* polynomial ideal generated by h

$$\langle h_1, \ldots, h_M \rangle \coloneqq \left\{ \sum_{i=1}^{M} \mu_i h_i \mid \mu_i \in \mathbb{R}[x], \right\},$$

is the set of all polynomial linear combinations of $h_1, \ldots, h_M$. The elements $h_1, \ldots, h_M$ are known as the generators *of the ideal.*

This is a linear subset of $\mathbb{R}[x]$ as

$$f_1, f_2 \in \langle h_1, \ldots, h_M \rangle \implies \lambda_1 f_1 + \lambda_2 f_2 \in \langle h_1, \ldots, h_M \rangle.$$

We will also introduce the concept of ideal membership in fixed degree. This quantity will depend on the set of generators used to describe the ideal

Definition 25 (Ideal Membership in Fixed Degree)**.** *After fixing a degree $\hat{d}$, we will consider the set of polynomials $\hat{h}$ which can be written as*

$$\hat{h}(x) = \sum_{i=1}^{M} \mu_i h_i, \;\; \mu_i \in \mathbb{R}[x]_{\leq \hat{d} - \deg(h_i)}. \tag{1.8}$$

We call these the polynomials whose membership in the ideal can be proven in degree $\hat{d}$. We will denote the set of all such polynomials $\langle h_1, \ldots, h_M \rangle_{\leq \hat{d}}$ or $\mathcal{I}_{\leq \hat{d}}$.

We also define the minimum degree required to prove that $\hat{h} \in \mathcal{I}$. This quantity is denoted as

$$\deg_{\langle h_1, \ldots, h_M \rangle}(\hat{h}) \coloneqq \min \hat{d} \text{ s.t. } \hat{h} \in \langle h_1, \ldots, h_M \rangle_{\leq \hat{d}}. \tag{1.9}$$

We stress that this quantity depends explicitly on the set of generators used to describe $\mathcal{I}$, but we will nonetheless use the shorthand $\deg_{\mathcal{I}}(\hat{h})$.

1.5.3 Conic Subsets of $\mathbb{R}[x]$

A natural, proper cone in $\mathbb{R}[x]$ is the set nonnegative polynomials.

Definition 26 (Nonnegative Polynomials)**.** *A polynomial p is called* nonnegative *if $p(x) \geq 0$, $\forall x \in \mathbb{R}^n$ and* positive *if $p(x) > 0$. These sets are denoted by $\mathbb{P}[x]$, which is a cone, and $\mathbb{P}_{++}[x]$, respectively. The $[x]$ is omitted when clear from context.*

Unfortunately, checking membership in $\mathbb{P}[x]$ is NP-hard [54,55]. Another natural cone in $\mathbb{R}[x]$ is the sum-of-squares cone $\boldsymbol{\Sigma}[x]$.

Definition 27 (Sum-of-Squares Cone)**.** *A polynomial $p \in \mathbb{R}[x]$ of degree $2d$ is called a* sum-of-squares *(SOS) if it can be written as a sum*

$$p(x) = \sum_{i=1}^{k} q_i^2(x),$$

with $\deg(q_i) \leq d$. The set of all sums-of-squares polynomials is denoted $\boldsymbol{\Sigma}[x]$. The $[x]$ is omitted when clear from context.

In general, we have that $\boldsymbol{\Sigma}[x] \subset \mathbb{P}[x]$ [56–58]. Notice that positive polynomials allow us to naturally define *polynomial conic combination*. We will see later why this is an attractive subset to work with computationally.

As we will later rely on the equivalence between homogeneous polynomials and symmetric tensors, we also introduce the nonnegative tensors and the tensors of squares.

Definition 28 (Nonnegative Tensor)**.** *A symmetric tensor $P \in \boldsymbol{Sym}^{2d}(\mathbb{R}^n)$ is* nonnegative *if*

$$\langle P, x^{\otimes 2d} \rangle \geq 0, \tag{1.10}$$

for all $x \in \mathbb{R}^n$. The tensor is called positive if the inequality is strict for $x \neq 0$.

The set of nonnegative symmetric tensors is denoted

$$\boldsymbol{Sym}_{+}^{2d}(\mathbb{R}^n) = \left\{ P \in \boldsymbol{Sym}^{2d}(\mathbb{R}^n) \mid \langle P, x^{\otimes 2d} \rangle \geq 0, \ \forall x \right\}. \tag{1.11}$$

Due to the isomorphism between tensors and polynomials, checking if a tensor is nonnegative is NP-hard [59, Theorem 11.2]. Therefore, we will find it convenient to define a tensor of squares.

Definition 29 (Tensor of Squares)**.** *A symmetric tensor $P \in \boldsymbol{Sym}^{2d}(\mathbb{R}^n)$ is a tensor of squares if*

$$P = Q \otimes_{sym} Q,$$

where $Q \in \boldsymbol{Sym}^{d}(\mathbb{R}^n)$. We denote the set of tensors of squares as

$$\boldsymbol{Sym}_{2}^{2d}(\mathbb{R}^n) \coloneqq \left\{ P \in \boldsymbol{Sym}^{2d}(\mathbb{R}^n) \mid P = Q \otimes_{sym} Q, \ Q \in \boldsymbol{Sym}^{d}(\mathbb{R}^n) \right\}.$$

Tensors of squares are exactly the polynomials which are pure squares.

Both positive polynomials and sum-of-squares polynomials give us analogous ways to define conic combinations and conical subsets of $\mathbb{R}[x]$. In what follows, let $\mathcal{S} \coloneqq \{g_1, \ldots, g_N\}$ be a set of polynomials

Definition 30 (Nonnegativity Cone)**.** *The* nonnegativity cone *generated by $\mathcal{S}$ is all nonnegative polynomial combinations of the elements of $\mathcal{S}$*

$$\mathbb{P}[\mathcal{S}] \coloneqq \{ s \in \mathbb{R}[x] \mid s = p_0 + \sum_{i=1}^{N} p_i g_i, \ p_i \in \mathbb{P}[x] \}.$$

Since computation with $\mathbb{P}$ will be difficult, it will be convenient to define an analogous object with SOS polynomials instead.

Definition 31 (Quadratic Module [52, Definition 3.119])**.** *The* quadratic module *generated by the set of polynomials $\mathcal{S}$ are the SOS-combinations of the g_i,*

$$qmodule[\mathcal{S}] \coloneqq \left\{ s \in \mathbb{R}[x] \mid s = \sigma_0 + \sum_{i=1}^{N} \sigma_i g_i,\ \sigma_i \in \mathbf{\Sigma} \right\}.$$

We denote by $qmodule_{\leq d}[g_1, \ldots, g_N]$ the elements of $qmodule[g_1, \ldots, g_N]$ with total degree at most d. While d is typically chosen larger than the largest degree of g_i, it can be chosen smaller, in which case any g_i with larger degree is dropped from $qmodule_{\leq d}$.

We use the notation $qmodule_d[g_1, \ldots, g_N]$ to denote the degree d homogeneous part of the quadratic module.

Another set we will encounter is the preorder of the set of polynomials $\mathcal{S}$.

Definition 32 (Preorder [52, Definition 3.120])**.** *The* preorder *generated by the set of polynomials $\mathcal{S}$ are the SOS-combinations of all 2^N subsets of products of the generators g_i,*

$$\begin{aligned} preorder[\mathcal{S}] &\coloneqq \left\{ s \in \mathbb{R}[x] \middle| s = \sum_{\mathcal{J} \subseteq [N]} \left(\sigma_{\mathcal{J}} \prod_{i \in \mathcal{J}} g_i \right),\ \sigma_{\mathcal{J}} \in \mathbf{\Sigma}[x] \right\} \\ &= \left\{ s \in \mathbb{R}[x] \middle| s = \sigma_0(x) + \sum_i \sigma_i(x) g_i(x) + \sum_{i \neq j} \sigma_{ij}(x) g_i(x) g_j(x) + \right. \\ &\qquad \left. \sum_{i \neq j \neq k} \sigma_{ijk}(x) g_i(x) g_j(x) g_k(x) + \ldots \right\}. \end{aligned}$$

We denote by $preorder_{\leq d}(g_1, \ldots, g_N)$ the elements of $preorder(g_1, \ldots, g_N)$ with total degree at most d. Notice that this may exclude some of the larger products of the g_i.

All three sets are cones of $\mathbb{R}[x]$. Moreover, if $f_{\mathbb{P}} \in \mathbb{P}[\mathcal{S}]$, $f_{\text{qmodule}} \in \text{qmodule}[\mathcal{S}]$, and $f_{\text{preorder}} \in \text{preorder}[\mathcal{S}]$ then polynomials in each of these sets preserve the nonnegative part of $\mathcal{S}$.

$$x \in \{x \in \mathbb{R}^n \mid g_i(x) \geq 0\} \implies \left. \begin{matrix} f_{\mathbb{P}}(x) \\ f_{\text{preorder}}(x) \\ f_{\text{qmodule}}(x) \end{matrix} \right\} \geq 0.$$

Finally, we have the inclusion:

$$\text{qmodule}[\mathcal{S}] \subseteq \text{preorder}[\mathcal{S}].$$

1.5.4 Semialgebraic Sets

While the definitions from Sections 1.5.2 and 1.5.3 define subsets of $\mathbb{R}[x]$, in this section we consider how polynomials help us define subsets of $\mathbb{R}^n$.

Definition 33 (Real Algebraic Variety)**.** *A subset $\mathcal{S} \subseteq \mathbb{R}^n$ is called a* real algebraic variety *if it is the zero set of a collection of polynomials.*

$$\mathcal{S} = \{x \in \mathbb{R}^n \mid h_j(x) = 0, \forall j \in [N]\}.$$

If we allow inequalities, we get closed, basic semialgebraic sets.

Definition 34 (Closed, Basic Semialgebraic Set [52, Definition A.47])**.** *A subset $\mathcal{S} \subset \mathbb{R}^n$ is called a* closed basic semialgebraic set *if it is the intersection of a finite number of polynomial inequalities and equalities.*

$$\mathcal{S} = \{x \in \mathbb{R}^n \mid g_i(x) \geq 0,\ h_j(x) = 0,\ i \in [N], j \in [M]\}.$$

Finally, if we allow finite unions of these sets, we get closed semialgebraic sets.

Definition 35 (Closed Semialgebraic Set [52, Definition A.48])**.** *A finite union of closed basic semialgebraic sets is called a* closed semialgebraic set*.*

1.6 Conic Programs

The main computational tool used in this thesis is the solution of conic optimization programs. Let $\mathbb{V}$ be a vector space and $\mathbb{V}^*$ its dual space under the duality pairing $\langle \cdot, \cdot \rangle$. Conic optimization programs come in one of two standard forms

$$\begin{aligned} &\min \langle c, x \rangle \\ &\text{subject to } \mathcal{A}(x) - b \in \mathcal{K}, \end{aligned} \tag{P}$$

and

$$\begin{aligned} &\max \langle b, \lambda \rangle \\ &\text{subject to } \mathcal{A}^*(\lambda) = c \\ &\qquad \lambda \in \mathcal{K}^*, \end{aligned} \tag{D}$$

where $\mathcal{K}$ is a cone and $\mathcal{K}^*$ is its dual cone. The two are connected to each other via Lagrange duality.

Instances of (P) are usually specified by concrete vectors and matrices A, b, and c, a product cone $\mathcal{K} = \mathcal{K}_1 \times \cdots \times \mathcal{K}_n$, and a linear subspace defined by C and d

$$\begin{aligned} &\min q^T x + r \\ &\text{subject to } Ax - b \in \mathcal{K}_1 \times \cdots \times \mathcal{K}_n \\ &\qquad Cx - d = 0, \end{aligned} \tag{CSF}$$

where $\mathcal{K}_i$ are proper cones.

A common extension to (CSF) allows for specifying convex quadratic costs

$$\begin{aligned} \min \ & \frac{1}{2}x^T P x + q^T x + r \\ \text{subject to } & Ax - b \in \mathcal{K}_1 \times \cdots \times \mathcal{K}_n \\ & Cx - d = 0, \end{aligned} \tag{QCSF}$$

where $P \succeq 0$. Problems of the form (QCSF) can also be standardized to (CSF) at the cost of adding additional variables and a Lorentz cone constraint [60, §3.2.3]

1.7 Graphs

Graphs will be a ubiquitous object used to encode structure in this thesis. A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a pair of sets $\mathcal{V}$ called the vertices and a subset of ordered or unordered pairs $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. If $\mathcal{E}$ is ordered $\mathcal{G}$ is called a directed graph, otherwise it is undirected. Given an edge (u, v) of a directed graph we call u the tail and v the head as we typically draw directed edges as arrows $u \to v$.

We define notation for several canonical objects

Definition 36 (Incident Edges)**.** *Given $v \in \mathcal{V}$, the edges e* incident to *v are the pairs*

$$\mathcal{E}_v \coloneqq \{e \mid e = (u, v) \in \mathcal{E} \text{ or } e = (v, u) \in \mathcal{E}\}. \tag{1.12}$$

When $\mathcal{G}$ is directed, we define the natural inward incident and outward incident edges:

$$\mathcal{E}_v^{in} \coloneqq \{e \mid e = (u, v) \in \mathcal{E}\}, \tag{1.13}$$
$$\mathcal{E}_v^{out} \coloneqq \{e \mid e = (v, u) \in \mathcal{E}\}. \tag{1.14}$$

A particularly common graph that appears in this thesis are cliques.

Definition 37 (Clique)**.** *A subset $\mathcal{C} \subseteq \mathcal{V}$ of an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a clique if for every pair of vertices $u, v \in \mathcal{C}$ then $(u, v) \in \mathcal{E}$.*

1.8 Summary of Notation

Table 1.1 collects the notation introduced in this chapter, grouped by topic.

Table 1.1: Summary of Notation

Notation	Definition
Number Systems	
$\mathbb{N} \coloneqq \{0, 1, 2, \dots\}$	The natural numbers.
$\mathbb{Z} \coloneqq \{\dots, -2, -1, 0, 1, 2, \dots\}$	The integers.

Continued on next page

Table 1.1: Summary of Notation (continued)

Notation	Definition
$\mathbb{Q}$	The rational numbers.
$\mathbb{R}$	The real numbers.
$\mathbb{C}$	The complex numbers.
Indexing	
$[n]$	The ordered index set $\{1, 2, \ldots, n\}$.
$\mathcal{J} \subseteq [n]$, $\lvert\mathcal{J}\rvert$	An ordered index set and its cardinality.
$x_{\mathcal{J}}$ or $x[\mathcal{J}]$	The subvector of x indexed by $\mathcal{J}$.
$A_{\mathcal{J},\mathcal{K}}$ or $A[\mathcal{J},\mathcal{K}]$	The submatrix of A indexed by rows $\mathcal{J}$ and columns $\mathcal{K}$.
Sets, vector spaces, and norms	
$\operatorname{int}(\mathcal{S})$, $\operatorname{cl}(\mathcal{S})$	The interior and closure of a set $\mathcal{S}$.
$\alpha\mathcal{S} + \beta\mathcal{T}$	The linear combination of sets $\mathcal{S}$ and $\mathcal{T}$.
$\mathbb{V}^*$	The dual space of a vector space $\mathbb{V}$.
$\langle \cdot, \cdot \rangle$	The duality pairing or inner product.
$\mathcal{A}^*$	The adjoint of a linear map $\mathcal{A}$.
$\mathbf{vec}(X)$, $\mathbf{mat}_{mn}(x)$	Column-major vectorization of a matrix and the inverse reshaping operation.
$\mathbb{D}^n$, $\mathbb{S}^n$	The sets of diagonal and symmetric $n \times n$ matrices.
$\lVert v \rVert$	A norm of a vector v.
$\lVert v \rVert_2$, $\lVert v \rVert_\infty$, $\lVert v \rVert_Q$	Euclidean, infinity, and Q-norms of a vector.
$\lVert V \rVert_F$, $\lVert V \rVert_2$	Frobenius and spectral norms of a matrix.
Convex sets and cones	
$\mathbb{0}^n$	The zero cone in $\mathbb{R}^n$.
$\mathbb{R}^n_+$, $\mathbb{R}^n_{++}$	The nonnegative orthant and its interior.
$\mathbb{L}^n$	The Lorentz cone.
$\mathbb{S}^n_+$, $\mathbb{S}^n_{++}$	The positive semidefinite cone and its interior.
$X \succeq 0$, $X \succ 0$	Membership in the positive semidefinite and positive definite cones.
$\mathbb{D}^n_+$, $\mathbb{D}^n_{++}$	The nonnegative diagonal matrices and their interior.
$\mathbf{aff}(\mathcal{S})$, $\mathbf{conv}(\mathcal{S})$, $\mathbf{conic}(\mathcal{S})$	The affine, convex, and conic hulls of $\mathcal{S}$.
$\mathcal{S}^\infty$	The recession cone of $\mathcal{S}$.
$\mathcal{S}^*$	The dual cone of $\mathcal{S}$.
$\sigma_{\mathcal{S}}$	The support function of $\mathcal{S}$.
Tensor products	
$\mathbb{V} \otimes \mathbb{W}$	The tensor product of vector spaces $\mathbb{V}$ and $\mathbb{W}$.
$x \otimes y$	The tensor product of vectors.
$\mathbb{V}^{\otimes k}$	The k-fold tensor product of $\mathbb{V}$ with itself.
$x^{\otimes k}$	The k-fold tensor product of x with itself.

Continued on next page

Table 1.1: Summary of Notation (continued)

Notation	Definition
$\mathbf{Sym}^k(\mathbb{V})$	The symmetric tensor product space.
$T_1 \otimes_{\mathrm{sym}} T_2$	The symmetric part of $T_1 \otimes T_2$.
$A \otimes B$	The Kronecker product of matrices A and B.
$\mathcal{K}_1 \otimes \mathcal{K}_2$	The minimal tensor product of cones.
$\mathfrak{P}(\mathcal{K}_1, \mathcal{K}_2)$	The cone of positive maps between cones.
Polynomials	
α, $\lvert\alpha\rvert$	A multi-index and its total degree.
x^α	The monomial with exponent vector α.
$\mathbb{R}[x]$, $\mathbb{R}[x]_{\leq d}$	Polynomials in x and the subspace of degree at most d.
$[x]_{\leq d}$	The standard monomial basis of degree at most d.
$\deg(p)$, $\mathrm{tdeg}(p)$	The total degree and term degree of p.
$\mathbb{R}[x]_d$, $[x]_d$	Homogeneous polynomials of degree-d and the standard homogeneous monomial basis.
$\tilde{p}(t,x) \coloneqq t^{\deg(p)} p(x/t)$	The homogenization of the polynomial p
$\langle h_1, \ldots, h_M \rangle$	The polynomial ideal generated by $h_1, \ldots, h_M$.
$\langle h_1, \ldots, h_M \rangle_{\leq d}$	The elements of the ideal whose membership is proven with degree at most d.
$\mathbb{P}[x]$, $\mathbb{P}_{++}[x]$	The cones of nonnegative and positive polynomials.
$\mathbf{Sym}_+^{2d}(\mathbb{R}^n)$	The nonnegative, degree-$2d$ symmetric tensors in n variables.
$\mathbf{\Sigma}[x]$	The cone of sums-of-squares polynomials.
$\mathbf{Sym}_2^{2d}(\mathbb{R}^n)$	The degree-$2d$ symmetric tensors in n variables which are squares.
$\mathbb{P}[\mathcal{S}]$	The nonnegativity cone generated by $\mathcal{S}$.
$\mathrm{qmodule}[\mathcal{S}]$, $\mathrm{qmodule}_{\leq d}[\mathcal{S}]$	The quadratic module generated by $\mathcal{S}$ and its degree-limited subset.
$\mathrm{preorder}[\mathcal{S}]$, $\mathrm{preorder}_{\leq d}[\mathcal{S}]$	The preorder generated by $\mathcal{S}$ and its degree-limited subset.
Graphs	
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	A graph with vertices $\mathcal{V}$ and edges $\mathcal{E}$.
$\mathcal{E}_v$, $\mathcal{E}_v^{in}$, $\mathcal{E}_v^{out}$	Incident, inward incident, and outward incident edges at v.

Chapter 2

Convex Relaxations: Sums-of-Squares and Tensor Product Relaxations

We begin this thesis by reviewing two complementary ways of designing convex relaxations for polynomial optimization problems. These two methods are related, and in some sense equivalent, through convex duality. This methodology is most recognized in the literature under the name of the Sums-of-Squares (SOS) hierarchy, the Moment hierarchy, or the SOS/Moment hierarchy. Its modern use as a computational tool was introduced in parallel by Parrilo [26,61] and Lasserre [27].

2.1 Sums-Of-Squares Programming

In this section, we begin by introducing the SOS hierarchy. Readers are referred to [52, Chapter 3] and [62, Chapter 2] for a more thorough introduction.

We begin with the following motivating problem.

Problem 2.1: Certifying Polynomial Nonnegativity

Let $x \in \mathbb{R}^n$ and let p be a polynomial of degree d. Prove:

$$p(x) \geq 0, \ \forall x \in \mathcal{S} = \{x \in \mathbb{R}^n \mid g_i(x) \geq 0, \ h_j(x) = 0, i \in [N], \ j \in [M]\}. \tag{2.1}$$

An alternative way to write (2.1) is as the implication

$$x \in \mathcal{S} \implies p(x) \geq 0. \tag{2.2}$$

This problem is known to be NP-hard [54,55] in general, even when $\mathcal{S} = \mathbb{R}^n$. However, suppose an oracle provided you the following expression for p:

$$\sigma_i(x) = \sum_{k=0}^{m_i} q_{ik}^2(x), \tag{2.3a}$$

$$p(x) = \underbrace{\sigma_0(x)}_{\geq 0 \ \forall x \in \mathbb{R}^n} + \sum_i \underbrace{\sigma_i(x)}_{\geq 0 \ \forall x \in \mathbb{R}^n} \underbrace{g_i(x)}_{\geq 0 \ \forall x \in \mathcal{S}} + \sum_j \mu_j(x) \underbrace{h_j(x)}_{=0 \ \forall x \in \mathcal{S}}. \tag{2.3b}$$

The polynomials $\sigma_i(x)$ are known as Sums-of-Squares (SOS) polynomials and are self-evidently nonnegative by virtue of being the sum of terms which are themselves nonnegative. The set of these polynomials is denoted by $\mathbf{\Sigma}[x]$. Moreover, the expression succeeds in solving Problem 2.1 as we have written $p(x)$ as the sum of terms which are all nonnegative for $x \in \mathcal{S}$.

Definition 38 (Sums-of-Squares Certificate of Nonnegativity)**.** *The set of polynomials* $\sigma_0(x) \in \mathbf{\Sigma}_{\leq 2d}$, $\sigma_i(x) \in \mathbf{\Sigma}_{\leq 2d-\deg(g_i)}$ *and* $\mu_j \in \mathbb{R}[x]_{\leq 2d-\deg(h_j)}$ *in* (2.3) *are a* degree $2d$ SOS certificate *for the nonnegativity of* $p(x)$*.*

We will frequently refer to a degree $2d$ SOS certificate as simply a certificate throughout this thesis.

Recalling the definition of the quadratic module from Definition 31 and ideal from Definition 24, we notice that (2.3) expresses p as:

$$p(x) = s(x) + f(x),\ s \in \text{qmodule}[g_1, \ldots, g_N],\ f \in \langle h_1, \ldots, h_M \rangle.$$

The key observation introduced in [26,63,64] is the following semidefinite characterization of SOS polynomials:

Theorem 8 ([52, Theorem 3.39], [64, Theorem 17.1])**.** *Let* $\sigma(x) = \sum_\alpha \sigma_\alpha x^\alpha \in \mathbb{R}[x]_{\leq 2d}$*. Then* $\sigma(x)$ *is a sum-of-squares if and only if there exists* $S \in \mathbb{S}_+^{\binom{n+d}{d}}$ *such that:*

$$\sigma_\alpha = \sum_{\beta+\gamma=\alpha} S_{\beta\gamma}, \quad S \succeq 0.$$

More generally, let $b(x)$ *be a basis of* $\mathbb{R}[x]_{\leq 2d}$ *and* $m(x)$ *be a basis of* $\mathbb{R}[x]_{\leq d}$*. Let* $\mathcal{M}$ *be the unique linear map satisfying* $\mathcal{M}(b(x)) = m(x)m(x)^T$ *and let* $\mathcal{M}^*$ *be its adjoint. Then* $\sigma(x) = \sigma^T b(x)$ *is a sum of squares if and only if:*

$$\sigma = \mathcal{M}^*(S), \quad S \succeq 0.$$

This is a system of $\binom{n+2d}{2d}$ *linear constraints and one* $\binom{n+d}{d}$ *semidefinite constraint.*

As (2.3b) is a linear equality between the coefficients of p and the coefficients in the certificates, Theorem 8 enables us to search for certificates of the type (2.3) using semidefinite programming [65].

We note that there are many excellent software packages for automatically converting SOS programs into their SDP form. These include `SumOfSquares.jl` [66,67] for `Julia`, `SumOfSquares.py` [68] for `Python`, `Drake`'s `MathematicalProgram` [69] for `C++` and `Python`, and `SOSTOOLS` [70] for `Matlab`.

2.1.1 Certificates of Nonnegativity

While certificates such as (2.3) are sufficient to accomplish our goals in Problem 2.1, other options are possible. A more general certificate could be:

$$p(x) = \sigma_0(x) + \sum_i \sigma_i(x) g_i(x) + \sum_{i \neq j} \sigma_{ij}(x) g_i(x) g_j(x) + \sum_{i \neq j \neq k} \sigma_{ijk}(x) g_i(x) g_j(x) g_k(x) + \cdots + \sum_j \mu_j(x) h_j(x) \tag{2.4}$$

$$\sigma_i, \sigma_{ij}, \sigma_{ijk}, \cdots \in \mathbf{\Sigma}[x],$$

which multiplies together all the square-free products of the generating nonnegative polynomials in $\mathcal{S}$. Recalling the definition of preorder from Definition 32, (2.4) can be written as

$$p(x) = s(x) + f(x), s \in \text{preorder}[g_1, \ldots, g_N],\ f \in \langle h_1, \ldots, h_M \rangle.$$

These types of certificates have been well studied and form a cornerstone of real algebraic geometry.

A natural question is whether the certificates of type (2.3) or (2.4) are always enough to prove that $p(x) \geq 0,\ \forall x \in \mathcal{S}$. A sequence of so-called representation theorems from real-algebraic geometry known as Positivstellensatz asserts that this is true. In the following let $\mathcal{S} = \{x \in \mathbb{R}^n \mid g_i(x) \geq 0,\ h_j(x) = 0, i \in [N], j \in [M]\}$ be a semialgebraic set and p be a polynomial.

Theorem 9 (Real Nullstellensatz [52, Corollary A.53]). *Let $N = 0$, i.e. $\mathcal{S} = \{x \in \mathbb{R}^n \mid h_j(x) = 0, j \in [M]\}$. Then $p(x) = 0,\ \forall x \in \mathcal{S}$ if and only if $\exists r \in \mathbb{N},\ r > 0,\ \sigma \in \mathbf{\Sigma}[x]$ such that:*

$$p^{2r}(x) + \sigma(x) \in \langle h_1, \ldots, h_M \rangle \tag{2.5}$$

$$\Updownarrow$$

$$\exists \mu_i \in \mathbb{R}[x],\ \sigma \in \mathbf{\Sigma}[x],\ r \in \mathbb{N}\ s.t.\ p^{2r} + \sigma = \sum_{i=1}^{M} \mu_i h_i. \tag{2.6}$$

Theorem 10 (Krivine-Stengle Positivstellensatz [71,72]). *The polynomial $p(x) \geq 0\ , \forall x \in \mathcal{S}$ if and only if $\exists r \in \mathbb{N}$ and $\exists f(x), s(x), \tau(x) \in \mathbb{R}[x]$ such that*

$$\begin{gathered} \tau(x)p(x) = p(x)^{2r} + f(x) + s(x) \\ s, \tau \in \mathit{preorder}[g_1, \ldots g_N], \quad f \in \langle h_1, \ldots, h_M \rangle \\ \Updownarrow \\ \exists \mu_i \in \mathbb{R}[x],\ \sigma_i \in \mathbf{\Sigma}[x] \quad s.t.\ \tau p = p^{2r} + \sum_{\mathcal{J} \subseteq [N]} \sigma_{\mathcal{J}} \prod_{i \in \mathcal{J}} g_i + \sum_{i=1}^{M} \mu_i h_i. \end{gathered} \tag{2.7}$$

If $p(x) > 0, \forall x \in \mathcal{S}$, then we can take $r = 0$.

Under further assumptions, we can make potentially stronger statements. In the case of requiring $p(x) > 0$ on a compact set, we have

Theorem 11 (Schmudgen's Positivstellensatz [73]). *Let $\mathcal{S} = \{x \in \mathbb{R}^n \mid g_i(x) \geq 0,\ i \in [N]\}$ be compact. If the polynomial $p(x) > 0,\ \forall x \in \mathcal{S}$, then*

$$\begin{gathered} p(x) \in \mathit{preorder}[g_1, \ldots, g_N] \\ \Downarrow \\ \exists \sigma_{\mathcal{J}} \in \mathbf{\Sigma}[x] \quad s.t.\ p = \sum_{\mathcal{J} \subseteq [N]} \sigma_{\mathcal{J}} \prod_{i \in \mathcal{J}} g_i. \end{gathered} \tag{2.8}$$

In order to state one final, stronger Positivstellensatz, we must first introduce the concept of an Archimedean description of $\mathcal{S}$. Finally, if $\mathcal{S}$ satisfies the stronger condition of being Archimedean, then we can use a stronger Positivstellensatz.

Definition 39 (Archimedean Description). *Let $\mathcal{S} = \{x \in \mathbb{R}^n \mid g_i(x) \geq 0,\ i \in [N]\}$. The description of $\mathcal{S}$ in terms of $g_1, \ldots, g_N$ is* Archimedean *if there exists $K \in \mathbb{N}$ such that*

$$K - \sum_{i=1}^{n} x_i^2 = s(x), \quad s \in \mathit{qmodule}[g_1, \ldots g_N].$$

Note that being Archimedean is *not* a property of the set, but rather a property of the *description* of the set. Sets with Archimedean descriptions are always compact; indeed the definition requires that the set fit inside the ball of size K. Roughly, a description of a set is Archimedean if its compactness can be certified using the quadratic module. A compact set $\mathcal{S} = \{x \in \mathbb{R}^n \mid g_i(x) \geq 0,\ i \in [N]\}$ can always be given an Archimedean description by writing it as

$$\mathcal{S} = \{x \in \mathbb{R}^n \mid g_i(x) \geq 0,\ i \in [N],\ K - \sum_{i=1}^{n} x_i^2 \geq 0\},$$

for a sufficiently large K.

The final Positivstellensatz we state is due to Putinar.

Theorem 12 (Putinar's Positivstellensatz [74]). *Let $\mathcal{S} = \{x \in \mathbb{R}^n \mid g_i(x) \geq 0,\ i \in [N]\}$ and suppose that the description of $\mathcal{S}$ is Archimedean. If the polynomial $p(x) > 0,\ \forall x \in \mathcal{S}$, then*

$$\begin{gathered} p(x) \in \mathit{qmodule}[g_1, \ldots g_N] \\ \Downarrow \\ \exists \sigma_i \in \mathbf{\Sigma}[x] \quad s.t.\ p = \sigma_0 + \sum_{i=1}^{N} \sigma_i g_i. \end{gathered} \tag{2.9}$$

We call certificates of the nonnegativity of p over $\mathcal{S}$ of the form (2.7) Krivine-Stengle certificates, (2.8) Schmudgen certificates, and (2.9) Putinar certificates. Notice that our example in (2.3) is a Putinar certificate. We call a certificate a degree $2d$ certificate if the

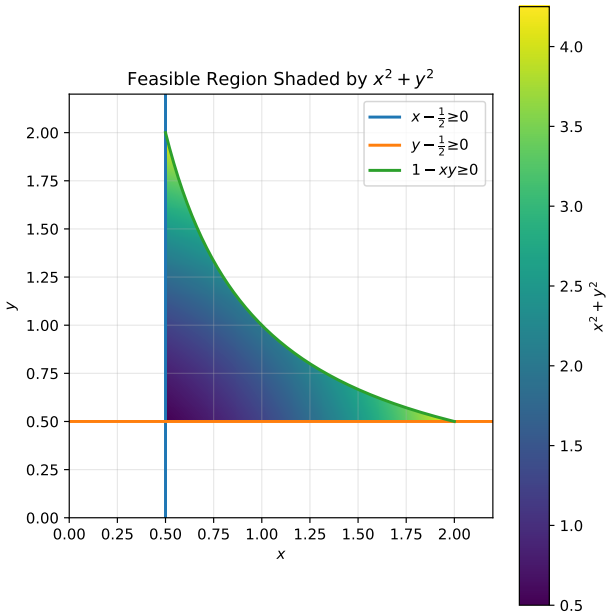


Figure 2.1: The feasible region from Example 2

terms on either side of (2.7), (2.8), and (2.9) are of degree no more than $2d$. The σ and μ terms are typically called multipliers.

We note that Theorem 8 allows us to *compute* these Positivstellensatz certificates via semidefinite programming. The condition that $f \in \langle h_1, \ldots, h_M \rangle$ can be written as a linear constraint between the coefficients of f, h's, and μ's. Membership of s in either the preorder or quadratic module of the g_i's can be written as a linear constraint between the coefficients of s, the g_i's, and the σ multipliers, while the membership of $\sigma_i \in \mathbf{\Sigma}$ can be written as a semidefinite constraint and additional linear constraints via Theorem 8.

2.1.1.1 Degree Bounded Certificates

Notice that the statements of Theorems 10, 11 and 12 make no claims on the necessary degrees of the certificates. For complexity reasons, these degrees may in fact be quite high. Moreover, while the certificates in Theorem 12 are simpler than both other theorems, the degree required to generate a Putinar certificate may be much higher than those in a Schmudgen certificate, and similarly for a Krivine-Stengle certificate [75]. Consider the following extreme example with a compact set drawn in Figure 2.1.

Example 2. *Consider the problem of testing that*

$$p(x) \coloneqq 17/4 - x^2 - y^2 \geq 0,$$
$$\forall (x, y) \in \mathcal{S} = \{(x, y) \in \mathbb{R}^2 \mid x - 1/2 \geq 0,\ y - 1/2 \geq 0,\ 1 - xy \geq 0\}.$$

We give an elementary proof in Section 2.3.1 that no certificate of the form (2.9) can exist. On the other hand, the following is an example of a certificate of the form (2.8).

$$\begin{aligned} 17/4 - x^2 - y^2 = {} & 2(y - 1/2)^2(x - 1/2) + 2(x - 1/2)^2(y - 1/2) + 5(1 - xy) \\ & + 7(x - 1/2)(y - 1/2) + 2(x - 1/2)(1 - xy) + 2(y - 1/2)(1 - xy) \end{aligned}.$$

This example is due to [76, Example 6.3.1].

Similarly, while in the infinite degree case, Theorem 9 assures us that the original generators of the ideal are sufficient, including other generators can substantially decrease the degrees required to construct our certificates.

Example 3. *Let $d \geq 2$ and let $\mathcal{S} = \left\{x, y \in \mathbb{R} \mid x^d = 0, x^2y - x = 0\right\}$. The following certificate proves that $x = 0,\ \forall x \in \mathcal{S}$:*

$$x = y^{d-1}\left(x^d\right) - \left(\sum_{i=0}^{d-2} x^i y^i\right)\left(x^2y - x\right).$$

This is a certificate of degree $2d - 1$. We defer a proof that no lower-degree certificate can exist to Section 2.3.2.

On the other hand, since $x^d = 0$, then $x^2 = 0$ and so the set $\mathcal{S}' = \{x, y \in \mathbb{R} \mid x^d = 0, x^2y - x = 0, x^2 = 0\}$ encodes the same set. Using the new description $\mathcal{S}'$ we have that

$$x = y(x^2) + 0(x^d) - 1(x^2y - x).$$

This certificate is of degree 3.

There are more vicious examples in the literature which show that checking if $p(x) = 0$ for all x satisfying some set of polynomial equalities requires multipliers whose degree is $d^{2^{\mathcal{O}(n)}}$ i.e. doubly exponential [77,78].

2.1.1.2 General Procedure

Though the representation theorems Theorems 10, 11 and 12 are necessary in infinite degree, in practice Example 2 and Example 3 show that including additional polynomials which do not change the description of the set can help substantially lower the degrees needed to produce certificates. This is a well-studied phenomenon in the polynomial proof complexity literature mostly in the context of only equality constraints [79–81], but with some study in the general basic semialgebraic case [82,83].

In general, given a set $\mathcal{S} = \{x \in \mathbb{R}^n \mid g_i(x) \geq 0,\ h_j(x) = 0,\ i \in [N],\ j \in [M]\}$ and a polynomial $p(x)$, we will search for certificates of non-negativity

$$\hat{\tau}p = \sum_{i=1}^{\hat{N}} \sigma_i \hat{g}_i + \sum_{j=1}^{\hat{M}} \mu_j \hat{h}_j$$
$$\sigma_i \in \mathbf{\Sigma},$$

where $\hat{\tau} \geq 0,\ \hat{g}_i(x) \geq 0,\ \hat{h}_j(x) = 0\ \forall x \in \mathcal{S}$. The polynomials $\hat{\tau},\ \hat{g},\ \hat{h}$ are known as *ansatz* (plural ansatze). We will always include the ansatz $1 \geq 0$. Theorem 12 takes as ansatz only the original g_i with $\hat{\tau} = 1$, while Theorem 11 takes as ansatz all subsets of products of the g_i and $\hat{\tau} \in \text{preorder}[g_1, \ldots, g_N]$. In Example 3 we start from the description $x^d = 0$ and $x^2y - x = 0$ and add as an ansatz $x^2 = 0$ to lower the degree of our certificate. Chapter 6 will be dedicated to the effective search for new equality ansatze $\hat{h}$ that will lower the degree needed to produce certificates.

2.1.2 The SOS Hierarchy For Polynomial Optimization

Equipped with the tools from Section 2.1.1, now consider the problem

$$\begin{aligned} &\inf\ p(x) \\ &\text{subject to } g_i(x) \geq 0,\ i = 1, \ldots, N\ . \\ &\qquad\quad h_j(x) = 0,\ j = 1, \ldots, M \end{aligned} \tag{2.10}$$

While this problem is difficult, a simple reformulation due to [26,63] suggests a way to prove a lower bound.

Problem 2.2: Polynomial Program Lower Bound

Let $\mathcal{S} = \{x \in \mathbb{R}^n \mid g_i(x) \geq 0,\ h_j(x) = 0, i \in [N], j \in [M]\}$. Certify the best lower bound γ for $p(x)$ for $x \in \mathcal{S}$.

$$\max\ \gamma \tag{2.11a}$$
$$\text{subject to } p(x) - \gamma \geq 0,\ \forall x \in \mathcal{S}. \tag{2.11b}$$

Now, we can replace the condition (2.11b) with one of the certificates (2.7), (2.8), or (2.9). For example, replacing (2.11b) with (2.8) yields

$$\sup\ \gamma \tag{2.12a}$$
$$\text{subject to } p(x) - \gamma = s(x) + f(x) \tag{2.12b}$$
$$s \in \text{preorder}[g_1, \ldots g_N] \tag{2.12c}$$
$$f \in \langle h_1, \ldots, h_M \rangle. \tag{2.12d}$$

Finally, Theorem 8 allows us to formulate (2.12) as a semidefinite program. Provided the antecedent of Theorem 11 hold, (2.12) is *not* a relaxation of (2.11). Even if the antecedent does not hold, any feasible solution of (2.12) is sufficient to produce a lower bound on (2.11). Additionally, in order to form and solve the semidefinite program associated to (2.12) we must pick a maximum, finite degree d for all polynomials.

$$\sup\ \gamma \tag{2.13a}$$
$$\text{subject to } p(x) - \gamma = s(x) + f(x) \tag{2.13b}$$
$$s \in \text{preorder}_{\leq d}(g_1, \ldots g_N) \tag{2.13c}$$
$$f \in \langle h_1, \ldots, h_M \rangle_{\leq d}. \tag{2.13d}$$

It is this choice of maximum degree that results in (2.13) being a *lower bound* on (2.10). Notice that for every choice of d, we get a different, potentially tighter instance of (2.13). This choice of maximum degree d leads to a hierarchy of relaxations that is the celebrated Sums-of-Squares Hierarchy.

We again emphasize that in finite degree, different choices such as which elements of the preorder versus the quadratic module and which members of the ideal are included in the program will affect the relaxation degree. Revisiting Example 2 and Example 3 makes this clear.

Example 4. *Consider the optimization problem over the set from Example 2*

$$\max\ x^2 + y^2 \tag{2.14}$$
$$\text{subject to } x - 1/2 \geq 0,\ y - 1/2 \geq 0,\ 1 - xy \geq 0. \tag{2.15}$$

The best upper bound attained by a certificate of the form Theorem 12:

$$\begin{aligned} &\inf\ \gamma \\ &\text{subject to } \gamma - x^2 - y^2 = \sigma_0 + \sigma_1(x - 1/2) + \sigma_2(y - 1/2) + \sigma_3(1 - xy) \\ &\qquad \sigma_0 \in \mathbf{\Sigma}[x, y]_{\leq 2d},\ \sigma_i \in \mathbf{\Sigma}[x, y]_{\leq 2d-2},\ i > 0 \end{aligned}$$

is ∞ for all degree d (i.e. the above program is infeasible). On the other hand, the best upper bound attained by the certificate of the form Theorem 11:

$$\begin{aligned} &\min\ \gamma \\ &\text{subject to } \gamma - x^2 - y^2 = \sigma_0 + \sigma_1(x - 1/2) + \sigma_2(y - 1/2) + \sigma_3(1 - xy) + \\ &\qquad \sigma_{12}(x - 1/2)(y - 1/2) + \sigma_{13}(x - 1/2)(1 - xy) + \sigma_{23}(y - 1/2)(1 - xy) \\ &\qquad \sigma_0 \in \mathbf{\Sigma}[x, y]_{2d},\ \sigma_i \in \mathbf{\Sigma}[x, y]_{2d-2},\ \sigma_{1i} \in \mathbf{\Sigma}[x, y]_{2d-2},\ \sigma_{23} \in \mathbf{\Sigma}[x, y]_{2d-4} \end{aligned}$$

is the globally optimal value of $17/4$ at the degree $d = 2$.

Example 5. *Consider the example over the set from Example 3.*

$$\begin{aligned} &\min\ x \\ &\text{subject to } x^2 y - x = 0,\quad x^d = 0. \end{aligned}$$

The relaxation

$$\begin{aligned} &\max\ \gamma \\ &\text{subject to } x - \gamma = \sigma_0(x, y) + \mu_0(x, y)(x^2 y - x) + \mu_1(x, y)x^d \\ &\qquad \sigma_0(x, y) \in \mathbf{\Sigma}[x, y]_{\leq 2D},\ \mu_0(x, y) \in \mathbb{R}[x, y]_{\leq 2D-3},\ \mu_1(x, y) \in \mathbb{R}[x, y]_{\leq 2D-d} \end{aligned}$$

attains value $\gamma = -\infty$ (i.e. is infeasible) if $2D < d$ and 0 at every degree $2D \geq 2d - 1$. For $d \leq 2D < 2d - 1$, then the relaxation has 0 as the supremum, but it is not attained.

The sensitivity of the tightness of SOS relaxations to their formulation will motivate the developments of Chapter 6.

2.1.3 Complexity of the SOS Hierarchy

It is worth dwelling briefly on the cost of the SOS hierarchy. State-of-the-art semidefinite programming solvers are typically interior point methods [22,84,85] which for semidefinite programs scale [43, Proposition 6.6.3] as

$$\mathcal{O}(\sqrt{l}(l^3 k + l^2 k^2 + k^3)), \tag{2.16}$$

where l is the dimension of the PSD variable and k is the number of linear constraints.

From Theorem 8, a SOS polynomial constraint in n variables of degree $2d$ requires adding $\binom{n+2d}{2d}$ linear constraints and a semidefinite constraint with $\binom{n+d}{d}$ rows.

Recall that for fixed n, $\binom{n+d}{d}$ grows as $\Theta(d^n)$. Therefore, we have that $l = \Theta(d^n)$ and $k = \Theta((2d)^n) = \Theta(d^n)$ and we get the complexity of stepping up degree in the SOS hierarchy as

$$\mathcal{O}(d^{4.5n}). \tag{2.17}$$

For fixed d, $\binom{n+d}{d}$ grows as $\Theta(n^d)$, so we get that $l = \Theta(n^d)$ and $k = \Theta(n^{2d})$. The complexity of stepping up in dimension in the SOS hierarchy is

$$\mathcal{O}(n^{6.5d}). \tag{2.18}$$

Note that we are typically in the regime where n is fixed, as this is the number of decision variables in our original problem (2.11). Therefore, the bound (2.17) is more relevant, and it says that the SOS hierarchy becomes much more expensive as the base problem becomes larger. This motivates many of the works related to stepping up in the hierarchy more efficiently.

2.2 Tensor Product Relaxations and the Moment Hierarchy

We now turn to another, systematic way of deriving a convex relaxation for an arbitrary polynomial optimization problem. We will show later that the two approaches are in fact duals of each other, but in many instances one view or the other is more natural. As with the prior section, we begin with a classic, motivating problem.

Problem 2.3: Nonconvex Quadratic Program

Let $P,\ Q_i,\ R_j \in \mathbb{S}^n$, $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{l\times n}$, and $x \in \mathbb{R}^n$. We consider the following nonconvex, quadratic program

$$\begin{aligned}
&\min\ x^T P x && \text{(2.19a)}\\
&\text{subject to } x^T Q_i x \geq 0,\ i \in [N] && \text{(2.19b)}\\
&\qquad x^T R_j x = 0,\ j \in [M] && \text{(2.19c)}\\
&\qquad Ax \geq 0 && \text{(2.19d)}\\
&\qquad Bx = 0, \quad x = \begin{bmatrix} 1 \\ y \end{bmatrix}. && \text{(2.19e)}
\end{aligned}$$

First, notice that (2.19) is already sufficient to encode mixed-integer linear programs (MILP), by setting $N = 0$, and encoding any binary constraints $x_k \in \{0, 1\}$ as $x_k(1-x_k) = 0$. MILPs are well known to contain many NP-complete problems [18], but are of very high practical interest with multiple companies such as Gurobi, Mosek, and Fico Xpress almost

entirely dedicated to building high-performance solvers for this class. Second, notice that (2.19) is a particular instance of (2.10) with only quadratic and linear polynomials, and so we could in principle derive lower bounds using the machinery of Section 2.1.

We instead consider the following well-known construction originally attributed to Naum Shor [86]. First, we recall the trace trick:

$$\mathbf{tr}(x^T P x) = \mathbf{tr}(P x x^T).$$

Therefore, (2.19) is in fact equivalent to the program

$$\begin{aligned}
\min \quad & \mathbf{tr}(PX) && \text{(2.20a)}\\
\text{subject to} \quad & \mathbf{tr}(Q_i X) \geq 0, \ i \in [N] && \text{(2.20b)}\\
& \mathbf{tr}(R_j X) = 0, \ j \in [M] && \text{(2.20c)}\\
& AXe_0 \geq 0 && \text{(2.20d)}\\
& BXe_0 = 0, \quad e_0^T X e_0 = 1 && \text{(2.20e)}\\
& X = xx^T. && \text{(2.20f)}
\end{aligned}$$

Notice that this program is *linear* in X, with the only nonconvexity now present in the rank-one constraint $X = xx^T$. The rank-one constraint is readily relaxed into the PSD constraint $X \succeq 0$. Doing so results in the following relaxation of (2.19) which is known as the Shor relaxation

$$\begin{aligned}
\min \quad & \mathbf{tr}(PX)\\
\text{subject to} \quad & \mathbf{tr}(Q_i X) \geq 0, \ i \in [N]\\
& \mathbf{tr}(R_j X) = 0, \ j \in [M]\\
& AXe_0 \geq 0\\
& BXe_0 = 0, \quad e_0^T X e_0 = 1\\
& X \succeq 0.
\end{aligned} \tag{Shor}$$

Solving this program will generate a relaxed solution X^* for which $\mathbf{tr}(PX^*)$ will lower bound (2.19). Additionally, if the resulting X is rank-one, then we can factor it as $X = vv^T$ and directly produce a solution $x = (e_0^T v)v$ to (2.19).

2.2.0.1 Multiplying Constraints

In Section 2.1, we showed that a broader class of nonnegativity certificates than (2.3) could be derived by multiplying together some of the constraints to produce certificates of the form (2.4). We can apply a similar idea here. For example,

$$\begin{aligned}
Ax \geq 0, \ e_0^T x = 1 \implies Axx^T A^T \geq 0,\\
Bx = 0 \implies Bxx^T = 0,
\end{aligned}$$

where the first inequality should be read elementwise.

This gives the following relaxation of (2.19)

$$\begin{aligned} \min \ & \mathbf{tr}(PX) \\ \text{subject to } & \mathbf{tr}(Q_i X) \geq 0, \ i \in [N] \\ & \mathbf{tr}(R_j X) = 0, \ j \in [M] \\ & AXe_0 \geq 0, \ AXA^T \geq 0 \\ & BX = 0, \quad e_0^T X e_0 = 1 \\ & X \succeq 0, \end{aligned} \tag{2.21}$$

which is stronger than (Shor) as every feasible solution of (2.21) is feasible for (Shor).

We can summarize this procedure into four steps.

1. **Lift:** $x \to xx^T =: X$.
2. **Linearize:** by rewriting all constraints on x in terms of X. For example, write $Ax \geq 0$ as $AXe_0 \geq 0$.
3. **Tighten:** by taking all the products of the constraints on x that result in constraints on X. For example, $Ax \geq 0 \implies AXA^T \geq 0$.
4. **Relax:** the rank-one constraint $X = xx^T$ to a tractable constraint such as $X \succeq 0$

2.2.1 Moment Relaxations

The procedure of the previous section can be generalized to handle the more general form (2.10). To do this, recall from Section 1.5.1 that a homogeneous degree-$2d$ polynomial $f(x)$ can be written as $f(x) = \langle F, x^{\otimes 2d} \rangle$ for some symmetric tensor F.

In the same way that (2.19) is homogenized by concentrating all the affine terms into $e_0^T x = 1$, we will assume that (2.10) has been similarly homogenized. Concretely, we will assume that every polynomial has been homogenized to degree $2d$ via the map $f(x) \mapsto t^{2d} f(x/t)$ for $2d$ sufficiently large. This, along with the conversion from homogeneous polynomials to symmetric tensors from Section 1.5.1, allows us to write (2.10) as

$$\inf \ \langle P, x^{\otimes 2d} \rangle \tag{2.22a}$$
$$\text{subject to } \langle G_i, x^{\otimes 2d} \rangle \geq 0, \ i = 1, \dots, N \tag{2.22b}$$
$$\langle H_j, x^{\otimes 2d} \rangle = 0, \ j = 1, \dots, M \tag{2.22c}$$
$$e_0^T x = 1. \tag{2.22d}$$

Next, we make the substitution $X = x^{\otimes 2d}$. We note that $e_0^T x = 1$ implies that $\langle e_0^{\otimes 2d}, x^{\otimes 2d} \rangle = 1$, allowing us to write (2.22) as

$$\inf \ \langle P, X \rangle \tag{2.23a}$$
$$\text{subject to } \langle G_i, X \rangle \geq 0, \ i = 1, \dots, N \tag{2.23b}$$
$$\langle H_j, X \rangle = 0, \ j = 1, \dots, M \tag{2.23c}$$
$$\langle e_0^{\otimes 2d}, X \rangle = 1 \tag{2.23d}$$
$$X = x^{\otimes 2d}. \tag{2.23e}$$

Notice that (2.23) is in essentially the same form as (2.20). The constraint (2.23e) is known as the Veronese variety [52, §4].

The program in (2.23) is equivalent to (2.10), but has all the non-convexity concentrated into the constraint (2.23e). Therefore, designing a convex relaxation amounts to finding a good convex relaxation of (2.23e). Ideally, we would replace (2.23e) with its convex hull, but this is known to be difficult to describe [52, §4]. In the $2d = 2$ case we replaced $X = x^{\otimes 2}$ with $X \succeq 0$. In the general $2d$ case, we can flatten $X = x^{\otimes 2d}$ into a symmetric matrix and constrain this to be PSD. Calling this linear map $\mathcal{M}$, we obtain the relaxation

$$\begin{aligned} \inf \ & \langle P, X \rangle && \text{(2.24a)} \\ \text{subject to } & \langle G_i, X \rangle \geq 0, \ i = 1, \dots, N && \text{(2.24b)} \\ & \langle H_j, X \rangle = 0, \ j = 1, \dots, M && \text{(2.24c)} \\ & \langle e_0^{\otimes 2d}, X \rangle = 1 && \text{(2.24d)} \\ & \mathcal{M}(X) \succeq 0. && \text{(2.24e)} \end{aligned}$$

The constraint (2.24e) is known as a pseudomoment constraint.

2.2.1.1 Increasing the Degree

In converting (2.22) to (2.23), we kept the tensor degree fixed at $2d$. We can construct a more general relaxation if we instead choose a tensor degree $2D \geq 2d$. Since $e_0^T x = 1$, then we have that

$$\langle P, x^{\otimes 2d} \rangle = \langle e_0^{\otimes (2D-2d)} \otimes_{\text{sym}} P, x^{\otimes 2D} \rangle.$$

When lifting $\langle G_i, x^{\otimes 2d} \rangle$ to higher degree, we note that for every degree-$(2D-2d)$ symmetric tensor $S_i \in \mathbf{Sym}_+^{2D-2d}(\mathbb{R}^n)$ then,

$$\langle S_i, x^{\otimes (2D-2d)} \rangle \langle G_i, x^{\otimes 2d} \rangle = \langle S_i \otimes_{\text{sym}} G_i, x^{\otimes 2D} \rangle \geq 0, \tag{2.25}$$

since $\langle S_i, x^{\otimes (2D-2d)} \rangle \geq 0$ for all x by definition.

Similarly, since $\langle H_j, x^{\otimes 2d} \rangle = 0$ then for every $R_j \in \mathbf{Sym}^{2D-2d}(\mathbb{R}^n)$ we have

$$\langle R_j, x^{\otimes (2D-2d)} \rangle \langle H_j, x^{\otimes 2d} \rangle = \langle R_j \otimes_{\text{sym}} H_j, x^{\otimes 2D} \rangle = 0.$$

The resulting degree-$2D$ tensor generalization of (2.23) is

$$\begin{aligned} \inf \ & \langle e_0^{\otimes (2D-2d)} \otimes_{\text{sym}} P, X \rangle && \text{(2.26a)} \\ \text{subject to } & \langle S_i \otimes_{\text{sym}} G_i, X \rangle \geq 0, \ \forall S_i \in \mathbf{Sym}_+^{2D-2d}(\mathbb{R}^n), \ i \in [N] && \text{(2.26b)} \\ & \langle R_j \otimes_{\text{sym}} H_j, X \rangle = 0, \ \forall R_j \in \mathbf{Sym}^{2D-2d}(\mathbb{R}^n), \ j \in [M], && \text{(2.26c)} \\ & \langle e_0^{\otimes 2D}, X \rangle = 1 && \text{(2.26d)} \\ & X = x^{\otimes 2D}. && \text{(2.26e)} \end{aligned}$$

Again, (2.26) is completely equivalent to (2.10). We stress that S_i and R_j are *not* decision variables. They are a family of tensors which are universally quantified. As written, (2.26) has infinitely many constraints, due to the universal quantifiers in (2.26b) and (2.26c).

To eliminate the universal quantifier in (2.26c), it suffices to enforce (2.26c) at only a set of basis elements for all homogeneous, degree-$(2D-2d)$ symmetric tensors.

Let $B_1, \ldots, B_K$ be a basis for $\mathbf{Sym}^{2D-2d}(\mathbb{R}^n)$ and let $K = \binom{n+2D-2d-1}{2D-2d}$, the dimension of $\mathbf{Sym}^{2D-2d}(\mathbb{R}^n)$. We replace (2.26c) with the finite list

$$\langle B_k \otimes_{\text{sym}} H_j, X\rangle = 0,\ \forall j \in [M],\ k \in [K].$$

Unfortunately, parametrizing the set of all nonnegative symmetric tensors is difficult [59, Theorem 11.2], since it is equivalent to parametrizing all nonnegative polynomials. To obtain a finite dimensional relaxation of (2.26b), rather than enforcing the constraint for all globally nonnegative multipliers $S_i \in \mathbf{Sym}_+^{2D-2d}(\mathbb{R}^n)$, we enforce it for all square multipliers $S_i \in \mathbf{Sym}_2^{2D-2d}(\mathbb{R}^n)$, i.e. $S_i = Q_i \otimes_{\text{sym}} Q_i$ with $Q_i \in \mathbf{Sym}^{D-d}(\mathbb{R}^n)$. Let $\mathcal{M}_{G_i}$ be the unique linear map satisfying

$$\langle Q_i \otimes_{\text{sym}} Q_i \otimes_{\text{sym}} G_i, X\rangle = \langle Q_i, \mathcal{M}_{G_i}(X) Q_i\rangle.$$

By linearity, this ensures that we certify nonnegativity for all tensors S_i which are sums-of-squares and not just pure squares. In the tensor literature $\mathcal{M}_{G_i}$ is the *partial contraction* of X against G_i [45, §14]. In the Moment/SOS literature, $\mathcal{M}_{G_i}(X)$ is known as the localizing matrix [62, §3.2.1.]. Using this parametrization, a relaxation of (2.26b) can be written as

$$\langle Q_i, \mathcal{M}_{G_i}(X) Q_i\rangle \geq 0, \forall Q_i \in \mathbf{Sym}^{D-d}(\mathbb{R}^n).$$

This universal quantifier over the pure squares is equivalent to $\mathcal{M}_{G_i}(X) \succeq 0$.

Combining this with our relaxation of the Veronese variety from Section 2.2.1 leads to the relaxation

$$\begin{aligned}
\inf\ & \langle e_0^{\otimes(2D-2d)} \otimes_{\text{sym}} P, X\rangle && \text{(2.27a)}\\
\text{subject to}\ & \mathcal{M}_{G_i}(X) \succeq 0,\ i = 1, \ldots, N && \text{(2.27b)}\\
& \langle B_k \otimes_{\text{sym}} H_j, X\rangle = 0,\ \forall j \in [M],\ k \in [K] && \text{(2.27c)}\\
& \langle e_0^{\otimes 2D}, X\rangle = 1 && \text{(2.27d)}\\
& \mathcal{M}(X) \succeq 0. && \text{(2.27e)}
\end{aligned}$$

The constraint (2.27b) relaxes (2.26b), (2.27c) exactly represents (2.26c), and (2.27e) relaxes (2.26e).

The relaxation (2.27) is known as the moment relaxation [62].

2.2.1.2 Multiplying Constraints

In reformulating (2.22) to (2.26), we replaced (2.23b) with (2.26b) which requires $S_i \in \mathbf{Sym}_+^{2D-2d}(\mathbb{R}^n)$. This, in general, is a more stringent requirement than is necessary. In general, we only need S_i to be nonnegative on the feasible set of (2.26) for (2.26b) to be valid.

Describing this set is in general more difficult than describing the set of all nonnegative tensors. However, a reasonable family of additional constraints one may consider adding are of the form:

$$\langle S_{ij} \otimes_{\text{sym}} G_i \otimes_{\text{sym}} G_j, X\rangle \geq 0,\ \forall S_{ij} \in \mathbf{Sym}_+^{2D-4d}(\mathbb{R}^n).$$

This constraint is nonnegative on the set since S_{ij} is a nonnegative tensor and the evaluation of the tensors G_i and G_j on the set are nonnegative.

Even more generally, for any subset $\mathfrak{S} \in 2^{[N]}$ such that $2|\mathfrak{S}|d \leq 2D$, we can include the constraint

$$\left\langle S_{\mathfrak{S}} \otimes_{\text{sym}} \mathbf{Sym}\left(\bigotimes_{i\in\mathfrak{S}} G_i\right), X \right\rangle \geq 0, \ \forall S_{\mathfrak{S}} \in \mathbf{Sym}_+^{2D-2|\mathfrak{S}|d}(\mathbb{R}^n). \tag{2.28}$$

Again, the universal quantifier in (2.28) is difficult to represent exactly in general, so we can replace $S_{\mathfrak{S}} \in \mathbf{Sym}_2^{2D-2|\mathfrak{S}|d}(\mathbb{R}^n)$. Letting $\mathcal{M}_{G_{\mathfrak{S}}}$ be the unique linear map satisfying

$$\left\langle Q_{\mathfrak{S}} \otimes_{\text{sym}} Q_{\mathfrak{S}} \otimes_{\text{sym}} \mathbf{Sym}\left(\bigotimes_{i\in\mathfrak{S}} G_i\right), X \right\rangle = \langle Q_{\mathfrak{S}}, \mathcal{M}_{G_{\mathfrak{S}}}(X) Q_{\mathfrak{S}} \rangle, \tag{2.29}$$

then (2.28) can be relaxed to the constraint that $\mathcal{M}_{G_{\mathfrak{S}}}(X) \succeq 0$. For $\mathfrak{S} = \emptyset$, we define $\mathbf{Sym}\left(\bigotimes_{i\in\mathfrak{S}} G_i\right) = 1$.

The resulting, stronger relaxation is

$$\begin{aligned}
&\inf \ \langle e_0^{\otimes(2D-2d)} \otimes_{\text{sym}} P, X \rangle && \text{(2.30a)}\\
&\text{subject to } \mathcal{M}_{G_{\mathfrak{S}}}(X) \succeq 0, \ \mathfrak{S} \subseteq [N] \text{ s.t.} 2|\mathfrak{S}|d \leq 2D && \text{(2.30b)}\\
&\qquad \langle B_k \otimes_{\text{sym}} H_j, X \rangle = 0, \ \forall j \in [M], \ k \in [K] && \text{(2.30c)}\\
&\qquad \langle e_0^{\otimes 2D}, X \rangle = 1 && \text{(2.30d)}\\
&\qquad \mathcal{M}(X) \succeq 0. && \text{(2.30e)}
\end{aligned}$$

Including (2.28) in (2.27) is exactly analogous to the choice of using $\text{qmodule}[g_1, \ldots, g_N]$ versus $\text{preorder}[g_1, \ldots, g_N]$ in (2.13). In fact this can be formalized via Lagrange duality.

2.2.2 Duality

It can be shown that (2.21) corresponds to the Lagrange dual of the SDP underlying the SOS relaxation of (2.19). This duality generalizes, connecting the moment hierarchy of (2.27) and (2.30) to the SOS hierarchy of (2.11) when (2.11b) is replaced with either an ideal plus a quadratic module constraint or an ideal plus preorder constraint as is done in (2.12) respectively [52, §3.5].

We make this connection explicit for the pair of programs (2.27) and

$$\begin{aligned}
&\max \ \gamma && \text{(2.31a)}\\
&\text{subject to } p(x) - \gamma = s(x) + f(x) && \text{(2.31b)}\\
&\qquad s \in \text{qmodule}_{2D}(g_1, \ldots g_N) && \text{(2.31c)}\\
&\qquad f \in \langle h_1, \ldots, h_M \rangle_{2D}. && \text{(2.31d)}
\end{aligned}$$

The Lagrangian of (2.27) is given by:

$$\begin{aligned}
\mathcal{L} = \langle e_0^{\otimes(2D-2d)} \otimes_{\text{sym}} P, X \rangle - \sum_{i=1}^{N} \langle Y_i, \mathcal{M}_{G_i}(X) \rangle - \sum_{j=1}^{M} \sum_{k=1}^{K} \mu_{jk} \langle B_k \otimes_{\text{sym}} H_j, X \rangle \\
- \gamma(\langle e_0^{\otimes(2D)}, X \rangle - 1) - \langle Y_0, \mathcal{M}(X) \rangle. \qquad \text{(2.32)}
\end{aligned}$$

Stationarity of the Lagrangian requires that

$$\nabla_X \mathcal{L} = 0$$
$$\implies e_0^{\otimes(2D-2d)} \otimes_{\text{sym}} P - \gamma e_0^{\otimes(2D)} = \mathcal{M}^*(Y_0) + \sum_{i=1}^{N} \mathcal{M}_{G_i}^*(Y_i) + \sum_{j=1}^{M}\sum_{k=1}^{K} \mu_{jk}(B_k \otimes_{\text{sym}} H_j).$$

Therefore, the dual program of (2.27) is

$$\max \gamma \tag{2.33a}$$
$$\text{subject to } e_0^{\otimes(2D-2d)} \otimes_{\text{sym}} P - \gamma e_0^{\otimes(2D)} = \tag{2.33b}$$
$$\mathcal{M}^*(Y_0) + \sum_{i=1}^{N} \mathcal{M}_{G_i}^*(Y_i) + \sum_{j=1}^{M}\sum_{k=1}^{K} \mu_{jk}(B_k \otimes_{\text{sym}} H_j)$$
$$Y_i \succeq 0, \ i = 0, 1, \ldots, N. \tag{2.33c}$$

The linear expression on the left-hand side of (2.33b) is exactly the coefficient of $p(x) - \gamma$ on the left-hand side of (2.31b) after homogenizing all polynomials to degree $2D$.

Letting $S = \mathcal{M}^*(Y_0) + \sum_{i=1}^{N} \mathcal{M}_{G_i}^*(Y_i)$ with $Y_i \succeq 0$ is exactly the explicit description that the polynomial $s(x)$ corresponding to the symmetric tensor S lies in qmodule$(g_1, \ldots g_N)$ (where we assume g_i has been properly homogenized to degree $2d$). The fact that $S \in \mathbf{Sym}^{2D}(\mathbb{R}^n)$ ensures that $s(x) \in \text{qmodule}_{2D}(g_1, \ldots g_N)$.

Finally, the polynomial $f(x)$ corresponding to the symmetric tensor $F = \sum_{j=1}^{M}\sum_{k=1}^{K} \mu_{jk}(B_k \otimes_{\text{sym}} H_j)$ is an explicit way of encoding that $f(x) = \sum_{j=1}^{M} \mu_j(x) h_j(x)$ where $\mu_j(x) \in \mathbb{R}[x]_{2D-2d}$ (again we assume h_j has been properly homogenized to degree $2d$).

2.2.3 General Tensor Product Relaxations

The program (2.10) can be generalized into the form

$$\min p(x) \tag{2.34a}$$
$$\text{subject to } g_i(x) \in \mathcal{K}_i, \ i \in [N], \tag{2.34b}$$
$$e_0^T x = 1, \tag{2.34c}$$

where we allow $g_i(x)$ to output more general vectors and $\mathcal{K}_i$ to be any convex cone including $\mathbb{0}$. Again, we assume that all polynomials in the program have been homogenized to degree $2d$. We introduce this more general form of moment/SOS relaxations as it will be useful for explaining the Graph of Convex Sets relaxation in Chapter 3. However, the developments in this chapter are likely of independent interest.

For example $g_i(x)$ could encode that the vector x maps to a matrix that must lie in the PSD cone, or that a certain vector of polynomial expressions lie in the Lorentz cone. We stress that (2.34) generalizes (2.10) and therefore is also non-convex.

Using the exact same machinery as in Section 2.2.1.1, we can express the polynomial expressions using degree-$2d$ symmetric tensors to obtain the reformulation

$$\begin{aligned} &\inf \ \langle P, X\rangle && \text{(2.35a)}\\ &\text{subject to } \langle G_i, X\rangle \in \mathcal{K}_i,\ i \in [N] && \text{(2.35b)}\\ &\langle e_0^{\otimes 2d}, X\rangle = 1 && \text{(2.35c)}\\ &X = x^{\otimes 2d}, && \text{(2.35d)} \end{aligned}$$

which is entirely analogous to (2.23). The constraint (2.35d) can be relaxed in the same manner as (2.23e).

In order to generate higher-order relaxations, we are interested in generalizing the constructions in Sections 2.2.1.1 and 2.2.1.2. We will begin with the conic generalization of (2.25). Since cones are invariant under nonnegative scaling, we again have that if $S_i \in \mathbf{Sym}_+^{2D-2d}(\mathbb{R}^n)$ then

$$\langle S_i, x^{\otimes(2D-2d)}\rangle\langle G_i, x^{\otimes 2d}\rangle \in \mathcal{K}_i$$

is a valid inequality.

Tracing exactly the same steps as in Section 2.2.1.1 results in the requiring that

$$\langle Q_i, \mathcal{M}_{G_i}(X) Q_i\rangle \in \mathcal{K}_i, \forall Q_i \in \mathbf{Sym}^{D-d}(\mathbb{R}^n).$$

The following theorem connects this constraint to the positive maps defined in Section 1.4.3.

Theorem 13. *The constraint*

$$\langle Q_i, \mathcal{M}_{G_i}(X) Q_i\rangle \in cl(\mathcal{K}_i), \forall Q_i \in \boldsymbol{Sym}^{D-d}(\mathbb{R}^n) \tag{2.36}$$

is equivalent to

$$\mathcal{M}_{G_i}(X) \in \mathfrak{P}(\mathcal{K}_i^*, \mathbb{S}_+^K), \tag{2.37}$$

where $K = \dim \boldsymbol{Sym}^{D-d}(\mathbb{R}^n)$.

Proof 2.1. Equation (2.36) is equivalent to

$$\langle \lambda, \langle Q_i, \mathcal{M}_{G_i}(X) Q_i\rangle\rangle = \left\langle Q_i, \sum_j \lambda_j \mathcal{M}_{G_i,j}(X) Q_i \right\rangle \geq 0,\ \forall \lambda \in \mathcal{K}_i^*,\ \forall Q_i. \tag{2.38}$$

This is equivalent to $\sum_j \lambda_j \mathcal{M}_{G_i,j}(X) \succeq 0, \forall \lambda \in \mathcal{K}_i^*$ which is exactly the condition that $\mathcal{M}_{G_i}(X) \in \mathfrak{P}(\mathcal{K}_i^*, \mathbb{S}_+^K)$ under the natural identification. □

The inclusion of the closure in Theorem 13 is technically a weaker condition, but practically causes little difference.

Next, we generalize multiplying constraints. This can be achieved using the tensor product of cones defined in Section 1.4.3. In particular, for every constraint $g_i(x) \in \mathcal{K}_i$ and

$g_j(x) \in \mathcal{K}_j$ such that $\deg(g_i) + \deg(g_j) = 4d \leq 2D$ (where $\deg(g_i)$ is the maximum element-wise over the whole vector) we can take:

$$\langle S_{ij}, x^{\otimes(2D-4d)} \rangle \left(\langle G_i, x^{\otimes 2d} \rangle \otimes \langle G_j, x^{\otimes 2d} \rangle \right) = \langle S_{ij} \otimes_{\text{sym}} (G_i \otimes G_j), x^{\otimes 2D} \rangle \in \mathcal{K}_i \otimes \mathcal{K}_j, \quad (2.39)$$

where again $S_{ij} \in \mathbf{Sym}_+^{2D-4d}(\mathbb{R}^n)$. It is important to note that we do *not* use the symmetric tensor product between $G_i \otimes G_j$; the tensor product of cones is in general not commutative unlike the scalar case. However, we do need to take the symmetric tensor product between $S_{ij} \otimes_{\text{sym}} (G_i \otimes G_j)$ as $\langle S_{ij}, x^{\otimes(2D-4d)} \rangle$ is a scalar and therefore is commutative. We recall that $\mathcal{K}_i \otimes \mathcal{K}_j$ is defined in Definition 16 and the constraint that $v \in \mathcal{K}_i \otimes \mathcal{K}_j$ is defined in Definition 17.

Once again, retracing the steps of Section 2.2.1.2 relaxes (2.39) to

$$\langle Q_{ij}, \mathcal{M}_{G_{ij}}(X) Q_{ij} \rangle \in \mathcal{K}_i \otimes \mathcal{K}_j, \forall Q_{ij} \in \mathbf{Sym}^{D-2d}(\mathbb{R}^n). \quad (2.40)$$

Applying Theorem 13 to (2.40) leads to the two-factor generalization of (2.30b) being written as

$$\mathcal{M}_{G_{ij}}(X) \in \mathfrak{P}((\mathcal{K}_i \otimes \mathcal{K}_j)^*, \mathbb{S}_+^{K_{ij}}), \quad (2.41)$$

where again $K_{ij} = \dim \mathbf{Sym}^{D-2d}(\mathbb{R}^n)$. The many factor generalization for subsets $\mathfrak{S} \subseteq [N]$ leads to the generalization

$$\mathcal{M}_{G_{\mathfrak{S}}}(X) \in \mathfrak{P}\left(\left(\bigotimes_{i \in \mathfrak{S}} \mathcal{K}_i \right)^*, \mathbb{S}_+^{K_{\mathfrak{S}}} \right) \quad (2.42)$$

of (2.30b), with $K_{\mathfrak{S}} \coloneqq \dim \mathbf{Sym}^{D-|\mathfrak{S}|d}(\mathbb{R}^n)$.

Appealing to Remark 1 allows us to rewrite (2.42) as

$$\mathcal{M}_{G_{\mathfrak{S}}}(X) \in \left(\bigotimes_{i \in \mathfrak{S}} \mathcal{K}_i \right) \otimes_{\max} \mathbb{S}_+^{K_{\mathfrak{S}}}. \quad (2.43)$$

This leads to the conic generalization of (2.30) as

$$\inf \ \langle e_0^{\otimes(2D-2d)} \otimes_{\text{sym}} P, X \rangle \quad (2.44a)$$

$$\text{subject to } \mathcal{M}_{G_{\mathfrak{S}}}(X) \in \left(\bigotimes_{i \in \mathfrak{S}} \mathcal{K}_i \right) \otimes_{\max} \mathbb{S}_+^{K_{\mathfrak{S}}}, \ \mathfrak{S} \subseteq [N] \text{ s.t.} |\mathfrak{S}| d \leq D \quad (2.44b)$$

$$\langle e_0^{\otimes 2D}, X \rangle = 1 \quad (2.44c)$$

$$\mathcal{M}(X) \succeq 0. \quad (2.44d)$$

We call (2.44) the conic tensor product relaxation of (2.34), or just tensor product relaxation (TPR) for short.

Several remarks are in order. First, we validate that (2.44) reduces to (2.30) when (2.34b) is a polynomial program. In this case, every $\mathcal{K}_i = \mathbb{R}_+$. One can immediately verify

that $\left(\bigotimes_{i\in\mathfrak{S}} \mathbb{R}_+\right) = \mathbb{R}_+$ and that $\mathbb{R}_+ \otimes_{\max} \mathbb{S}_+^{K_\mathfrak{S}} = \mathbb{S}_+^{K_\mathfrak{S}}$. Therefore, if $\mathcal{K}_i = \mathbb{R}_+$ for all i, we have that (2.44b) reduces to (2.30b).

Next, though (2.44) is conceptually appealing as a relaxation, it is not very useful in its current form as exactly characterizing (2.44b) is difficult in general. To obtain efficient relaxations, it becomes necessary to further relax the cones in (2.44b). The following gives one such relaxation that is useful when the minimal tensor products of the cones is easy to describe.

Theorem 14. *If $\mathbb{S}_+^{K_\mathfrak{S}}$ in (2.44b) is relaxed to $(\mathbb{D}_+^{K_\mathfrak{S}})^*$ (where duality is taken in the space $\mathbb{S}^{K_\mathfrak{S}}$) then (2.44b) is relaxed to*

$$\langle e_k \otimes_{sym} e_k, \mathcal{M}_{G_\mathfrak{S}}(X)\rangle \in cl\left(\bigotimes_{i\in\mathfrak{S}} \mathcal{K}_i\right), \ k \in [K_\mathfrak{S}], \tag{2.45}$$

where e_k is the kth standard basis element of $\mathbb{R}^{K_\mathfrak{S}}$.

That is to say, every diagonal slice of $\mathcal{M}_{G_\mathfrak{S}}(X)$ must lie in the closure of the separable cone.

Proof 2.2. We have that

$$\left(\bigotimes_{i\in\mathfrak{S}} \mathcal{K}_i\right) \otimes_{\max} (\mathbb{D}_+^{K_\mathfrak{S}})^* = \left(\left(\bigotimes_{i\in\mathfrak{S}} \mathcal{K}_i\right)^* \otimes \mathbb{D}_+^{K_\mathfrak{S}}\right)^*.$$

Since $\mathbb{D}_+^{K_\mathfrak{S}} = \mathbf{conic}\{e_k \otimes_{\rm sym} e_k \mid k \in [K_\mathfrak{S}]\}$ then $Z \in \left(\bigotimes_{i\in\mathfrak{S}} \mathcal{K}_i\right) \otimes_{\max} (\mathbb{D}_+^{K_\mathfrak{S}})^*$ if and only if

$$\langle Z, u \otimes (e_k \otimes_{\rm sym} e_k)\rangle \geq 0, \ \forall\ u \in \left(\bigotimes_{i\in\mathfrak{S}} \mathcal{K}_i\right)^*, \forall k \in [K_\mathfrak{S}]. \tag{2.46}$$

Since

$$\langle Z, u \otimes (e_k \otimes_{\rm sym} e_k)\rangle = \langle\langle e_k \otimes_{\rm sym} e_k, Z\rangle, u\rangle,$$

we have that Z satisfies (2.46) if and only if

$$\langle e_k \otimes_{\rm sym} e_k, Z\rangle \in {\rm cl}\left(\bigotimes_{i\in\mathfrak{S}} \mathcal{K}_i\right), \forall k \in [K_\mathfrak{S}].$$

□

Remark 3. *Another way to derive the relaxation with (2.45) would be to consider a simpler condition in (2.40), namely only enforcing (the multifactor generalization of) (2.40) for tensors $Q_\mathfrak{S}$ of the form e_k and taking the closure.*

The description in Theorem 14 is efficient if we can represent $\bigotimes_{i\in\mathfrak{S}} \mathcal{K}_i$. Examples of when this is the case are when all but one $\mathcal{K}_i$ are $\mathbb{O}^l$ or $\mathbb{R}_+^l$ for some l. If $\mathfrak{S} = \{i, j\}$ and $\mathcal{K}_i = \mathbb{L}^{l_i}$ and $\mathcal{K}_j = \mathbb{L}^{l_j}$, then (2.45) can be represented using Theorem 6.

Conceptually, (2.44) provides a principled, flexible abstraction for designing convex relaxations. Practitioners have several levers for trading off efficiency and strength. Increasing the degree $2D$ tightens the relaxation by allowing higher order tensor powers. This is powerful, but may lead to excessively large positive semidefinite constraints. Taking tensor products of constraints similarly strengthens the relaxation, but representing these tensor powers of constraints may be difficult. However, this reduces the task of designing a good relaxation to the more abstract task of approximating the tensor product of cones well. Approximations of complicated cones by simpler cones is a well-studied topic [49,87–91] and can be leveraged for designing more tractable relaxations.

2.3 Deferred Proofs

We briefly collect some of the proofs deferred in the main body of this chapter for completeness.

2.3.1 Proof of Example 2

The feasible region of (2.14) is plotted at Figure 2.1. To prove Example 2, we first must define the leading degree homogeneous part.

Definition 40. *Given a polynomial p, its* leading degree homogeneous part *is all the monomials with the maximum degree. For example $p(x,y) = 4x^3 - 2x^2y + 5x^2 - 7y + 10$ has leading degree homogeneous part of $\bar{p}(x,y) = 4x^3 - 2x^2y$. We denote this by $\bar{p}$.*

We will rely on the following lemma.

Lemma 1. *If $p \in \mathbf{\Sigma}[x]$ then $\bar{p} \in \mathbf{\Sigma}[x]$.*

Proof 2.3. We decompose:

$$\begin{aligned} p &= \sum_i q_i^2 \\ &= \sum_i (\underbrace{\bar{q}_i}_{\deg=d} + \underbrace{r_i}_{\deg<d})^2 \\ &= \sum_i \underbrace{\bar{q}_i^2}_{\deg=2d} + \underbrace{2\bar{q}_i r_i}_{\deg<2d} + \underbrace{r_i^2}_{\deg<2d}. \end{aligned}$$

This implies that the leading degree term of p is indeed SOS. □

Proof 2.4 (Example 2). The global optimizers occur at $(x,y) = (1/2, 2)$ and $(x,y) = (2, 1/2)$ which attains the value $17/4$.

Let $g_1 = x - 1/2$, $g_2 = y - 1/2$, $g_3 = 1 - xy$. The preorder terms are:

$$\begin{aligned} g_{12} &= xy - (1/2)x - (1/2)y + 1/4, \\ g_{13} &= -x^2y + (1/2)xy + x - (1/2), \\ g_{23} &= -xy^2 + (1/2)xy + y - 1/2. \end{aligned}$$

A degree-2 Schmudgen upper bound of the global optimizer is

$$17/4 - x^2 - y^2 = 2(y-1/2)^2(x-1/2) + 2(x-1/2)^2(y-1/2) + 5(1-xy) + 7(x-1/2)(y-1/2) + 2(x-1/2)(1-xy) + 2(y-1/2)(1-xy)$$

.

To show that the Putinar Certificate is vacuous, we show that the Quadratic Module is not Archimedean. This example and a proof that it is not Archimedean is given in [76, Section 6.3]. An elementary proof follows. Suppose that:

$$\gamma - x^2 - y^2 = \sigma_0 + \sigma_1(x-1/2) + \sigma_2(y-1/2) + \sigma_3(1-xy).$$

Let D be the maximum degree of the right-hand side before cancellation. If D is odd, then it must come from the second and third terms, and the leading part is given by:

$$x\bar{\sigma}_1(x,y) + y\bar{\sigma}_2(x,y) = 0.$$

On the region $x > 0$ and $y > 0$, then every factor and term in the above is nonnegative. Therefore, it must be the case that $\bar{\sigma}_1(x,y) = 0$ and $\bar{\sigma}_2(x,y) = 0$ for $x > 0$ and $y > 0$. Since the polynomials vanish on an open set, then $\bar{\sigma}_1$ and $\bar{\sigma}_2$ are identically zero and so the leading degree is not D.

Therefore, D is even. If $D > 2$ the leading term is given by $\bar{\sigma}_0(x,y) - xy\bar{\sigma}_3(x,y)$. Similarly, on the region $x < 0$ and $y > 0$, we have that every term and factor of the leading term is nonnegative, resulting in $\bar{\sigma}_0(x,y) = 0$ and $\bar{\sigma}_3(x,y) = 0$, a contradiction, and therefore the leading degree cannot be $D > 2$. If $D = 2$, then $\deg(\sigma_i) = 0$ for every $i \neq 0$. The leading part of each side is therefore:

$$\begin{aligned} -x^2 - y^2 &= \sigma_0(x,y) - \sigma_3 xy \\ \implies \sigma_0(x,y) &= \begin{bmatrix} x \\ y \end{bmatrix}^T \begin{bmatrix} -1 & \sigma_3/2 \\ \sigma_3/2 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}. \end{aligned}$$

Since the first principal minor of the right-hand side is not positive semidefinite, then σ_0 is not SOS, D cannot be 2, and so we conclude that the SOS relaxation is infeasible. □

2.3.2 Proof of Degree Bound in Example 3

Proof 2.5. To prove that $x \in \mathcal{I}$ cannot be proven using multipliers of degree less than $d-1$ and the generators $\{x^d, x^2y - x\}$, take $(x,y) = (t, 1/t)$. Then a certificate must show that:

$$\begin{aligned} t &= \mu_1(t,1/t)t^d + \mu_2(t,1/t)\left(\frac{t^2}{t} - t\right) \\ &= \mu_1(t,1/t)t^d \\ &= \sum_{ij} c_{ij} t^i (1/t)^j t^d \\ &= \sum_{ij} c_{ij} t^{i-j+d}. \end{aligned}$$

Therefore, we need at least one term on the right to have degree $i - j + d = 1 \implies j = i + d - 1 \implies j \geq d - 1$. □

Chapter 3

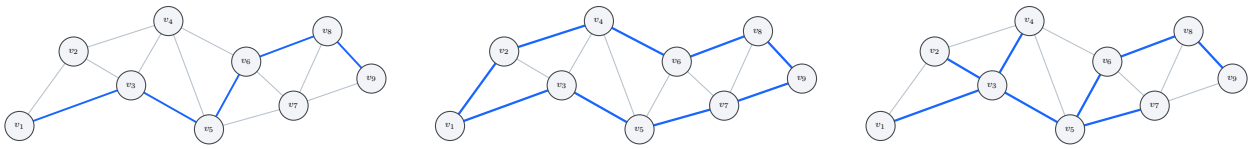
Graphs of Convex Sets

Graphs of Convex Sets (GCS) is a framework for modeling mixed discrete-continuous optimization programs introduced in [1] and subsequently expanded on in the dissertation [38] and paper [37]. The method extends classic graph problems such as shortest paths, minimum spanning trees, and traveling salesman to continuous domains by extending the discrete vertices and edges of the graph to convex optimization problems. GCS problems are mixed-integer convex programs (MICPs), but their efficient solution in practice is greatly bolstered by a relatively tight convex relaxation. As with many convex relaxations in the literature, the GCS relaxation can be viewed as a type of SOS relaxation, with pragmatic choices made to trade off between tightness and efficiency. The method has been used to great effect in a large number of robotics applications including collision-free motion planning [92–94], contact-rich manipulation [95,96], and multi-agent coordination [97].

Our interest in GCS for the purposes of this thesis is two-fold. First, construction of the actual GCS object for real robotics problems will motivate some of the developments in Chapters 5 and 6. Second, the success of the GCS relaxation in practice makes it an interesting test family for developing custom solvers for SOS-style relaxations. This will motivate Chapter 9.

Due to its pervasive use as an example in this thesis, we briefly review GCS and its convex relaxation. We refer the reader to the dissertation [38] and paper [37] for an in-depth treatment of the subject.

3.1 Graph Problems



(a) Shortest path. (b) Traveling Salesman. (c) Minimum Spanning tree.

Figure 3.1: A visualization of many classic graph problems.

To formally introduce GCS, we must first introduce the concept of a graph problem.

Given the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we consider a family of admissible subgraphs $\mathfrak{H}$. These subgraphs can be e.g. s-t paths if we are considering shortest paths from s to t, tours if we are solving the TSP, or trees if we are solving a spanning tree problem. See Figure 3.1 for examples.

We encode the selection of the subgraph as picking a binary vector $y = (y_{\mathcal{V}}, y_{\mathcal{E}}) \in \{0,1\}^{|\mathcal{V}|+|\mathcal{E}|}$. The vector $y^{\mathcal{H}}$ encodes selection of the subgraph $\mathcal{H} = (\mathcal{W}, \mathcal{F})$ if

$$y_v^{\mathcal{H}} = \begin{cases} 1 & v \in \mathcal{W} \\ 0 & v \notin \mathcal{W} \end{cases}, \qquad y_e^{\mathcal{H}} = \begin{cases} 1 & e \in \mathcal{F} \\ 0 & e \notin \mathcal{F} \end{cases}.$$

The majority of interesting families of subgraphs admit a description as a binary polytope. We assume that $\mathcal{A}_{\mathfrak{H}} : \mathbb{R}^{|\mathcal{V}|+|\mathcal{E}|} \to \mathbb{R}^{m_{\mathfrak{H},\geq}}$ and $\mathcal{C}_{\mathfrak{H}} : \mathbb{R}^{|\mathcal{V}|+|\mathcal{E}|} \to \mathbb{R}^{m_{\mathfrak{H},=}}$ are linear maps and encode the family of subgraphs as

$$\mathcal{H} \in \mathfrak{H} \iff \left\{ y \mid \mathcal{A}_{\mathfrak{H}}(y) - b_{\mathfrak{H}} \geq 0,\ \mathcal{C}_{\mathfrak{H}}(y) - d = 0,\ y \in \{0,1\}^{|\mathcal{V}|+|\mathcal{E}|} \right\}. \tag{3.1}$$

As $y^{\mathcal{H}}$ must encode a subgraph, we can assume that the constraint

$$y_v \geq y_e, \forall e \in \mathcal{E}_v, \tag{3.2}$$

is a part of the encoding $\mathcal{A}_{\mathfrak{H}}(y^{\mathcal{H}}) - b_{\mathfrak{H}} \geq 0$. The constraints (3.2) encode the fact that if e is included in the subgraph, then the vertex must also be a member of the subgraph.

Example 6 (s-t paths)**.** *A path from s to t can be encoded as the polytope*

$$y_v = \sum_{e \in \mathcal{E}_v^{in}} y_e + \delta_{sv} \tag{3.3a}$$

$$y_v = \sum_{e \in \mathcal{E}_v^{out}} y_e + \delta_{tv} \tag{3.3b}$$

$$\sum_{e \in \mathcal{E}_{\mathcal{U}}} y_e \leq |\mathcal{U}| - 1 : \forall \mathcal{U} \subseteq \mathcal{V} : |\mathcal{U}| \geq 2 \tag{3.3c}$$

$$y_v \geq y_e, \forall e \in \mathcal{E}_v \tag{3.3d}$$

$$y \in \{0,1\}^{|\mathcal{V}|+|\mathcal{E}|}. \tag{3.3e}$$

Equations (3.3a) and (3.3b) encode that at most one edge entering the vertex and one edge leaving the vertex can be active. Equation (3.3c) ensures that the final selected subgraph contains no cycles. As there are exponentially many constraints of the form (3.3c), in practice only a subset of these are included. When solving a shortest path problem with positive edge weights, the cycle elimination constraints are redundant [98].

See [37,38] for further examples, including binary polytopic encodings of the TSP and spanning tree problems.

After associating costs c_v and c_e to each vertex and edge respectively, a graph optimization problem is simply

$$\begin{aligned} \min \ & \sum_{v\in\mathcal{V}} c_v y_v + \sum_{e\in\mathcal{E}} c_e y_e && \text{(3.4a)} \\ \text{subject to } & \mathcal{A}_{\mathfrak{H}}(y) - b_{\mathfrak{H}} \geq 0 && \text{(3.4b)} \\ & \mathcal{C}_{\mathfrak{H}}(y) - d = 0 && \text{(3.4c)} \\ & y \in \{0,1\}^{|\mathcal{V}|+|\mathcal{E}|}. && \text{(3.4d)} \end{aligned}$$

3.1.1 Vertex Local Constraints

For many graph-problems, a large subset of the constraints have a vertex-local structure; they can be written in the form

$$a_v y_v + \sum_{i\in\mathcal{E}_v} a_e y_e \geq 0,$$

or similarly for equalities.

In (3.3) of Example 6, constraints (3.3a) and (3.3b) are both vertex local equalities. Additionally, (3.3c) is vertex local if $\mathcal{U} = \{u, v\}$ and u and v are adjacent in the graph.

The set of all vertex-local constraints will play an important role later. We will denote the set of all vertex local equalities and inequalities at a given vertex v as

$$\begin{aligned} \mathcal{A}^v_{\mathfrak{H}}(y_v, y_{\mathcal{E}_v}) - b^v_{\mathfrak{H}} \geq 0 \\ \mathcal{C}^v_{\mathfrak{H}}(y_v, y_{\mathcal{E}_v}) - d^v_{\mathfrak{H}} = 0. \end{aligned} \tag{3.5}$$

We emphasize that these are not new constraints. They simply collect the subset of the constraints $\mathcal{A}_{\mathfrak{H}}(y) - b \geq 0$ and $\mathcal{C}_{\mathfrak{H}}(y) - d = 0$ which can be written using only the binaries associated to a vertex and its incident edges. We denote the number of vertex-local equalities at a vertex v as $m^v_{\mathfrak{H},=}$ and the number of vertex-local inequalities as $m^v_{\mathfrak{H},\geq}$.

3.2 The GCS Extension

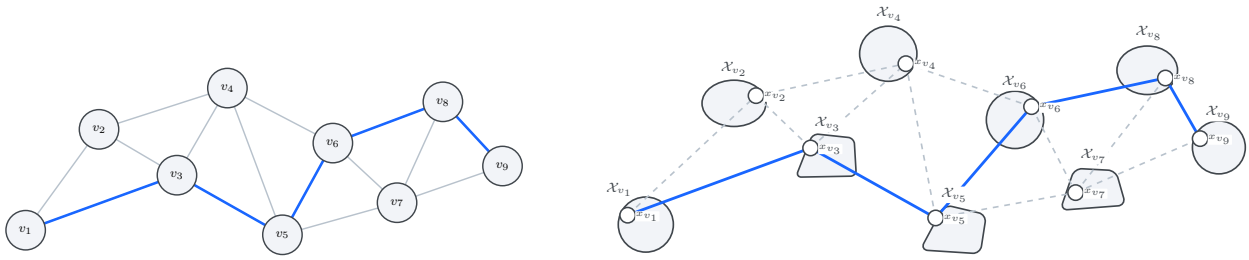
In this section we review the GCS model and develop the actual MICP that we wish to solve.

Given the graph $\mathcal{G}$, we begin by associating a conic program to each vertex

$$\begin{aligned} \min \ & \hat{f}_v^T x_v + g_v \\ \text{subject to } & \hat{\mathcal{A}}_v(x_v) - b_v \in \mathcal{K}_v, \end{aligned} \tag{3.6}$$

where $\hat{\mathcal{A}}_v : \mathbb{R}^{n_v} \to \mathbb{R}^{m_v}$ is a linear function and $\mathcal{K}_v$ is a product of proper cones $\mathcal{K}_v = \mathcal{K}_{v1} \times \cdots \times \mathcal{K}_{vn}$. In addition to proper cones, we also allow $\mathcal{K}_{vi} = \mathbb{0}$. We will find it convenient to introduce the notation vectors $f_v = \begin{bmatrix} \hat{f}_v \\ g_v \end{bmatrix}$ and linear operator $\mathcal{A}_v(x,t) = \hat{\mathcal{A}}_v(x) - b_v$, which are known as the *homogenizations* of the linear operators, and write (3.6) as

$$\begin{aligned} \min \ & f_v(x_v, 1) \\ \text{subject to } & \mathcal{A}_v(x_v, 1) \in \mathcal{K}_v. \end{aligned} \tag{3.7}$$



(a) A path chooses a sequence of edges and vertices in a graph.

(b) A GCS path chooses the sequence of edges and vertices as well as a point in each vertex.

Figure 3.2: A path in a GCS must select both the path in the graph, and a point in every convex set satisfying the edge constraints.

Similarly, we associate a conic program to every edge of $\mathcal{G}$:

$$\begin{aligned} \min \ & \begin{bmatrix} \hat{f}_{e_u}^T & \hat{f}_{e_v}^T & \hat{f}_{e_e}^T \end{bmatrix} \begin{bmatrix} x_u \\ x_v \\ x_e \end{bmatrix} + g_e \\ \text{subject to } & \hat{\mathcal{A}}_e(x_u, x_v, x_e) - b_e \in \mathcal{K}_e. \end{aligned} \tag{3.8}$$

Again, $\hat{\mathcal{A}}_e : \mathbb{R}^{n_u+n_v+n_e} \to \mathbb{R}^{m_e}$ is a linear function and $\mathcal{K}_e$ is a product of proper cones and the zero cone. We introduce the homogenizations

$$\begin{aligned} f_e(x_u, x_v, x_e, t) &= \begin{bmatrix} \hat{f}_{e_u}^T & \hat{f}_{e_v}^T & \hat{f}_{e_e}^T \end{bmatrix} \begin{bmatrix} x_u \\ x_v \\ x_e \end{bmatrix} + g_e t \\ \mathcal{A}_e(x_u, x_v, x_e, t) &= \hat{\mathcal{A}}_e(x_u, x_v, x_e) - b_e t, \end{aligned}$$

and write (3.8) as

$$\begin{aligned} \min \ & f_e(x_u, x_v, x_e, 1) \\ \text{subject to } & \mathcal{A}_e(x_u, x_v, x_e, 1) \in \mathcal{K}_e. \end{aligned} \tag{3.9}$$

Given a family of admissible subgraphs $\mathfrak{H}$, the GCS extension of the underlying graph optimization problem combines the graph constraints of (3.4) with the conic programs (3.7) and (3.9) to produce the program

$$\min \sum_{v \in \mathcal{V}} y_v f_v(x_v, 1) + \sum_{e \in \mathcal{E}} y_e f_e(x_u, x_v, x_e, 1) \tag{3.10a}$$

$$\text{subject to } y_v \mathcal{A}_v(x_v, 1) \in \mathcal{K}_v \ \forall v \in \mathcal{V} \tag{3.10b}$$

$$y_e \mathcal{A}_e(x_u, x_v, x_e, 1) \in \mathcal{K}_e \ \forall e \in \mathcal{E} \tag{3.10c}$$

$$\mathcal{A}_{\mathfrak{H}}(y) - b_{\mathfrak{H}} \geq 0, \ \mathcal{C}_{\mathfrak{H}}(y) - d_{\mathfrak{H}} = 0 \tag{3.10d}$$

$$y \in \{0, 1\}^{|\mathcal{V}|+|\mathcal{E}|}. \tag{3.10e}$$

Constraints (3.10d) and (3.10e) encode that (3.10) must select a subgraph of interest. If $y_v = 0$, then (3.10b) can be trivially satisfied by choosing any finite x_v since $\mathcal{K}_v$ is a product

of a proper cone and $\mathbb{0}$. Conversely, if $y_v = 1$, then (3.10b) encodes the constraint that x_v must be in the conic set associated to the vertex v. The interpretation of (3.10c) is similar for the edge e. Finally, the cost (3.10a) associates a penalty to choosing the points x_v (and the variables at the edges x_e) for only those vertices and edges which are included in the selected subgraph.

For a fixed set of binaries, or equivalently after selecting a subgraph, (3.10) is a convex program and selects points in the convex sets which minimize the cost of the subgraph. For a fixed set of points x_v and x_e, (3.10) recovers the original graph optimization problem (3.4) and amounts to finding the cheapest subgraph given the cost of each vertex and edge. Together, (3.10) jointly optimizes both a set of discrete choices, the subgraph, and a set of continuous choices, the points in each set.

GCS problems can be NP-Hard to solve in general, even if the original discrete graph problem is in P. This is true for the shortest path [1]. Nevertheless, the following section shows both how to transform (3.10) into a MICP and how to design an effective convex relaxation that is quite tight in practice.

3.3 The GCS Convex Relaxation

In this section, we review the mixed-integer convex formulation of (3.10) and a specific relaxation proposed by [37,38]. The proposed formulation is very closely related to the TPRs described in Section 2.2.3 We begin by noting the following quadratic equality that is implied by the subgraph constraint

$$\left.\begin{array}{l} y_v \geq y_e \\ y_v, y_e \in \{0,1\} \end{array}\right\} \implies y_v y_e = y_e. \tag{3.11}$$

Next, we introduce the following product variables

$$z_v = y_v x_v, \quad z_e = y_e x_e, \quad z_{e_v} = y_e x_v = y_e z_v$$

where the last equality follows from (3.11).

Additionally, we encode $y \in \{0,1\}^{|\mathcal{V}|+|\mathcal{E}|}$ as

$$y \in \{0,1\}^{|\mathcal{V}|+|\mathcal{E}|} \iff \begin{cases} y_v(1-y_v) = 0 & \forall v \in \mathcal{V}, \\ y_e(1-y_e) = 0 & \forall e \in \mathcal{E}. \end{cases}$$

In these variables, we can express (3.10) as

$$\min \sum_{v\in\mathcal{V}} f_v(z_v, y_v) + \sum_{e\in\mathcal{E}} f_e(z_{e_u}, z_{e_v}, z_e, y_e) \tag{3.12a}$$
$$\text{subject to } \mathcal{A}_v(z_v, y_v) \in \mathcal{K}_v \ \forall v \in \mathcal{V} \tag{3.12b}$$
$$\mathcal{A}_e(z_{e_u}, z_{e_v}, z_e, y_e) \in \mathcal{K}_e \ \forall e \in \mathcal{E} \tag{3.12c}$$
$$\mathcal{A}_{\mathfrak{H}}(y) - b_{\mathfrak{H}} \geq 0,\ \mathcal{C}_{\mathfrak{H}}(y) - d_{\mathfrak{H}} = 0 \tag{3.12d}$$
$$y_v(1 - y_v) = 0,\ \forall v \in \mathcal{V} \tag{3.12e}$$
$$y_e(1 - y_e) = 0,\ \forall e \in \mathcal{E} \tag{3.12f}$$
$$z_{e_v} = y_e z_v,\ \forall v \in \mathcal{V},\ \forall e \in \mathcal{E}_v \tag{3.12g}$$
$$z_v = y_v x_v,\ \forall v \in \mathcal{V},\ z_e = y_e x_e,\ \forall e \in \mathcal{E} \tag{3.12h}$$

Notice that now x_v and x_e enter only through (3.12h) and are not coupled to any other variables, therefore we can remove (3.12h) entirely. We emphasize that (3.12) is equivalent to (3.10). However, in this form we can immediately apply the machinery of Section 2.2 to design a convex relaxation of (3.12). We will refer to the variable y as "flows" or "binaries" interchangeably and z variables as "spatial variables".

The first level of the Moment/SOS hierarchy prescribed by Section 2.2 takes the tensor product between all variables

$$(z_{\mathcal{V}}, z_{\mathcal{E}}, y_{\mathcal{V}}, y_{\mathcal{E}}, x_v, 1) \otimes (z_{\mathcal{V}}, z_{\mathcal{E}}, y_{\mathcal{V}}, y_{\mathcal{E}}, x_v, 1) \tag{3.13}$$

where $z_{\mathcal{V}} = (z_{v1}, \ldots, z_{v|\mathcal{V}|})$ and $z_{\mathcal{E}} = (z_{e1}, \ldots, z_{e|\mathcal{E}|})$. It also suggests adding all possible tensor products between the pairs of constraints.

In practice, forming and solving this relaxation is already prohibitively large. Therefore, [37,38] suggest forming a weaker relaxation that takes a parsimonious subset of the products. In particular, the authors suggest taking only the tensor products of the constraints in a vertex-local fashion. Concretely, using the notation for vertex-local constraints from Section 3.1.1, we consider implied conic constraints:

$$\mathcal{A}_v(z_v, y_v) \otimes \begin{bmatrix}\mathcal{A}^v_{\mathfrak{H}} & -b^v_{\mathfrak{H}}\end{bmatrix} (y_v, y_{\mathcal{E}_v}, 1) \in \mathcal{K} \otimes \mathbb{R}_+^{m^v_{\mathfrak{H},\geq}} \tag{3.14a}$$
$$\Updownarrow$$
$$A_v \begin{bmatrix} y_v z_v & y_v z_{\mathcal{E}_v} & z_v \\ y_v^2 & y_v y_{\mathcal{E}_v} & y_v \end{bmatrix} \begin{bmatrix} A^{vT}_{\mathfrak{H}} \\ -b^{vT}_{\mathfrak{H}} \end{bmatrix} \in \bigoplus_{i=1}^{m^v_{\mathfrak{H},\geq}} \mathcal{K}_v \tag{3.14b}$$

where A_v is an explicit representation of $\mathcal{A}_v$ and similarly, $A^v_{\mathfrak{H}}$ is an explicit representation of $\mathcal{A}^v_{\mathfrak{H}}$. In going from (3.14a) to (3.14b), we have used the mixed product property that $Fu \otimes Gv = (F \otimes G)(u \otimes v)$, that $u \otimes v \cong uv^T$, and the result from Theorem 4 that $\mathcal{K}_v \otimes \mathbb{R}_+^{m^v_{\mathfrak{H},\geq}} \cong \bigoplus_{i=1}^{m^v_{\mathfrak{H},\geq}} \mathcal{K}_v$.

At first glance, (3.14b) introduces several new product variables. However, we can actually dramatically simplify the matrix of variables. First, we notice $y_v \in \{0, 1\} \implies y_v^2 = y_v$. Therefore,

$$\left.\begin{aligned} y_v^2 &= y_v \\ z_v &= y_v x_v \end{aligned}\right\} \implies y_v z_v = z_v.$$

Additionally, (3.11) can be used to show that

$$\left.\begin{aligned} y_v y_e &= y_e \\ z_{e_v} &= y_e z_v \end{aligned}\right\} \implies y_{\mathcal{E}_v} z_v = z_{\mathcal{E}_v}.$$

This allows us to simplify:

$$\begin{bmatrix} y_v z_v & y_v z_{\mathcal{E}_v} & z_v \\ y_v^2 & y_v y_{\mathcal{E}_v} & y_v \end{bmatrix} = \begin{bmatrix} z_v & z_{\mathcal{E}_v} & z_v \\ y_v & y_{\mathcal{E}_v} & y_v \end{bmatrix}$$

and therefore, we can write (3.14b) as

$$A_v \begin{bmatrix} z_v & z_{\mathcal{E}_v} \\ y_v & y_{\mathcal{E}_v} \end{bmatrix} (A_{\mathfrak{H}}^{vT} - b_{\mathfrak{H}}^{vT}) \in \bigoplus_{i=1}^{m^v_{\mathfrak{H},\geq}} \mathcal{K}_v \tag{3.15}$$

Formally, after reducing (3.14b) modulo the equations (3.11), (3.12g), and (3.12h), the tensor product in (3.14b) results in (3.15) and adds implied constraints without introducing additional variables.

Following a similar argument, and using Theorem 5, we obtain that

$$(z_v, y_v) \otimes \begin{bmatrix} \mathcal{C}_{\mathfrak{H}}^v & -d_{\mathfrak{H}}^v \end{bmatrix} (y_v, y_{\mathcal{E}_v}, 1) = 0 \tag{3.16a}$$

$$\Updownarrow$$

$$\begin{bmatrix} z_v & z_{\mathcal{E}_v} \\ y_v & y_{\mathcal{E}_v} \end{bmatrix} (C_{\mathfrak{H}}^{vT} - d_{\mathfrak{H}}^{vT}) = 0 \tag{3.16b}$$

Using these implied constraints, we can write (3.12) as

$$\min \sum_{v \in \mathcal{V}} f_v(z_v, y_v) + \sum_{e \in \mathcal{E}} f_e(z_{e_u}, z_{e_v}, z_e, y_e) \tag{3.17a}$$

$$\text{subject to } A_v \begin{bmatrix} z_v & z_{\mathcal{E}_v} \\ y_v & y_{\mathcal{E}_v} \end{bmatrix} (A_{\mathfrak{H}}^{vT} - b_{\mathfrak{H}}^{vT}) \in \bigoplus_{i=1}^{m^v_{\mathfrak{H},\geq}} \mathcal{K}_v \ \forall v \in \mathcal{V} \tag{3.17b}$$

$$\begin{bmatrix} z_v & z_{\mathcal{E}_v} \\ y_v & y_{\mathcal{E}_v} \end{bmatrix} (C_{\mathfrak{H}}^{vT} - d_{\mathfrak{H}}^{vT}) = 0 \ \forall v \in \mathcal{V} \tag{3.17c}$$

$$\mathcal{A}_e(z_{e_u}, z_{e_v}, z_e, y_e) \in \mathcal{K}_e \ \forall e \in \mathcal{E} \tag{3.17d}$$

$$\mathcal{A}_{\mathfrak{H}}(y) - b_{\mathfrak{H}} \geq 0, \ \mathcal{C}_{\mathfrak{H}}(y) - d_{\mathfrak{H}} = 0 \tag{3.17e}$$

$$y \in \{0,1\}^{|\mathcal{V}|+|\mathcal{E}|} \tag{3.17f}$$

$$z_{e_v} = y_e z_v, \ \forall v \in \mathcal{V}, \ \forall e \in \mathcal{E}_v \tag{3.17g}$$

In [37], it is shown that if $y_v \geq y_e$ and $y_e \geq 0$ are included in the vertex-local constraints, then (3.17g) is implied by the combination of (3.17b) and (3.17f). Therefore, we can drop (3.17g) which results in (3.17) being an MICP. The GCS convex relaxation is given by (3.17), without the constraint (3.17g) and with (3.17f) relaxed to $y \in [0,1]$.

3.4 Connection to Sums Of Squares

The relaxation in (3.17) is a TPR, but it is in principle weaker than the general first-order TPR described in Section 2.2.3. This is because rather than taking all possible tensor products of constraints, the relaxation only takes a particular subset that is efficient to include. Put another way, (3.17) is a TPR where we have further relaxed the problem by dropping those constraints which are too large to handle. In particular, (3.17) adds only linearly many constraints without introducing any additional variables when compared to the first-order relaxation which would add quadratically many more variables and constraints.

Despite this further relaxation, many works have verified that (3.17) is sufficiently tight in practice. For some problems, the relaxation is effectively exact [37,92], for others branch-and-bound on the MICP (3.17) takes dramatically fewer steps than a simpler MICP, and for others the relaxed solution is sufficiently accurate that simple non-linear rounding gives satisfactory solutions [95,99,100].

The GCS relaxation is very typical of using SOS/Tensor Product relaxations in practice. Frequently, adding all possible variables and constraints suggested by the SOS relaxation leads to convex programs that are too large to solve efficiently in practice. Practitioners therefore construct theoretically weaker relaxations by parsimoniously introducing a subset of all possible variables and constraints, trading off strength for speed of computation. Empirically, these relaxations can be sufficiently strong for a particular task while being orders of magnitude cheaper to solve. Additionally, these relaxations are typically tailored to preserve some structure in the problem.

In Chapter 9, we will further design a solver that matches the structure of both the original GCS problem and its convex relaxation. The task of developing this solver will motivate the entirety of Part III

Chapter 4

Alternating Direction Method of Multipliers

The alternating direction method of multipliers is a classic algorithm introduced by [39] for solving convex optimization problems of the form

$$\min\ f(x) + g(z) \tag{4.1}$$
$$\text{subject to } Ax - Bz = c, \tag{4.2}$$

where A and B are linear maps into a common constraint space, c is a vector in that space, and f and g are extended-real-value functions that are closed, proper, and convex. A more recent, in-depth review of the subject is available in [101]. ADMM will be the computational workhorse of the solvers we develop in Chapters 7 and 9.

ADMM operates by finding a saddle point of the augmented Lagrangian of (4.1)

$$\mathcal{L}_{\mathcal{D}}(x, z, y) = f(x) + g(z) + \langle y, Ax - Bz - c\rangle + \mathcal{D}(Ax - Bz - c), \tag{4.3}$$

where $\mathcal{D}$ is a positive definite quadratic form. We assume that $\mathcal{L}_{\mathcal{D}}$ has a saddle point.

By mapping $y \mapsto \nabla\mathcal{D}(u)$, the augmented Lagrangian can also be written in the following scaled form

$$\mathcal{L}_{\mathcal{D}}(x, z, u) = f(x) + g(z) + \mathcal{D}(Ax - Bz - c + u) - \mathcal{D}(u). \tag{4.4}$$

The steps of ADMM are slightly simpler when written in terms of (4.4). ADMM proceeds by alternately minimizing $\mathcal{L}_{\mathcal{D}}$ with respect to first x, then z, and finally performing a step of dual ascent. The base algorithm is summarized in Algorithm 1.

Remark 4. *ADMM is typically introduced with $\mathcal{D}(r) = \frac{\rho}{2}\|r\|_2^2$ for some choice of ρ. The algorithm is very sensitive to the choice of $\mathcal{D}$. Recent work has shown the advantage of allowing the algorithm to use more general quadratic forms, including cases where $\mathcal{D}$ is positive semidefinite [102,103].*

Algorithm 1: ADMM for (4.1)

Input: A positive definite quadratic form $\mathcal{D}$ and initial iterates x^0, z^0, and u^0 and a relaxation parameter $\alpha \in (0, 2)$

$k \leftarrow 0$
while *not converged* **do**
  $x^{k+1} \leftarrow \operatorname{argmin}_x \left\{ f(x) + \mathcal{D}\left(Ax - Bz^k - c + u^k\right) \right\}$
  $z^{k+1} \leftarrow \operatorname{argmin}_z \left\{ g(z) + \mathcal{D}\left(\alpha(Ax^{k+1} - c) + (1-\alpha)Bz^k - Bz + u^k\right) \right\}$
  $u^{k+1} \leftarrow u^k + \alpha(Ax^{k+1} - c) + (1-\alpha)Bz^k - Bz^{k+1}$
  $k \leftarrow k + 1$

Algorithm 1 is known to converge asymptotically [21] though in the most general setting only a sublinear ergodic rate is possible. Global linear rates exist in certain settings such as when f is strongly convex [104,105]; local linear rates exist in other settings such as conic programming [106,107].

Several stopping criteria can be used in Algorithm 1 (see [101, §3.3.1]), including tailored ones adapted to specific instantiations of (4.1) such as conic programming [23,40].

4.1 Acceleration Via Restarted Anchoring

Algorithm 2: Restarted Accelerated ADMM for (4.1)

Input: A positive definite quadratic form $\mathcal{D}$, an initial iterate $w^{0,0} = (x^{0,0}, z^{0,0}, u^{0,0})$, parameters $\alpha \in (0, 2)$, $\beta \geq 2$, and $\gamma \in (0, 2]$, and a restart trigger

Set $t \leftarrow 0$ and $\hat{w}^{0,0} \leftarrow w^{0,0}$
while *not converged* **do**
  Set $k \leftarrow 0$
  while *not converged and no restart trigger has fired* **do**
    $\bar{x}^{t,k+1} \leftarrow \operatorname{argmin}_x \left\{ f(x) + \mathcal{D}\left(Ax - Bz^{t,k} - c + u^{t,k}\right) \right\}$
    $\bar{z}^{t,k+1} \leftarrow \operatorname{argmin}_z \left\{ g(z) + \mathcal{D}\left(\alpha(A\bar{x}^{t,k+1} - c) + (1-\alpha)Bz^{t,k} - Bz + u^{t,k}\right) \right\}$
    $\bar{u}^{t,k+1} \leftarrow u^{t,k} + \alpha(A\bar{x}^{t,k+1} - c) + (1-\alpha)Bz^{t,k} - B\bar{z}^{t,k+1}$
    $\hat{w}^{t,k+1} \leftarrow (1-\gamma)w^{t,k} + \gamma(\bar{x}^{t,k+1}, \bar{z}^{t,k+1}, \bar{u}^{t,k+1})$
    $w^{t,k+1} \leftarrow w^{t,k} + \frac{\beta}{2(k+\beta)}(\hat{w}^{t,k+1} - w^{t,k}) + \frac{k}{k+\beta}\left(\hat{w}^{t,k+1} - \hat{w}^{t,k}\right)$
    $k \leftarrow k + 1$
  $w^{t+1,0} \leftarrow w^{t,k}$, $\hat{w}^{t+1,0} \leftarrow w^{t+1,0}$
  $t \leftarrow t + 1$

ADMM can be viewed as a fixed-point iteration

$$(x^{k+1}, z^{k+1}, u^{k+1}) = \mathcal{T}_\alpha(x^k, z^k, u^k), \tag{4.5}$$

where $\mathcal{T}_\alpha$ denotes one full loop of Algorithm 1. We let $w^k \coloneqq (x^k, z^k, u^k)$.

Halpern iteration was introduced in [108] for improving the worst-case convergence of nonexpansive fixed-point iterations such as ADMM. Given an initial point w^0, Halpern iteration modifies (4.5) to

$$w^{k+1} = \frac{k+1}{k+2}\mathcal{T}_\alpha(w^k) + \frac{1}{k+2}w^0. \tag{4.6}$$

Halpern iteration converges to the element of the fixed-point set closest to the anchor w^0 at an optimal rate of $\mathcal{O}(1/k)$.

The authors of [103,109] suggest the more general iteration

$$\hat{w}^{k+1} = (1-\gamma)w^k + \gamma\mathcal{T}_\alpha(w^k) \tag{4.7a}$$

$$w^{k+1} = w^k + \frac{\beta}{2(k+\beta)}(\hat{w}^{k+1} - w^k) + \frac{k}{k+\beta}\left(\hat{w}^{k+1} - \hat{w}^k\right), \tag{4.7b}$$

with $\beta \geq 2$ and $\gamma \in (0, 2]$.

Notice that the scheme in (4.7) has three hyperparameters α, β, γ which must be chosen. With the choice of $\beta = 2$, (4.7b) corresponds to Halpern iteration on the shadow sequence (4.7a). When $\beta = 2$ and $\gamma = 1$, (4.7) reduces to (4.6). When $\beta = 2$ and $\gamma = 2$, (4.7) recovers the reflected Halpern iteration proposed in [110]. The authors of [103] show that when $\beta = 2$, the sequence $\mathcal{T}_\alpha(w^k)$ from the recurrence (4.7) exhibits $O(1/k)$ convergence to a solution. When $\beta > 2$, $\mathcal{T}_\alpha(w^k)$ converges at an $o(1/k)$ rate to a solution.

In addition to the extrapolation scheme (4.7), it has been shown that periodically resetting the anchor point w^0 can improve the convergence by a constant factor both in theory [106,110] and in practice [111]. Restarts can occur at fixed intervals or be triggered adaptively by the progress of the algorithm.

Combining (4.7) with restarts results in Algorithm 2. Lines 2.5, 2.6 and 2.7 are the same as the main loop in Algorithm 1. Lines 2.8 and 2.9 correspond to the acceleration steps in (4.7) and Line 2.11 corresponds to restarting the algorithm.

Finally, as pointed out in [112, §12.3], an "accelerated" algorithm on the worst case instance does not necessarily accelerate an algorithm in practice. In particular, if Algorithm 1 is converging faster than the worst-case rate on a particular instance, then it is possible for the accelerated variant in Algorithm 2 to be slower on this instance.

Part II

Effective Sums-of-Squares Programming in Robotics

Chapter 5

Certifying Collision-Free Subsets of Configuration Space

Collision-free motion planning is the fundamental problem of moving a robot from one configuration in space to another without colliding with the environment. This task is complicated by the interplay between two spaces

1. The real three-dimensional world called the *task space*.
2. The n-dimensional mathematical space describing the configuration of the robot's joints known as the *configuration space* ($\mathcal{C}^{\text{space}}$).

$\mathcal{C}^{\text{space}}$ was introduced in the seminal work of [113] and mathematically reduces the state of the robot to a single point in the abstract mathematical space. This makes describing a robot trajectory quite easy and natural; it is simply a curve through $\mathcal{C}^{\text{space}}$. Conversely, the workspace of the robot, including the obstacles in the scene, is most naturally specified in task space. As the mapping from task-space into $\mathcal{C}^{\text{space}}$ is in general many-to-one [93,114] (this map is known as the inverse kinematics), it is frequently preferable to plan in $\mathcal{C}^{\text{space}}$. Understanding the set of configurations in $\mathcal{C}^{\text{space}}$ for which the robot is not colliding with an obstacle in task-space is therefore of paramount importance to collision-free motion planning. This space is called $\mathcal{C}^{\text{free}}$.

In this chapter, we consider the application of SOS programming to the problem of describing $\mathcal{C}^{\text{free}}$. In particular, we demonstrate how to use SOS programming to describe large, convex volumes of a bijective, rational parameterization of $\mathcal{C}^{\text{space}}$ known as the tangent configuration space[1] ($\mathcal{T}^{\text{space}}$). These volumes are easily integrated into a large variety of motion-planning problems. Additionally, we show how to apply essentially the same theory to the narrower problem of certifying that a fixed trajectory is collision-free. We see that specializing to this case provide substantial computational benefits. Finally, motivated by the ability to compute a single large volume in either $\mathcal{C}^{\text{free}}$ or $\mathcal{T}^{\text{free}}$, we turn our attention to the problem of covering $\mathcal{C}^{\text{free}}$ using a family of these volumes. In all cases, we emphasize

[1]The name tangent-configuration space is due to the use of the trigonometric tangent function used to map between $\mathcal{C}^{\text{space}}$ and $\mathcal{T}^{\text{space}}$. $\mathcal{T}^{\text{space}}$ is *not* the tangent space of the configuration space in the sense of differential geometry.

high-quality, open-source implementations of our ideas using `Drake` [69] and show how our methods scale to realistic robotics settings.

In this chapter, we begin by reviewing some related work in Section 5.1. After stating our problem formally in Section 5.2, we review some essential mathematical preliminaries to our approach in Section 5.3. We introduce our certification method in Section 5.4, its specialization to the trajectory case in Section 5.4.4, and how to use the method to grow provably collision free regions in Section 5.5. We continue with results on the certification procedure in Section 5.6 before finally turning our attention to covering collision-free space in Section 5.7.

The work in this chapter is based on the sequence of papers [115–118]. Portions of the current chapter and results are reproduced or adapted from these papers.

5.1 Related Work

We begin by reviewing related work on describing $\mathcal{C}^{\text{free}}$; broadly, there are two complementary approaches.

The first approach attempts to find an explicit description of the $\mathcal{C}^{\text{space}}$ obstacles from their task-space description mapped through the inverse kinematics (IK). We refer to this approach as the negative approach, as $\mathcal{C}^{\text{free}}$ is described as the complement of the set of $\mathcal{C}^{\text{space}}$ obstacles. In its full generality, the problem of describing $\mathcal{C}^{\text{space}}$ obstacles is intractable [119], and so limiting assumptions on the robot are often made. For example, [120] develops a method for computing $\mathcal{C}^{\text{space}}$ obstacles based on the Fast Fourier Transform under the assumption that the robot can only translate in the workspace. In [121], explicit descriptions of $\mathcal{C}^{\text{space}}$ obstacles due to the presence of point, line, and planar task-space obstacles are presented for two- and three-degree-of-freedom (DOF) robots. A thorough review of describing $\mathcal{C}^{\text{space}}$ obstacles can be found in [122, Chapter 3]. There, it is shown that if all the task-space obstacles are described as semi-algebraic sets (see Section 1.5.4) then $\mathcal{C}^{\text{space}}$ obstacles are also semi-algebraic. This is an important result from a complexity-theoretic standpoint as it shows that describing the $\mathcal{C}^{\text{space}}$ obstacles is at least decidable, though still very hard.

We refer to the second approach to describing $\mathcal{C}^{\text{free}}$ as the positive approach, since it seeks to directly describe $\mathcal{C}^{\text{free}}$ as a union of simpler sets. This description is attractive as a variety of optimization-based motion planning methods can efficiently leverage such descriptions, particularly when the simpler sets are convex [1,92,123,124].

The most common class of subsets of $\mathcal{C}^{\text{space}}$ which are studied are trajectories. Rapidly-exploring Random Trees (RRT) [125], Probabilistic Roadmaps (PRM) [126], and their variants can all be considered examples of the positive approach, describing $\mathcal{C}^{\text{free}}$ using piecewise-linear paths.

The most common approach for checking membership in $\mathcal{C}^{\text{free}}$ for a trajectory is by sampling and the problem of checking whether a trajectory is collision free is called dynamic collision checking. Algorithms for checking whether a single configuration is collision free are quite mature and can be performed in microseconds on modern hardware [127,128]. Sampling-based methods provide probabilistic guarantees that a path or region contains no collisions. Naive use of such methods is inherently optimistic, claiming safety even if collision can occur. A variety of more sophisticated strategies that are conservative in the other

direction (i.e. certifying safety at the cost of occasionally discarding some safe trajectories) fall broadly into three major families: feature tracking, swept volumes, and trajectory parameterization methods [129].

Both feature tracking and swept volume methods have effectively scaled to applications in both graphics and robotics. The most successful methods rely on precomputing a hierarchy of bounding volumes at various resolutions and certifying non-intersection at a finite number of points along the motion plan [130,131]. Interval arithmetic is used to certify the safety in between the static configurations [132]. The choice of samples is done in an adaptive manner which guarantees non-collision and some algorithms are fast enough for use as dynamic collision checkers during randomized motion planning [131,133].

These methods have two primary drawbacks. The first is that they frequently only consider piecewise-linear $\mathcal{C}^{\text{space}}$ motion plans and so cannot generalize to smoother plans. The second drawback can be their conservativeness. Typically, these methods require difficult-to-compute parameters such as an upper bound on the maximum length of a curve along the swept volume [133] or bounding the rate of change of the distance between two objects over the course of the trajectory [134]. If these bounds are too loose, collision-free motion plans in tight configuration spaces cannot be certified, which is arguably the most important regime.

On the other hand, trajectory parametrization methods can certify any collision-free motion plan to arbitrary precision. This is because these methods make relatively minimal assumptions to obtain exact guarantees, requiring only knowledge of the forward kinematics of the robot and a concrete description of the collision geometries [135,136]. Both are often necessary information for the simulation of any robotic system and so are readily available. However, most of these methods rely on exact, algebraic computations such as polynomial root finding, and so are typically too slow for practical deployment.

Methods to certify and construct full-dimensional volumes in $\mathcal{C}^{\text{free}}$ are less common, but do exist. Works such as [137–139] all seek to describe non-zero volume subsets of $\mathcal{C}^{\text{free}}$. Similar to RRTs and PRMs, these methods have the advantage of working in arbitrary configuration spaces, make no assumptions on the $\mathcal{C}^{\text{space}}$ obstacles, and proceed via sampling. Therefore, they are typically relatively simple to implement and quite fast in low dimensions. Unfortunately, these methods only provide probabilistic guarantees of non-collision and can have difficultly scaling to higher dimensions.

When the $\mathcal{C}^{\text{space}}$ obstacles are assumed to be convex, rigorous descriptions of $\mathcal{C}^{\text{free}}$ may be possible, though hardness results exist to obtaining a complete description. For example, in two and three dimensions with polyhedral $\mathcal{C}^{\text{space}}$ obstacles, it is known that finding a minimal decomposition is $\exists\mathbb{R}$-complete [140], and therefore NP-hard [141], to solve exactly and even APX-hard [142] to approximate[2]. Works such as [143] and [144] overcome these hardness results by finding decompositions that are unions of approximately convex sets.

In arbitrary dimensions and under the assumption of known, convex $\mathcal{C}^{\text{space}}$ obstacles, $\mathcal{C}^{\text{free}}$ can be decomposed into convex polyhedra by using the IRIS algorithm of [145]. As it is based on convex programming, IRIS is relatively fast, and is also able to generate rigorous certificates of non-collision. Unfortunately, it is often the case that obstacles are naturally

[2]A problem is said to be APX-hard if no polynomial time algorithm can achieve an approximation ratio of $1+\delta$ for some $\delta > 0$ unless $P = NP$.

described as convex sets in task space, which are rarely convex in $\mathcal{C}^{\text{space}}$.

Recently, generalizations of IRIS-like methods have also been introduced [146,147]. Similar to sampling-based certificates for certifying non-collision of trajectories, these methods generate probabilistically collision-free regions incredibly quickly, trading off exact rigorous correctness for speed. The methods presented in this chapter are slower, but are the first rigorous method, to our knowledge, for certifying that a full-dimensional volume of $\mathcal{C}^{\text{space}}$ is collision-free for robots with arbitrary degrees of freedom.

5.2 Problem Statement

We consider a known, task-space environment where our robot and all obstacles have been decomposed as a union of compact, convex bodies[3] for example cylinders, capsules, spheres, or vertex representation (V-rep) polytopes. Such collision geometries of our task space are readily available through standard tools such as V-HACD [149] and are often a required step for simulating any given environment.

Our robot is a mechanism composed of $N+1$ links connected via either revolute or prismatic joints [114]:

- Revolute (R): a 1-DOF joint permitting revolution about an axis of symmetry. An example is a door handle.
- Prismatic (P): a 1-DOF joint permitting translation along an axis. An example is a linear rail.

We assume that all revolute joints are constrained from undergoing complete rotations and all prismatic joints have bounded translation. Formally, if θ is the configuration-space variable associated with a revolute joint, then:

$$-\pi < \theta_l \leq \theta \leq \theta_u < \pi, \tag{5.1}$$

and if z is the configuration-space variable associated to a displacement then:

$$z_l \leq z \leq z_u. \tag{5.2}$$

where the bounds θ_l, θ_u, z_l, and z_u are fixed constants.

Our objective is to find large, convex regions of $\mathcal{T}^{\text{free}}$ regardless of the dimension of the configuration space. We will also design an algorithm for automatically placing these regions. Finally, we will be interested in the narrower problem of certifying that a trajectory through $\mathcal{T}^{\text{free}}$ is collision free. This objective is beyond the scope of current decomposition for non-convex spaces/objects such as V-HACD due to the dimensionality of the problem for interesting robots and the complexity of the non-linear kinematics.

Remark 5. *Our approach can handle a robot composed of any of the five algebraic joints: revolute, prismatic, planar, cylindrical, and spherical [114]. We restrict ourselves to R and P joints as the other joints can be seen as a composition of these two (see Appendix A.1 for details).*

[3]For technical reasons, we formally assume that the bodies are compact, convex sets expressible as basic semi-algebraic sets with an Archimedean description. See Definition 39 or [148, Chapter 7] for the definition of Archimedean description.

5.3 Mathematical Preliminaries

We introduce some necessary mathematical preliminaries specific to this section. We begin by recalling some classic theorems pertaining to the separation of convex bodies. We also review a parameterization of a robot's forward kinematics using rational functions.

5.3.1 Separating Convex Bodies

In this section, we review two dual ways to check whether two compact, convex sets $\mathcal{A}$ and $\mathcal{B}$ intersect by using convex optimization. Our certification programs in Section 5.4 relies on generalizations of the programs introduced in this section.

A well-known result from convex optimization is the Separating Hyperplane Theorem [42, Section 2.5] which states that $\mathcal{A}$ and $\mathcal{B}$ do not intersect if and only if there exists a hyperplane $\mathcal{H}(a,b) = \{x \mid a^T x + b = 0, (a,b) \neq (0,0)\}$ that strictly separates the two bodies. The hyperplane $\mathcal{H}(a,b)$ serves as a *certificate* of non-intersection. Such a hyperplane is visualized in Figure 5.1a and is described by the solution to program (5.3). Many previous works [150,151] have applied the Separating Hyperplane Theorem to find a *single* collision-free posture; in this chapter we apply the theorem to find a convex set of collision-free postures.

Conversely, if $\mathcal{A}$ and $\mathcal{B}$ do intersect, then it is possible to certify this by finding a point in $\mathcal{A} \cap \mathcal{B}$. Such a point can be found by solving the convex optimization program (5.4). A certificate of the *infeasibility* of (5.4) proves that $\mathcal{A}$ and $\mathcal{B}$ do not intersect. A certificate of infeasibility can be obtained by considering the dual of (5.4) [42, Section 5.8].

A solution to program (5.3) has the advantage of being able to quantify the magnitude of separation between the two bodies. Therefore, in Section 5.5 we will prefer to base our algorithm on a generalization of (5.3). However, we will see that certain results will be easier to show by considering the *infeasibility* of program (5.4).

We conclude by noting that programs (5.3) and (5.4) are *strong alternatives*; exactly one of the two programs is feasible. The key to solving either program is to find a finite parameterization of conditions (5.3a), (5.3b), and (5.4a). In Table 5.1, we provide a convenient reference for some common geometries.

Remark 6. *Problem* (5.3) *is frequently written with non-strict inequalities* (5.3a) *and* (5.3b) *to make it compatible with modern solvers. Such a formulation requires excluding the trivial solution* $(a,b) = (0,0)$ *via extra constraints as well as planes which are not strictly separating. The conditions given in Table 5.1 accomplish both with the constraint* $a^T x + b \geq 1$ *with* $x = v_i$ *for polytopic geometries and* $x = o$ *for spheres, cylinders, and capsules.*

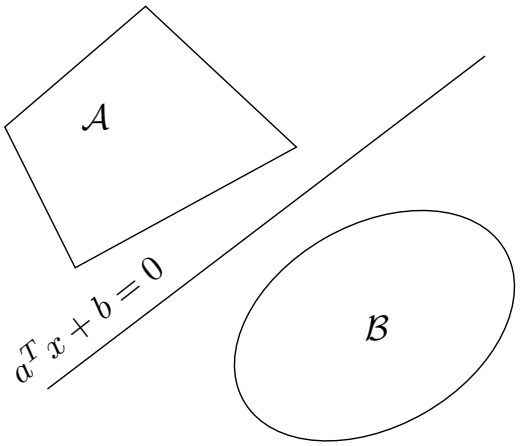
After the publication of our work, the authors of [152] proposed a different formulation for checking non-collision of two convex objects by minimizing a scaling of the two sets such that they still contain a common point. In their formulation, scalings more than one correspond to a certificate of non-collision and scalings smaller than one certify collisions. The proposed methods of Sections 5.4.1 and 5.4.2 can be directly extended to their formulation.

[4] Strictly speaking, the formulation (5.3a) given for the sphere, capsule, and cylinder only enforce non-strict separation i.e. $a^T x + b \geq 0$. This can be remedied by replacing r with $r + \varepsilon$ for any $\varepsilon > 0$.

Find $a,\ b$

$$a^T x + b > 0,\ \forall\ x \in \mathcal{A} \tag{5.3a}$$

$$a^T y + b < 0,\ \forall\ y \in \mathcal{B} \tag{5.3b}$$

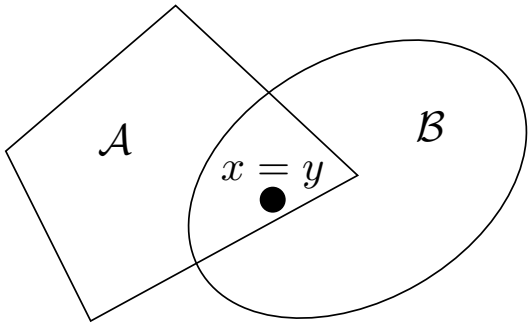


(a) If $\mathcal{A} \cap \mathcal{B} = \emptyset$ then there exists a hyperplane $a^T x + b = 0$ which separates the two bodies.

Find $x,\ y$ subject to

$$x \in \mathcal{A},\ y \in \mathcal{B} \tag{5.4a}$$

$$x = y \tag{5.4b}$$



(b) If $\mathcal{A} \cap \mathcal{B} \neq \emptyset$ then there exists $x \in \mathcal{A}$ and $y \in \mathcal{B}$ such that $x = y$.

Figure 5.1: Program (5.3) searches for a hyperplane that separates $\mathcal{A}$ and $\mathcal{B}$ while program (5.4) searches for a point in $\mathcal{A} \cap \mathcal{B}$. Both of these are convex optimization programs, and exactly one of these programs is feasible.

5.3.2 Rational Forward Kinematics

Our method in Section 5.4 will rely critically on parameterizing the forward kinematics of our robot using polynomials. Many robots contain rotational joints and so their forward kinematics are naturally specified as trigonometric functions. In this section, we review a standard change of variables of our robot kinematics that will enable us to parameterize the forward kinematics as a rational function.

The forward kinematics of a rigid-body robot with N joints can be written by composing rigid transforms [153,154]. Written in homogeneous coordinates, and using the monogram notation [154][5], the pose of a frame A, expressed in the reference frame F, as a function of the robot configuration q assumes the form:

$$ {}^{F}X^{A} = \begin{bmatrix} {}^{F}R^{A}(q) & {}^{F}p^{A}(q) \\ 0_{1\times 3} & 1 \end{bmatrix} = \prod_{i\in\mathcal{J}_{F,A}} {}^{P_i}X^{C_i}(q_i)\ {}^{C_i}X^{P_{i+1}} \tag{5.5} $$

In equation (5.5), $\mathcal{J}_{F,A} = \{i_1, \ldots, i_n\} \subseteq [N]$ is the set of joints lying on the kinematic chain between F and A. We attach two frames to each joint, with P_i rigidly fixed to the parent link of the i^{th} joint, and C_i rigidly fixed to the child link of the same joint. The two frames P_i and C_i coincide when the joint configuration $q_i = 0$. The subset of configuration variables q_i defines the degrees of freedom at the i^{th} joint, ${}^{P_i}X^{C_i}(q_i)$ is the relative transform of the joint after the joint moves by q_i. The rigid transform ${}^{C_i}X^{P_{i+1}}$ describes the physical properties of the i^{th} link such as its length. We assume that the reference frame F is the P_{i_1}, the parent frame of the first joint i_1; while the frame A is C_{i_n}, the child frame of the last joint i_n. [6] We choose to be explicit about the reference frame F at the risk of being pedantic, as the choice of reference frame F will have important consequences for the scalability of the approach described in Section 5.5 (see Appendix A.5.1 for a detailed discussion).

The matrices ${}^{P_i}X^{C_i}(q_i)$ assume the following forms [114]:

$$ {}^{P_i}X^{C_i}(q_i) = \begin{cases} \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & 0 \\ \sin(\theta_i) & \cos(\theta_i) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & \text{if } i^{\text{th}} \text{ joint is Revolute,} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & z_i \\ 0 & 0 & 0 & 1 \end{bmatrix} & \text{if } i^{\text{th}} \text{ joint is Prismatic,} \end{cases} \tag{5.6} $$

Expression (5.5) expresses the position of our robot as a *multilinear trigonometric polynomial function*. Concretely, the w^{th} component (where $w \in \{x, y, z\}$) of the position of A relative to F and expressed in F is an expression of the form:

$$ {}^{F}p^{A}_{w}(q) = \sum_{j} c_{jw} \prod_{i\in\mathcal{J}_{F,A}} \xi_{ij,w}(q_i) \tag{5.7} $$

[5] In monogram notation, the pose of a frame A expressed in a frame F is denoted as ${}^{F}X^{A}$.

[6] Since joint i_n is the last joint on this chain $\mathcal{J}_{F,A}$, we assume ${}^{C_{i_n}}X^{P_{i_n+1}} = I$.

with $\xi_{ij,w}(q_i) \in \{\cos(\theta_i), \sin(\theta_i), z_i\}$. The scalar constants c_{jw} are determined by the robot kinematic parameters (link length, joint axis, etc). Therefore, our configuration-space variables are

$$q = \bigcup_i \{\theta_i, z_i\}.$$

Multilinear trigonometric functions have many fortunate algebraic properties which we exploit throughout this chapter, the first of which will be a change of variables enabling us to write (5.7) as a rational function.

Specifically, we will introduce the substitution:

$$t_i \coloneqq \tan\left(\frac{\theta_i}{2}\right), \tag{5.8}$$

which allows us to write

$$\cos(\theta_i) = \frac{1 - t_i^2}{1 + t_i^2}, \quad \sin(\theta_i) = \frac{2t_i}{1 + t_i^2}.$$

This substitution is known as the stereographic projection [155] and is bijective if $\theta_i \in (-\pi, \pi)$, which we have assumed is the case for our robotic system. After performing this change of variables, our forward kinematics variables are

$$s = \bigcup_i \{t_i, z_i\}.$$

We refer to the configuration-space variable s as the *tangent-configuration-space* ($\mathcal{T}^{\text{space}}$) variable. Note that the tangent-configuration-space is *not* the tangent space of the configuration space.

In the $\mathcal{T}^{\text{space}}$ variable, our forward kinematics are a *rational function* with a polynomial numerator and a positive, polynomial denominator. This is an expression of the form

$${}^F p^A_w(s) = \sum_j c_{jw} \prod_{i \in \mathcal{J}_{F,A}} \frac{{}^F f^A_{ij,w}(s_i)}{{}^F g^A_{ij,w}(s_i)} = \frac{{}^F f^A_w(s)}{{}^F g^A_w(s)}, \quad w \in \{x, y, z\}, \tag{5.9}$$

where

$$\frac{{}^F f^A_{ij,w}(s_i)}{{}^F g^A_{ij,w}(s_i)} \in \left\{\frac{1 - t_i^2}{1 + t_i^2}, \frac{2t_i}{1 + t_i^2}, \frac{z_i}{1}\right\}.$$

We will abbreviate the vector quantity:

$${}^F p^A(s) = \frac{{}^F f^A(s)}{{}^F g^A(s)} \tag{5.10}$$

where ${}^F f^A(s)$ is a *vector of polynomials* and ${}^F g^A(s)$ is a single, positive polynomial. Notice that ${}^F g^A(s) > 0$ since each denominator ${}^F g^A_{ij,w}(s_i) \in \{1 + t_i^2, 1\}$, which are both strictly positive.

We emphasize again that we have assumed:

$$-\pi < \theta_{l,i} \leq \theta_i \leq \theta_{u,i} < \pi,$$
$$z_{l,i} \leq z_i \leq z_{u,i}.$$

and so generically $s_l \leq s \leq s_u$ component-wise. Therefore, our substitution between q and s is bijective so trajectories in $\mathcal{T}^{\text{space}}$ correspond unambiguously to trajectories in $\mathcal{C}^{\text{space}}$. Moreover, this assumption on boundedness of our configuration space allows us to seek collision-free regions $\mathcal{P}$ that are contained within $\mathcal{P}_{lim} := \{s \mid s_l \leq s \leq s_u\}$, a polytope encoding our joint limit.

Example 7. *As an example, we consider the double pendulum [156].*

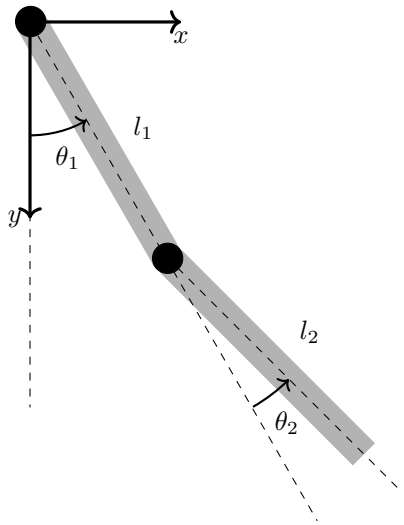


Figure 5.2: The forward kinematics of the double pendulum described in [156] can be described in the form (5.5).

The pose of the tip of the second pendulum can be written as:

$$\left[\begin{array}{c|c} R(\theta) & \begin{matrix} p_x(\theta) \\ p_y(\theta) \end{matrix} \\ \hline 0 & 1 \end{array}\right] = \begin{bmatrix} \cos(\theta_1) & \sin(\theta_1) & 0 \\ \sin(\theta_1) & -\cos(\theta_1) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & l_1 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} \cos(\theta_2) & \sin(\theta_2) & 0 \\ \sin(\theta_2) & -\cos(\theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & l_2 \\ 0 & 0 & 1 \end{bmatrix}.$$

The difference in the sign of the trigonometric part ensures that the y-axis is pointing down. Expanding out this product enables us to write the x coordinate of the tip of the system as:

$$p_x(\theta_1, \theta_2) = l_2(\sin(\theta_2)\cos(\theta_1) - \sin(\theta_1)\cos(\theta_2)) + l_1 \sin(\theta_1)$$
$$p_y(\theta_1, \theta_2) = l_2(\sin(\theta_1)\sin(\theta_2) + \cos(\theta_1)\cos(\theta_2)) - l_1 \cos(\theta_1)$$

Notice that these are multilinear trigonometric polynomials, i.e. no term contains $\cos(\theta_i)\sin(\theta_i)$ *for the same index* i*. We can perform the substitution given in* (5.8) *to express the position as a rational function:*

$$p_x(t_1,t_2) = \frac{2l_2(t_2(1-t_1)^2 - t_1(1-t_2)^2) + 2l_1t_1(1+t_2)^2}{(1+t_1^2)(1+t_2^2)}$$
$$p_y(t_1,t_2) = \frac{l_2(4t_1t_2 + (1-t_1)^2(1-t_2)^2) - l_1(1-t_1)^2(1+t_2)^2}{(1+t_1)^2(1+t_2)^2}.$$

Remark 7. *An alternative algebraic reparameterization of* (5.9) *introduces the substitutions* $\sigma_i := \sin(\theta_i)$ *and* $c_i := \cos(\theta_i)$ *with the constraint that* $\sigma_i^2 + c_i^2 = 1$ *and is known as the algebraic-configuration space (*$\mathcal{A}^{\text{space}}$*). All results in this chapter pertaining to* certifying *a region of* $\mathcal{T}^{\text{space}}$ *are readily converted to methods for* $\mathcal{A}^{\text{space}}$*. This is the focus of Section 5.4.*

However, in Section 5.5 we will be interested in growing regions inside $\mathcal{T}^{\text{free}}$*. In this case, we require notions of volume and size that are much more difficult to define analogously for* $\mathcal{A}^{\text{space}}$ *due to the quotient ring structure induced by* $\sigma_i^2 + c_i^2 = 1$*. Therefore, only the results of Section 5.4.4 are of practical interest for the* $\mathcal{A}^{\text{space}}$ *reparameterization.*

5.4 Certifying a Region is in $\mathcal{T}^{\text{free}}$

In this section, we consider the problem of certifying the non-collision of two convex bodies, $\mathcal{A}$ and $\mathcal{B}$, whose poses in task space are a function of the configuration of our robot. While programs (5.3) and (5.4) can be used to certify non-collision between $\mathcal{A}$ and $\mathcal{B}$ for any fixed configuration, they are insufficient to certify $\mathcal{A}$ and $\mathcal{B}$ do not intersect for all configurations in an entire region $\mathcal{P}$ of the configuration space. Therefore, in Sections 5.4.1 and 5.4.2, we show how to combine the ingredients of Sections 2.1 and 5.3 to generalize programs (5.3) and (5.4).

The presence of trigonometric functions when the forward kinematics are expressed in the variable q precludes using SOS programming, our tool of choice. Therefore, we assume that $\mathcal{A}(s)$ and $\mathcal{B}(s)$ are convex sets in task space with their poses expressed as *rational functions* in the $\mathcal{T}^{\text{space}}$ variable s. This can be achieved using the developments in Section 5.3.2. Our objective is to certify that $\mathcal{A}(s)$ and $\mathcal{B}(s)$ do not intersect for all $s \in \mathcal{P} = \{s \mid Cs \leq d\} \subseteq \mathcal{P}_{lim} = \{s \mid s_l \leq s \leq s_u\}$.

Under these assumptions, the generalizations of (5.3) and (5.4) respectively take the form of certifying a polynomial implication and certifying the emptiness of a basic-semialgebraic set respectively. We give a formulation of each as a SOS program. We conclude in Section 5.4.3 by proving that feasibility of our convex optimization programs is both necessary and sufficient for $\mathcal{P}$ to be collision-free.

5.4.1 Method via Parametrized Hyperplane Certificates of Non-Collision

In this section, we generalize (5.3) and use SOS to search for a polynomial family of hyperplanes parametrized by the $\mathcal{T}^{\text{space}}$ variable s which will certify the non-collision of $\mathcal{A}(s)$ and

$\mathcal{B}(s)$ for all $s \in \mathcal{P} = \{s \mid Cs \leq d\}$, where C and d are the given matrix/vector in this section. In Section 5.5 we search for C and d.

We begin by remarking that even if $\mathcal{A}(s)$ and $\mathcal{B}(s)$ do not collide for all $s \in \mathcal{P}$, there may not be a single, static hyperplane $\mathcal{H} = (a, b)$ that certifies this fact. An example of this can be seen in Figure 5.3.

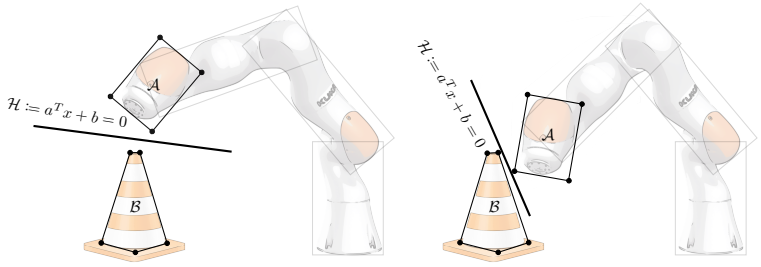


Figure 5.3: The convex collision geometries $\mathcal{A}(s)$ and $\mathcal{B}(s)$ are collision-free if and only if there exists a family of hyperplanes $\mathcal{H}(s)$ separating the two for each configuration s_0. The planes act as a certificate of non-collision.

We therefore will look for a *polynomial family* of hyperplanes $\mathcal{H}(s) = \{x \mid a(s)^T x + b(s) = 0\}$ parametrized by our $\mathcal{T}^{\text{space}}$ variable s. Inspection of Table 5.1 shows that we must generalize

$$s \in \mathcal{P} \implies a^T(s)\ {}^F p^v(s) + b(s) \geq 1, \tag{5.11}$$

for particular points v specific to each of the geometries, and

$$s \in \mathcal{P} \implies a^T(s)\ {}^F p^o(s) + b(s) \geq r \left\| a(s) \right\|, \tag{5.12}$$

for center o if $\mathcal{A}(s)$ is either a sphere or capsule. The generalization of the conditions for the cylinder are similar to those of the sphere and capsule, and so we defer its complete derivation to Appendix A.4.

To generalize (5.11) and (5.12), we recall that the position of any point $A \in \mathcal{A}(s)$ (and similarly $\mathcal{B}(s)$) can be expressed as a rational function ${}^F p^A(s) = \frac{{}^F f^A(s)}{{}^F g^A(s)}$ where ${}^F g^A(s) > 0$.

Therefore, we can express (5.11) as:

$$s \in \mathcal{P} \implies a^T(s)\ {}^F f^v(s) + (b(s) - 1)\ {}^F g^v(s) \geq 0 \tag{5.13}$$

This is a polynomial implication of the form (2.2). As $\mathcal{P} \subseteq \mathcal{P}_{lim}$ is a compact polytope, $\mathcal{P}$ is Archimedean [148, Theorem 7.1.3] and so we can use Theorem 12 to express condition (5.11) as:

$$a^T(s)\ {}^F f^v(s) + (b(s) - 1)\ {}^F g^v(s) = \lambda_{01}(s) + \sum_{j=1}^{m} \lambda_{j1}(s)(d_j - c_j^T s) \tag{5.14}$$

where $\lambda_{j1}, j = 0, \ldots, m$ are all SOS polynomials.

The condition (5.12), can be expressed as a polynomial matrix inequality using the Schur complement[7] [42]

$$s \in \mathcal{P} \implies \begin{bmatrix} \left((a(s))^T \ {}^F f^o(s) + b(s) \ {}^F g^o(s)\right) I_3 & ra(s) \ {}^F g^o(s) \\ r(a(s))^T \ {}^F g^o(s) & (a(s))^T \ {}^F f^o(s) + b(s) \ {}^F g^o(s) \end{bmatrix} \succeq 0. \quad (5.15)$$

This is known as a *matrix SOS* condition which can be represented as a set of semidefinite constraints [157]. Specifically, by introducing a vector auxiliary variable u, we can write (5.15) as:

$$s \in \mathcal{P}, u^T u = 1 \implies u^T \begin{bmatrix} \left(a^T(s) \ {}^F f^o(s) + b(s) \ {}^F g^o(s)\right) I_3 & ra(s) \ {}^F g^o(s) \\ r(a(s))^T \ {}^F g^o(s) & (a(s))^T \ {}^F f^o(s) + b(s) \ {}^F g^o(s) \end{bmatrix} u \geq 0 \quad (5.16)$$

which can be expressed as the SOS condition:

$$u^T \begin{bmatrix} \left((a(s))^T \ {}^F f^o(s) + b(s) \ {}^F g^o(s)\right) I_3 & ra(s) \ {}^F g^o(s) \\ r(a(s))^T \ {}^F g^o(s) & (a(s))^T \ {}^F f^o(s) + b(s) \ {}^F g^o(s) \end{bmatrix} u = \lambda_{02}(u,s) + \sum_{j=1}^{m} \lambda_{j2}(u,s)(d_j - c_j^T s) + \phi(u,s)(1 - u^T u) \quad (5.17)$$

where λ_{j2} are all SOS polynomials, and $\phi \in \mathbb{R}[u,s]$. We introduce the additional equality $u^T u = 1$ to ensure the description of the set $\{(u,s) | s \in \mathcal{P}, u^T u = 1\}$ is Archimedean.

We are now ready to describe our convex program certifying that $\mathcal{P}$ is a region of $\mathcal{T}^{\text{space}}$ containing no collision. For each pair of bodies $\mathcal{A}(s)$ and $\mathcal{B}(s)$ that can collide in the scene, we search for a polynomial hyperplane via the optimization program:

$$\forall \text{ pairs } \mathcal{A}, \mathcal{B} \quad (5.18a)$$

$$\textbf{find } a_{\mathcal{A},\mathcal{B}}, b_{\mathcal{A},\mathcal{B}} \ \textbf{ subject to} \quad (5.18b)$$

$$\forall \ s \in \mathcal{P}, \ a_{\mathcal{A},\mathcal{B}}^T(s)x + b_{\mathcal{A},\mathcal{B}}(s) > 0, \ \forall x \in \mathcal{A}(s) \quad (5.18c)$$

$$\forall \ s \in \mathcal{P}, \ a_{\mathcal{A},\mathcal{B}}^T(s)y + b_{\mathcal{A},\mathcal{B}}(s) < 0 \ \forall y \in \mathcal{B}(s) \quad (5.18d)$$

$$\lambda_{ij}^{\mathcal{A},\mathcal{B}}(u,s), \ \mu_{ij}^{\mathcal{A},\mathcal{B}}(u,s) \in \mathbf{\Sigma}, \ \phi^{\mathcal{A},\mathcal{B}}(u,s), \ \chi^{\mathcal{A},\mathcal{B}}(u,s) \in \mathbb{R}[u,s] \quad (5.18e)$$

where $(a_{\mathcal{A},\mathcal{B}}(s), b_{\mathcal{A},\mathcal{B}}(s))$ are the parameters of the polynomial hyperplane separating $\mathcal{A}$ and $\mathcal{B}$, the polynomials $\lambda_{ij}^{\mathcal{A},\mathcal{B}}(s)$ and $\phi^{\mathcal{A},\mathcal{B}}(s)$ collect all the multiplier polynomials for enforcing (5.18c), and $\mu_{ij}^{\mathcal{A},\mathcal{B}}(s)$ and $\chi^{\mathcal{A},\mathcal{B}}(s)$ collect all the multiplier polynomials for enforcing (5.18d) by using (5.14) and (5.17) depending on the geometry of $\mathcal{A}$ and $\mathcal{B}$. We stress in the above program that the decision variables are the *coefficients* of the polynomials $a_{\mathcal{A},\mathcal{B}}$, $b_{\mathcal{A},\mathcal{B}}$, and the multiplier polynomials. The symbols u and s are known as *indeterminates* and are not explicitly searched over.

[7] We have that $\gamma \geq r \|a\|$ if and only if the Schur complement $\begin{bmatrix} \gamma I_3 & ra \\ ra^T & \gamma \end{bmatrix} \succeq 0$.

In Table A.2, we summarize the conditions for enforcing (5.18c) and (5.18d) for common families of sets. We call a feasible solution to (5.18) a *certificate* for the polytope $\mathcal{P}$ which we denote:

$$\mathcal{C}_{\mathcal{P}} = \bigcup_{(\mathcal{A},\mathcal{B})} \{a_{\mathcal{A},\mathcal{B}}(s),\ b_{\mathcal{A},\mathcal{B}}(s),\ \lambda_{ij}^{\mathcal{A},\mathcal{B}}(u,s),\ \phi^{\mathcal{A},\mathcal{B}}(u,s),\ \mu_{ij}^{\mathcal{A},\mathcal{B}}(u,s),\ \chi^{\mathcal{A},\mathcal{B}}(u,s)\} \tag{5.19}$$

5.4.2 Method via Polynomial Infeasibility Certificates

As we remarked in Section 5.3.1, non-collision of two convex shapes $\mathcal{A}$ and $\mathcal{B}$ can be checked by certifying the *infeasibility* of (5.4). The infeasibility of (5.4) can be extended to the case when the locations of $\mathcal{A}(s)$ and $\mathcal{B}(s)$ are a function of s.

$$\textbf{Certify that } \nexists\ s \in \mathcal{P},\ x, y \in \mathbb{R}^3 \textbf{ such that} \tag{5.20a}$$
$$x \in \mathcal{A}(s), y \in \mathcal{B}(s) \tag{5.20b}$$
$$x = y \tag{5.20c}$$

An equivalent, and perhaps more instructive, way of expressing (5.20) begins by letting $\gamma_i^{\mathcal{A}}(s,x,u_{\mathcal{A}})$ and $h_j^{\mathcal{A}}(s,x,u_{\mathcal{A}})$ be the polynomials encoding the condition that $x \in \mathcal{A}(s)$ with $u_{\mathcal{A}}$ collecting any extra variables needed to write this condition. Similarly, $u_{\mathcal{B}}$, $\gamma_k^{\mathcal{B}}(s,x,u_{\mathcal{B}})$, and $h_l^{\mathcal{B}}(s,x,u_{\mathcal{B}})$ encode that $x \in \mathcal{B}(s)$. We provide explicit expressions for $\gamma_i^{\mathcal{A}}, \gamma_k^{\mathcal{B}}$ and $h_j^{\mathcal{A}}, h_l^{\mathcal{B}}$ in Table A.3 (given in Appendix A.2) for a few common geometries. We now consider the set

$$\mathcal{S}_{\mathcal{P},\mathcal{A},\mathcal{B}} = \{x, s \mid s \in \mathcal{P},\ x \in \mathcal{A}(s),\ x \in \mathcal{B}(s)\} \tag{5.21}$$

$$= \left\{ x,\ s,\ u_{\mathcal{A}},\ u_{\mathcal{B}} \;\middle|\; \begin{array}{c} Cs \le d, \\ \gamma_i^{\mathcal{A}}(s,x,u_{\mathcal{A}}) \ge 0,\ h_j^{\mathcal{A}}(s,x,u_{\mathcal{A}}) = 0 \\ \gamma_k^{\mathcal{B}}(s,x,u_{\mathcal{B}}) \ge 0, h_l^{\mathcal{B}}(s,x,u_{\mathcal{B}}) = 0, \\ i \in [n_{\mathcal{A}}],\ j \in [m_{\mathcal{A}}],\ k \in [n_{\mathcal{B}}],\ l \in [m_{\mathcal{B}}] \end{array} \right\}, \tag{5.22}$$

and consider the problem

$$\textbf{Certify that } \mathcal{S}_{\mathcal{P},\mathcal{A},\mathcal{B}} = \emptyset. \tag{5.23}$$

Problems of the form (5.23) can be solved using the following SOS program.

Theorem 15 ([158]). *Suppose $\mathcal{S}_{g,h}$ is Archimedean. Then $\mathcal{S}_{g,h} = \emptyset$ if and only if there exist polynomials $\phi_j(x)$ and SOS polynomials $\lambda_i(x)$ such that*

$$-1 = \lambda_0(x) + \sum_i \lambda_i(x) g_i(x) + \sum_j \phi_j(x) h_j(x). \tag{5.24}$$

Example 8. *If $\mathcal{A}$ is a polytope with $n_{\mathcal{A}}$ vertices given by $v_{\mathcal{A}_i}$, and $\mathcal{B}$ is a sphere with center*

$o_{\mathcal{B}}$ and radius $r_{\mathcal{B}}$, then we can write

$$\mathcal{S}_{\mathcal{P},\mathcal{A},\mathcal{B}} = \left\{ x,\ s,\ \mu_{\mathcal{A}_i} \;\middle|\; \begin{array}{c} Cs \leq d, \\ \left(\prod_i {}^F g^{v_{\mathcal{A}_i}}\right)\left(x - \sum_{i=1}^{n_{\mathcal{A}}} \mu_{\mathcal{A}_i}\left(\frac{{}^F f^{v_{\mathcal{A}_i}}(s)}{{}^F g^{v_{\mathcal{A}_i}}(s)}\right)\right) = 0, \\ 1 - \sum_{i=1}^{n_{\mathcal{A}}} \mu_{\mathcal{A}_i} = 0, \\ \mu_{\mathcal{A}_i} \geq 0 \ \forall\ i \in [n_{\mathcal{A}}], \\ \left({}^F g^{o_{\mathcal{B}}}(s)\right)^2 \left(r_{\mathcal{B}}^2 - \left\| x - \frac{{}^F f^{o_{\mathcal{B}}}(s)}{{}^F g^{o_{\mathcal{B}}}(s)} \right\|^2\right) \geq 0 \end{array} \right\}.$$

Now, we note that $\mathcal{S}_{\mathcal{P},\mathcal{A},\mathcal{B}}$ is an Archimedean set. This implies that we can use Theorem 15 to write (5.23) as an optimization problem. Denoting $u = \{u_{\mathcal{A}}, u_{\mathcal{B}}\}$, this can be written explicitly as

$$\textbf{find } \lambda_0,\ \lambda_j^{\mathcal{P}},\ \lambda_j^{\mathcal{A}},\ \lambda_j^{\mathcal{B}},\ \phi_k^{\mathcal{A}},\ \phi_k^{\mathcal{B}} \tag{5.25a}$$

$$\begin{aligned} -1 = \lambda_0(s,x,u) + \sum_{j=1}^{n} \lambda_j^{\mathcal{P}}(s,x,u)(d_j - c_j^T s) + \\ \sum_{i=1}^{n_{\mathcal{A}}} \lambda_i^{\mathcal{A}}(s,x,u)\gamma_i^{\mathcal{A}}(s,x,u_{\mathcal{A}}) + \sum_{j=1}^{m_{\mathcal{A}}} \phi_j^{\mathcal{A}}(s,x,u)h_j^{\mathcal{A}}(s,x,u_{\mathcal{A}}) + \\ \sum_{l=1}^{n_{\mathcal{B}}} \lambda_l^{\mathcal{B}}(s,x,u)\gamma_l^{\mathcal{B}}(s,x,u_{\mathcal{B}}) + \sum_{k=1}^{m_{\mathcal{B}}} \phi_k^{\mathcal{B}}(s,x,u)h_k^{\mathcal{B}}(s,x,u_{\mathcal{B}}) \end{aligned} \tag{5.25b}$$

$$\lambda_0,\ \lambda_j^{\mathcal{P}},\ \lambda_i^{\mathcal{A}},\ \lambda_l^{\mathcal{B}} \in \mathbf{\Sigma} \tag{5.25c}$$

$$\phi_j^{\mathcal{A}},\ \phi_k^{\mathcal{B}} \in \mathbb{R}[s,x,u] \tag{5.25d}$$

We again emphasize that in program (5.25) the decision variables are the coefficients of λ_0, $\lambda_j^{\mathcal{P}}$, $\lambda_i^{\mathcal{A}}$, $\lambda_l^{\mathcal{B}}$, $\phi_j^{\mathcal{A}}$, and $\phi_k^{\mathcal{B}}$, while the symbols $\{x, s, u\}$ are not decision variables but rather polynomial indeterminates. Similar to the program (5.18), a certificate of non-collision can be obtained by solving (5.25) for each pair $(\mathcal{A}, \mathcal{B})$ with the multipliers acting as the certificate.

5.4.3 Power of the Certification Programs

In this section, we consider the power of both certification programs. Specifically, in Sections 5.4.1 and 5.4.2 we argued that feasibility of (5.18) and (5.25) are sufficient to prove that $\mathcal{P}$ is collision-free. In this section, we present two theorems showing that the feasibility of these programs is also *necessary*.

Such a result is important given the fact that, as stated, (5.18) and (5.25) are infinite dimensional and therefore, in practice must be solved by selecting a basis of finite degree for the polynomials. Other subtleties about the power of our formulation are discussed in Section 5.9.1. Fortunately, we can prove that there do exist finite degrees such that both programs become feasible when $\mathcal{P}$ is truly collision-free.

Theorem 16. *Let all multiplier polynomials from (5.18) have degree at least ρ and let all the polynomials in the parameterization of the hyperplane have degree at least κ. Suppose $\mathcal{P} \subseteq \mathcal{P}_{lim}$ is a subset of $\mathcal{T}^{\text{free}}$.*

Then there exists finite ρ and κ sufficiently large such that (5.18) is feasible.

A similar theorem can be stated for the program in (5.25).

Theorem 17. *Let $\mathcal{P} \subseteq \mathcal{P}_{lim}$ be a compact, polytopic subset of $\mathcal{T}^{\text{free}}$ and let all multiplier polynomials from (5.25) have degree at least ρ. There exists a finite ρ sufficiently large such that (5.25) is feasible.*

We delay the proofs and further discussion of these results to Section 5.9.1. For now, we simply remark that Theorems 16 and 17 assert that the certification programs presented in this section are both complete in the sense that any collision-free polytope $\mathcal{P}$ can be certified with our technique.

5.4.4 Specialization to Trajectories

In this section, we consider the specialization of the method in Section 5.4.1 to the case of certifying that a given trajectory in $\mathcal{T}^{\text{free}}$ is collision free. We denote the trajectory

$$\omega(\tau) : [0, 1] \mapsto s \tag{5.26}$$

and assume that $\omega(\tau)$ is a polynomial, for example a cubic spline or Bezier curve.

Substituting (5.26) into (5.9) yields a univariate rational expression ${}^{F}p^{A}(\omega(\tau))$ parametrizing the restriction of the kinematics to the trajectory. The domain of this univariate function is an interval, a one-dimensional polytope.

We can therefore immediately use the machinery of either Section 5.4.1 or Section 5.4.2. However, two stronger representation theorems hold in the univariate polynomial case.

Theorem 18 (Markov-Lucasz [159])**.** *A univariate polynomial $p(x)$ is non-negative on the non-empty interval $x \in [a, b]$ if and only if it can be expressed as*

$$p(x) = \begin{cases} \lambda(x) + (x-a)(b-x)\nu(x), & \deg(p) = 2d \\ (x-a)\lambda(x) + (b-x)\nu(x), & \deg(p) = 2d+1 \end{cases} \tag{5.27}$$

where $\lambda,\ \nu \in \mathbf{\Sigma}[x]$. Moreover, if $d = \left\lfloor \frac{\deg(p)}{2} \right\rfloor$ then

$$\deg(\lambda) \leq 2d, \quad \deg(\nu) \leq \begin{cases} 2d-2 & \text{if } \deg(p) = 2d \\ 2d & \text{if } \deg(p) = 2d+1 \end{cases}. \tag{5.28}$$

An analogous theorem holds in the univariate matrix case.

Theorem 19. *Let $P(x) \in \mathbb{R}[x]^{m \times m}$ be a symmetric matrix of univariate polynomials. Then $P(x) \succeq 0$ on the non-empty interval $x \in [a, b]$ if and only if $p(x, y) = y^T P(x) y$ can be expressed as:*

$$p(x,y) = \begin{cases} \lambda(x,y) + (x-a)(b-x)\nu(x,y), & \deg(P) = 2d \\ (x-a)\lambda(x,y) + (b-x)\nu(x,y), & \deg(P) = 2d+1 \end{cases} \tag{5.29}$$

where $\lambda,\ \nu \in \mathbf{\Sigma}[x, y]$. *Moreover, if* $d = \left\lfloor \frac{\deg(P)}{2} \right\rfloor$ *then*

$$\deg_{y_i}(\lambda) = 2,\ \deg_{y_i}(\nu) = 2,\ \deg_x(\lambda) \leq 2d$$
$$\deg_x(\nu) \leq \begin{cases} 2d-2 & \text{if } \deg(P) = 2d \\ 2d & \text{if } \deg(P) = 2d+1 \end{cases}. \tag{5.30}$$

Proof 5.1. This follows by applying a similar argument as used in [159] to prove Theorem 18 to the matrix $P(x)$ and leveraging the result of [160] that univariate PSD matrices are always SOS matrices. □

The main difference between Theorems 18 and 19 compared to the representation theorems in Section 2.1.1 are the explicit degree bounds in (5.29) and (5.27).

To transfer these results to the method of Section 5.4.1, note that after substitution of the trajectory into the forward kinematics, the polynomial separating hyperplane certificates (5.18c) and (5.18d) become univariate. Therefore, after fixing the degree of the trajectory, the only hyperparameters left to choose in the certification program (5.19) are the degrees of the hyperplanes. Theorems 18 and 19 tell us that choosing the degree of the certificate polynomials to match the degree of the overall expression is necessary and sufficient. This is in contrast to the general method where choosing a larger degree may produce better certificates at the cost of longer solver times. Similar results hold if we instead use the formulation from Section 5.4.2.

The computational advantages of the univariate case are explored in Section 5.6.4.

5.5 Growing Regions in $\mathcal{T}^{\text{free}}$

In this section, we describe our algorithm for rapidly generating a certified, polyhedral decomposition of $\mathcal{T}^{\text{free}}$. Our algorithm can be seen as a generalization of the IRIS algorithm of [145] to non-convex $\mathcal{T}^{\text{space}}$ obstacles and so we name it C-IRIS (Configuration-Space, Iterative Regional Inflation by Semidefinite programming). The key idea is to iteratively grow certified convex polytopes of increasing size around various important configurations in the $\mathcal{T}^{\text{space}}$[8]. This is achieved by solving a series of convex optimization programs. The complete algorithm is summarized in Algorithm 3.

We begin by discussing how we measure the size of our polytope $\mathcal{P} = \{s \mid Cs \leq d\}$. While it may be attractive to measure the size of a polytope by its volume, it is known that computing the volume of a half-space representation (H-Rep) polytope is #P-hard[9] [162] and therefore intractable as an objective. A useful surrogate for the volume of $\mathcal{P}$ used in [145] is the volume of the maximum volume inscribed ellipse of $\mathcal{P}$: the set $\mathcal{E}_{\mathcal{P}} = \{Qs + s_0 \mid \|s\|_2 \leq 1\}$ where Q is a positive semidefinite matrix describing the shape of the ellipsoid and s_0 its center. The problem of finding the maximum volume inscribed ellipsoid in a polytope is a

[8] Note that a large polytope in the $\mathcal{T}^{\text{space}}$ does not necessarily correspond to a large region in $\mathcal{C}^{\text{space}}$. The $\mathcal{T}^{\text{space}}$ variable s is a good approximation of the C-space variable $\theta/2$ near $\theta = 0$. This approximation becomes looser as θ approaches $\pm\pi$.

[9] #P-hard problems are at least as hard as NP-complete problems [161].

semidefinite program described in [42, Section 8.4.2].

$$\max_{Q,s_0} \ \text{logdet}(Q) \ \text{ subject to} \tag{5.31a}$$
$$\|Qc_i\|_2 \leq d_i - c_i^T s_0 \ \ \forall\ i \in [m] \tag{5.31b}$$
$$Q \succeq 0 \tag{5.31c}$$

As we wish our polytopes to cover diverse areas of $\mathcal{T}^{\text{free}}$, we will grow each polytope $\mathcal{P}$ around some nominal configuration s_s we call the seed point. We will describe a method for choosing a good set seed points in Section 5.7. The polytope $\mathcal{P}$ is required to contain s_s as it grows.

A maximal volume, certified polytope around s_s can be obtained by solving the following optimization program which combines the ellipsoidal program (5.31) with the certification program (5.18) from Section 5.4.1.

$$\max_{\substack{Q,s_0,C,d,\\ \forall(\mathcal{A},\mathcal{B})\\ \lambda_{ij}^{\mathcal{A},\mathcal{B}},\ \phi^{\mathcal{A},\mathcal{B}},\\ \mu_{ij}^{\mathcal{A},\mathcal{B}},\ \chi^{\mathcal{A},\mathcal{B}}\\ a_{\mathcal{A},\mathcal{B}},\ b_{\mathcal{A},\mathcal{B}}}} \ \text{logdet}(Q) \ \text{ subject to} \tag{5.32a}$$
$$(5.31b), (5.31c) \tag{5.32b}$$
$$Cs_s \leq d \tag{5.32c}$$
$$\|c_i\|_2 \leq 1 \ \forall\ i \in [m] \tag{5.32d}$$
$$(5.18c),\ (5.18d),\ (5.18e) \tag{5.32e}$$

The condition $\mathcal{E}_\mathcal{P} \subset \mathcal{P}$ is given by the constraints (5.32b). Constraint (5.32c) enforces that $\mathcal{P}$ grows around s_s. The added constraint (5.32d) prevents numerically undesirable scaling. Finally, (5.32e) enforces that we search for hyperplanes $(a_{\mathcal{A},\mathcal{B}}(s), b_{\mathcal{A},\mathcal{B}}(s))$ which separate each collision pair $\mathcal{A}(s)$ and $\mathcal{B}(s)$.

While this program is attractive as a specification, it is not convex due to bilinearity between Q and c_i in (5.31b) and the bilinearity between the multipliers and the defining equations of $\mathcal{P}$ implicit in (5.32e) (see Section 5.4.1). This bilinearity precludes simultaneous search of the polytope $\mathcal{P}$, inscribed ellipsoid $\mathcal{E}_\mathcal{P}$, and the corresponding certificate $\mathcal{C}_\mathcal{P}$ using convex programming. Therefore, we approximate the solution to (5.32) by alternating between two convex programs; one which generates certificates of non-collision and one which improves our polytope without violating the previous certificate.

Remark 8. *It is possible to replace* (5.32e) *with the equivalent constraints from program* (5.25). *We prefer to base our algorithm on* (5.18) *as it can be visualized (i.e. planes in the task space) and the polynomials contain fewer indeterminates. Hence the optimization problem size is smaller. Also, the separating planes approach produces separating certificates with quantifiable margins by measuring the distance from the collision geometries to the plane in task space.*

We begin by demonstrating how a certified polytopic region can be improved. Suppose that a convex polytope $\mathcal{P} = \{s \mid Cs \leq d\}$ has been certified with certificate $\mathcal{C}_\mathcal{P}$ and the

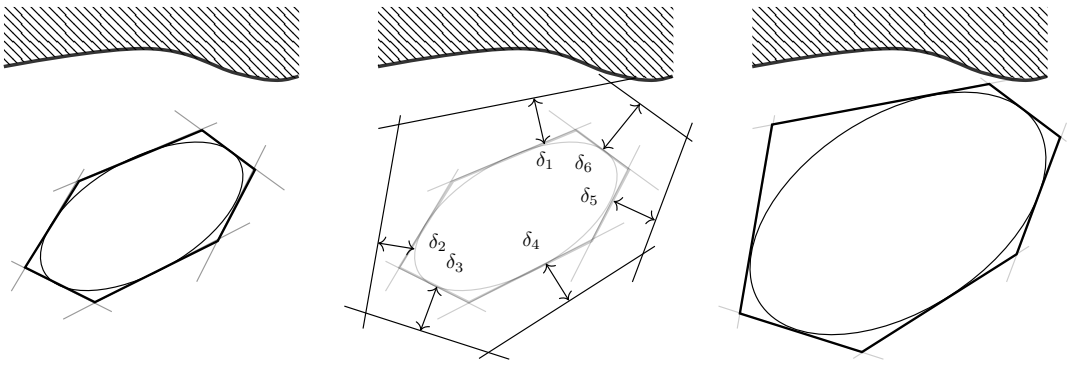


Figure 5.4: In (5.33) we search for the maximum distance the polytope's faces can be pushed away from the current inscribed ellipse without violating the certificate found in the previous step.

maximum inscribed ellipse $\mathcal{E}_\mathcal{P}$ has been computed using (5.31). A new, larger polytope $\mathcal{P}'$ can be found by solving the convex optimization program (5.33) which pushes the faces of $\mathcal{P}'$ as far away from the surface of $\mathcal{E}_\mathcal{P}$ as possible without violating the certificate $\mathcal{C}_\mathcal{P}$. This procedure is visualized in Figure 5.4.

This can be achieved with the following optimization program:

$$\max_{\substack{C,d,\delta,\\ \forall(\mathcal{A},\mathcal{B})\\ \lambda_{01}^{\mathcal{A},\mathcal{B}},\ \lambda_{02}^{\mathcal{A},\mathcal{B}},\ \phi^{\mathcal{A},\mathcal{B}}\\ \mu_{01}^{\mathcal{A},\mathcal{B}},\ \mu_{02}^{\mathcal{A},\mathcal{B}},\ \chi^{\mathcal{A},\mathcal{B}}\\ a_{\mathcal{A},\mathcal{B}},\ b_{\mathcal{A},\mathcal{B}}}} \prod_{i=1}^{m}(\delta_i+\varepsilon_0) \text{ subject to} \tag{5.33a}$$

$$\|Qc_i\|_2 \leq d_i - \delta_i - c_i^T s_0,\ \delta_i \geq 0\ \forall\ i \in [m] \tag{5.33b}$$

$$(5.32c), (5.32d), (5.32e)\ \forall \text{pairs}\ (\mathcal{A}(s), \mathcal{B}(s)) \tag{5.33c}$$

where $\varepsilon_0 > 0$ is some positive constant ensuring that the objective is never 0 (hence even if one of the margins δ_i is 0, the optimization will still attempt to increase the other margins so as to increase the objective). We use the product $\prod_{i=1}^m(\delta_i+\epsilon_0)$ instead of the sum $\sum_{i=1}^m(\delta_i+\epsilon_0)$ as the objective function, to encourage a more uniform increase on every margin δ_i, which leads to a larger inscribed ellipsoid. We recall that (5.33c) is either a constraint of the form (5.14) or (5.17). We emphasize that in (5.33), $\lambda_{i1}, \lambda_{i2}, \mu_{i1}, \mu_{i2}, i \geq 1$ are all fixed and it is the variables c_j and d_j which are searched over.

Our complete algorithm proceeds in three steps; first, an initial, collision-free polytope $\mathcal{P}_0$ containing a seed point s_s is certified using (5.19) to obtain $\mathcal{C}_{\mathcal{P}_0}$. Next, the maximum inscribed ellipsoid $\mathcal{E}_{\mathcal{P}_0}$ is computed using (5.31). Finally, $\mathcal{P}_0$ is improved using (5.33) to obtain a new polytope $\mathcal{P}_1$. This polytope $\mathcal{P}_1$ has the same number of defining inequalities as $\mathcal{P}_0$. We iterate this process until the volume of $\mathcal{E}_\mathcal{P}$ stops improving. This algorithm is formalized in Algorithm 3. Every step of this process involves solving a *convex program* for which very fast commercial solvers exist [22,85,163].

The initial seed polytope can be constructed in a variety of manners. One practical strategy we employ is to use one of the fast, but optimistic methods IrisNp2 and IrisNp

Algorithm 3: Given an initial polytopic region $\mathcal{P}_0$ and seed point $s_s \in \mathcal{P}_0$ for which (5.32) is feasible, return a new polytopic region $\mathcal{P}_i$ with a maximal inscribed ellipse $\mathcal{E}_{\mathcal{P}_i}$ with larger volume than $\mathcal{E}_{\mathcal{P}_0}$ and a collision-free certificate $\mathcal{C}_{\mathcal{P}_i}$.

3.1 $i \leftarrow 0$
3.2 **do**
3.3 $\quad \mathcal{C}_{\mathcal{P}_i} \leftarrow$ Solution of (5.18) with data $\mathcal{P}_i$
3.4 $\quad \mathcal{E}_{\mathcal{P}_i} \leftarrow$ Solution of (5.31) with data $\mathcal{P}_i$
3.5 $\quad (\mathcal{P}_{i+1}, \mathcal{C}_{\mathcal{P}_{i+1}}) \leftarrow$ Solution of (5.33) with data $(\mathcal{E}_{\mathcal{P}_i}, \mathcal{C}_{\mathcal{P}_i})$
3.6 $\quad i \leftarrow i + 1$
3.7 **while** $\left(\boldsymbol{vol}(\mathcal{E}_{\mathcal{P}_i}) - \boldsymbol{vol}(\mathcal{E}_{\mathcal{P}_{i-1}})\right) / \boldsymbol{vol}(\mathcal{E}_{\mathcal{P}_{i-1}}) \geq \textit{tolerance}$
3.8 **return** $(\mathcal{P}_i, \mathcal{C}_{\mathcal{P}_i})$

from [146,147,164] to propose an initial, large region. As this may contain collisions, we then scale the polytope around its Chebyshev center by a factor of α and attempt to certify the region using (5.19). By bisecting on the value α, we are able to find a large, initial collision-free polytope.

Remark 9. *Some practical considerations for improving the runtime of Algorithm 3 are discussed in the appendices. Specifically, in Appendix A.5 we expand on design choices that substantially impact the size of the optimization programs as well as which parts of Algorithm 3 can be parallelized.*

5.6 Results

We demonstrate the use of Algorithm 3 on systems of varying complexity. We begin with very simple robots where both the task space and configuration space can be visualized and demonstrate that our algorithm can find very large portions of $\mathcal{T}^{\text{space}}$ and achieve near-complete coverage for simple systems in reasonable time.

We then demonstrate the use of Algorithm 3 on various robots commonly found in industry. These include a KUKA iiwa reaching into a shelf, a bimanual KUKA iiwa, and similar setups for the UR3e. Our objective is to show the scalability of our algorithm in realistic settings, as well as demonstrate the diversity of shapes our approach can handle.

A mature implementation of our algorithm is available in the open-source robotics toolbox Drake[10] [69]. We furthermore provide examples of our algorithm in interactive Python notebooks[11]. Animations of various figures in this section can also be found on this project's website[12].

The implementation details of all experiments in this section, such as the choice of reference frame for each plane, the degree of the polynomials parametrizing the hyperplanes, and

[10] https://drake.mit.edu/

[11] https://deepnote.com/workspace/alexandre-amice-c018b305-0386-4703-9474-01b867e6efea/project/C-IRIS-7e82e4f5-f47a-475a-aad3-c88093ed36c6/notebook/2d_example_bilinear_alternation-14f1ee8c795e499ca7f577b6885c10e9

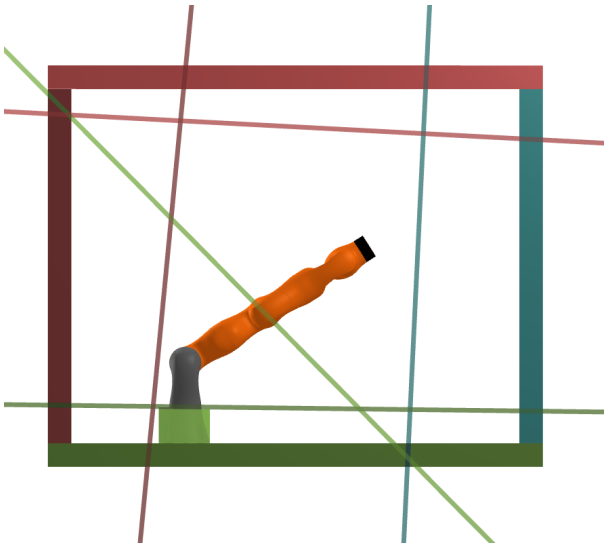
[12] https://alexandreamice.github.io/project/c-iris

the degree of the multiplier polynomials in each program are expounded on in Appendix A.5. The results of this section are from [115,116].

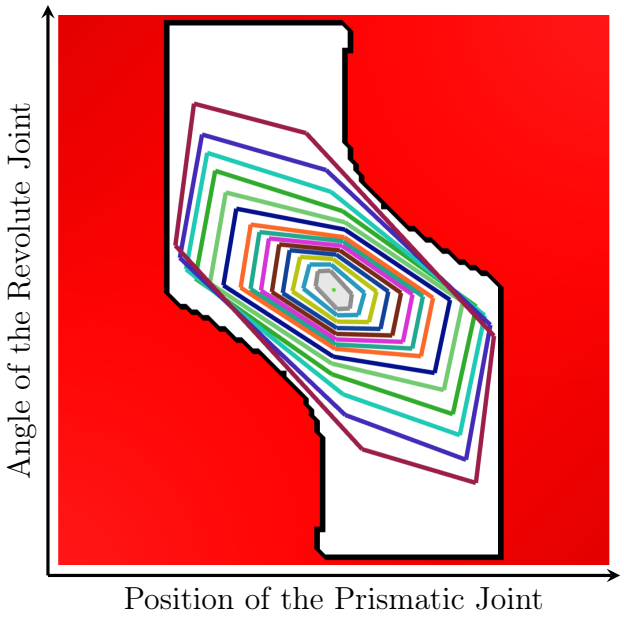
5.6.1 Simple Robots

In this section, we consider two simple robots each containing only two degrees of freedom. This enables us to visualize both the task space, as well as the configuration space. Though containing few degrees of freedom, each environment maintains rich, realistic collision geometries.

5.6.1.1 Pendulum on a Rail



(a) The pendulum on a rail robot. The green base can slide freely to the left and right, while the orange arm can rotate freely. Each hyperplane is a function of the TC-variable s and separates the collision body of the same color from the tip of the robot highlighted in black.



(b) The tangent configuration space of the pendulum on a rail robot. The tangent-configuration-space obstacle is in red. A sample of the polytopes obtained running Algorithm 3 around the configuration $(0, 0)$ are shown.

Figure 5.5: A 2-DOF robot consisting of a revolute joint at the base of the orange link and a prismatic joint between the base and the box.

Our first robot, shown in Figure 5.5a, consists of a single arm, shown in orange, connected to a base via a revolute joint and placed within a box. The base of the robot is connected to the box via a prismatic joint. The collision geometries of the robot and box are approximated using polytopic boxes. A total of 42 pairs of geometries can collide in this scene (i.e. certifying non-collision requires solving 42 instances of either (5.18) or (5.25)). In

Figure 5.5b, we visualize the two-dimensional tangent configuration space of our robot with the $\mathcal{T}^{\text{space}}$ obstacle shown in red. We emphasize the highly non-convex shape of $\mathcal{T}^{\text{free}}$.

We run Algorithm 3 starting with a regular octagon of side length 0.01 centered at the configuration $(0, 0)$, a configuration with the arm fully extended upwards and centered in the box. We obtain a sequence of certified polytopes of increasing size in the $\mathcal{T}^{\text{space}}$, which are plotted in varying colors in Figure 5.5b.

The algorithm terminates after 86 iterations of the while loop from Algorithm 3, taking a total of 314 seconds of wall clock time. During the course of the algorithm, the volume of the maximum inscribed ellipsoid improves by a factor of 83, from a starting value of 0.021 to 1.746. The improvement in the volume of the inscribed ellipsoid and the average time spent by the solver to solve both the certification program (5.18) and (5.33) are reported in Figure 5.6a and Figure 5.6b respectively. Additionally, we note the total amount of time (wall-clock) time required to grow the entire region. The majority of the time is spent transcribing the program in the symbolic form to the optimization solver and in this example accounts for almost half the time spent by the algorithm. Such overhead could be reduced using parametric programming and code generation [165].

After completion, we select a single random configuration within our final certified region. In Figure 5.5, we highlight the tip of the pendulum in black. Additionally, we color each collision body for which the tip can collide in a separate color and plot the separating plane certificate between the tip and the body in the same color.

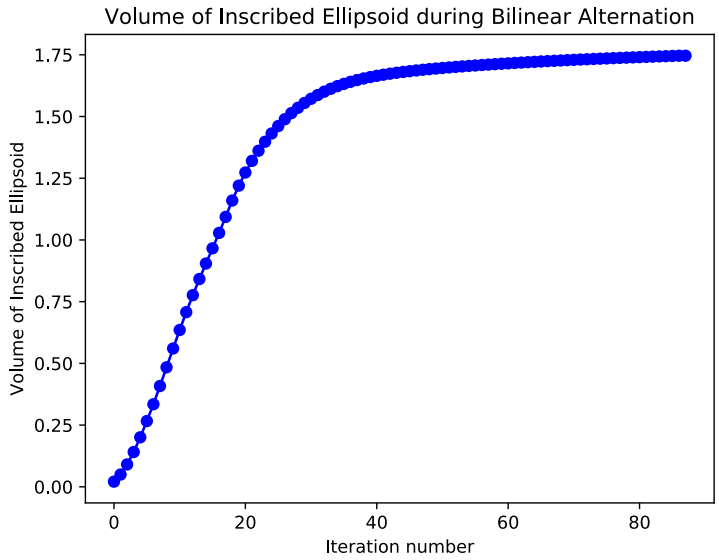
5.6.1.2 Pinball Flipper

We refer to our second system, shown in Figure 5.7a, as the pinball flipper. Each orange arm is connected to its gray base via a revolute joint. Each collision geometry in the scene is approximated with a box, and a total of 130 collision pairs exist. We similarly plot the $\mathcal{T}^{\text{space}}$ in Figure 5.7b with the $\mathcal{T}^{\text{space}}$ obstacle highlighted in red. In this experiment, we attempt to almost completely cover $\mathcal{T}^{\text{free}}$ with polytopic regions in order to enable a motion plan where the flippers exchange positions. Overall, this scene exhibits a much more complicated $\mathcal{T}^{\text{space}}$ obstacle as well as substantially more collision pairs when compared to the system from Section 5.6.1.1.

We seed Algorithm 3 with octagonal regions of side length 0.01, each centered at one of 5 different configurations shown as the black dots in Figure 5.7b. The resulting regions are also plotted in Figure 5.7b, and almost completely cover the space. Though each region was initially seeded with a polytope of the same shape, our algorithm successfully adapts the shape of each polytope to fill the space. Our algorithm also is not conservative; it successfully finds regions that are tight to the $\mathcal{T}^{\text{space}}$ obstacle in all cases.

The change in volume of the maximum inscribed ellipsoid of each region is shown in Figure 5.8a. We remark that the volume of each region exhibits a diverse set of behaviors over the iterations. Each region was grown sequentially, with a total wall clock time to cover the space of 1439s. This time could easily be improved by growing each region in parallel.

In Figure 5.9, we demonstrate the behavior of our certificates for various poses of our robot. In the top panel, we highlight in black the two tips of each flipper. The current configuration is highlighted as the green dot in the bottom panel. For each configuration, we also plot the hyperplane that proves the separation between the two black tips. Notice



(a) The volume of the maximum inscribed ellipsoids of the $\mathcal{T}^{\text{free}}$ regions shown in Figure 5.5b is plotted over iterations of Algorithm 3. This volume grows by a factor of 83 over the course of 86 iterations Algorithm 3.

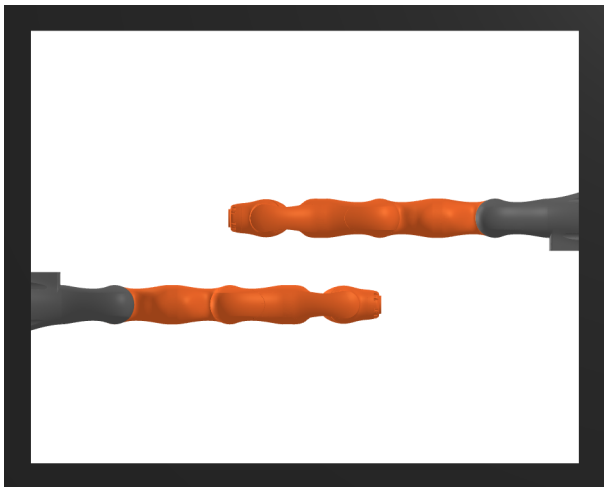
Number of collision pairs	42
Size of the largest PSD variable	2
Average time to solve (5.18)	0.191s
Average time to solve (5.33)	0.423s
Wall time to grow single region	314s

(b) Statistics dominating the run time of Algorithm 3 for the pendulum on a rail system. The complexity scales with the number of collision geometries as well as the size of the largest PSD matrix variable for enforcing the Psatz conditions in Programs (5.18) and (5.33).

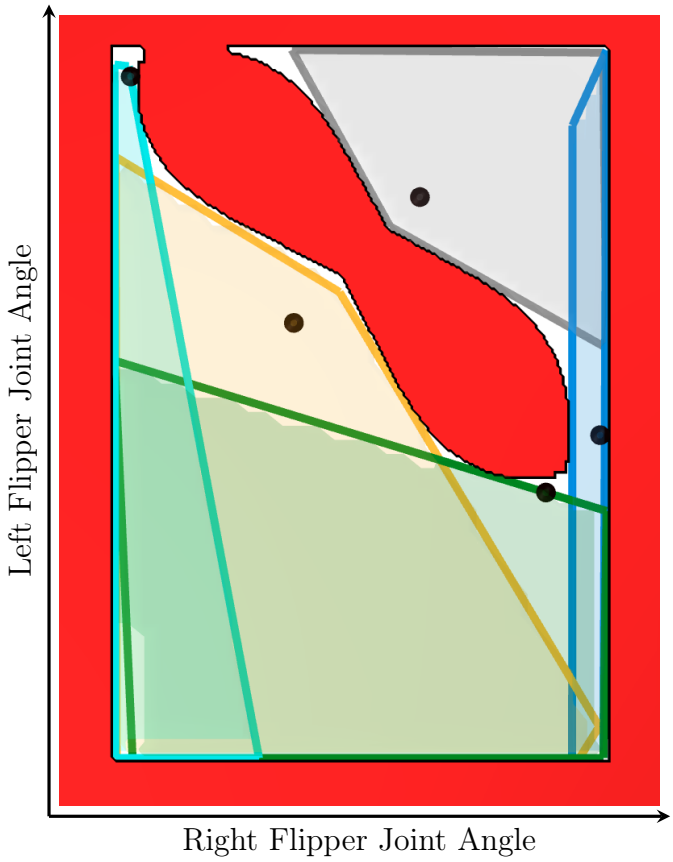
Figure 5.6: The progress of Algorithm 3 on the pendulum on a rail system for a single polytopic region is plotted. Statistics dominating the run time of the algorithm are also reported. The reported times for solving (5.18) and (5.33) are the amount of time spent within the optimization solver. The additional wall time to grow the region is due to transcription from the optimization parsing software into the solver.

that in Figures 5.9g, 5.9h, and 5.9i, the current configuration is contained in multiple regions at once. Therefore, each hyperplane in Figure 5.9a - 5.9e is drawn in the same color as its associated $\mathcal{T}^{\text{space}}$ region in Figures 5.9f - 5.9j.

We draw attention to the fact that at every configuration s_0 in $\mathcal{T}^{\text{free}}$, many different separating hyperplanes exist. The hyperplane obtained by evaluating the output of our certifier at s_0 is highly dependent on the region that is being certified. For example, in Figure 5.9h, the blue region corresponds largely to a change in the position of the left flipper, while the green region corresponds largely to a change in the right flipper. We see, in Figure 5.9c, that the algorithm finds different separating planes for the blue and the green region, even for the same configuration, so as to accommodate the different range of robot motion in each region. For the blue region, which includes a large rotation of the left flipper, the blue plane would continue to separate the left flipper from the right flipper as the left flipper moves. Similarly, the green plane would continue to separate the right flipper from the left as the right flipper moves.

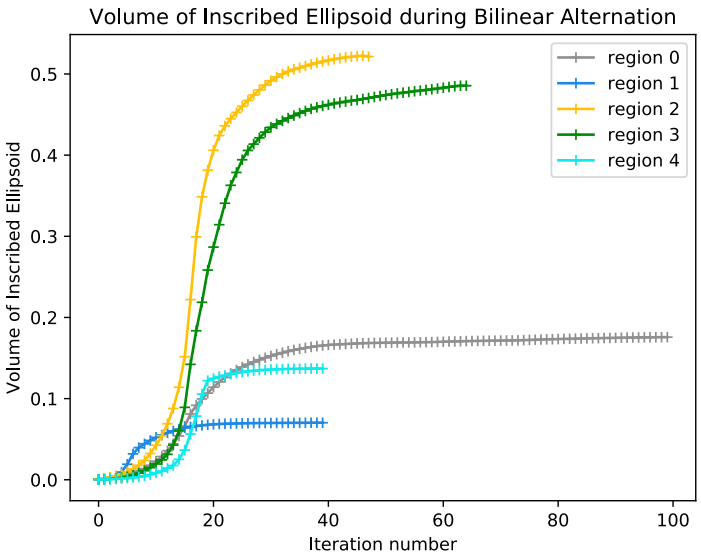


(a) The pinball flipper system consists of pendulums each with a revolute joint between the orange link and the gray base. All collision geometries in the scene are approximated using boxes.



(b) The $\mathcal{T}^{\text{space}}$ of the 2DOF pendulum flipper system. The $\mathcal{T}^{\text{space}}$ obstacle is shown in red. Algorithm 3 is run for five different polytopes each initially centered around the black dots. The polytopes output by the algorithm are plotted in various colors. These polytopes almost fully cover $\mathcal{T}^{\text{free}}$ and are guaranteed to be collision-free by construction.

Figure 5.7: The pinball flipper system and its $\mathcal{T}^{\text{space}}$. Algorithm 3 is able to successfully cover $\mathcal{T}^{\text{free}}$ with polytopic regions. An animation of the regions growing to cover this space is available here



(a) The volume of the maximum inscribed ellipsoid as the polytope is grown around various seedpoints is improved during Algorithm 3. The final polytopes associated to each color are shown in Figure 5.7b.

Number of collision pairs	130
Size of the largest PSD variable	2
Average time to solve (5.18)	0.638s
Average time to solve (5.33)	1.319s
Wall time to grow cover	1439s

(b) Statistics dominating the run time of Algorithm 3 for the pinball flipper system. The complexity scales with the number of collision geometries as well as the size of the largest PSD matrix variable for enforcing the Psatz conditions in Programs (5.18) and (5.33).

Figure 5.8: The progress of Algorithm 3 on the pinball flipper system for each polytopic region is plotted. Statistics dominating the run time of the algorithm are also reported. The reported times for solving (5.18) and (5.33) are the amount of time spent within the optimization solver. The additional wall time to grow the region is due to transcription from the optimization parsing software into the solver.

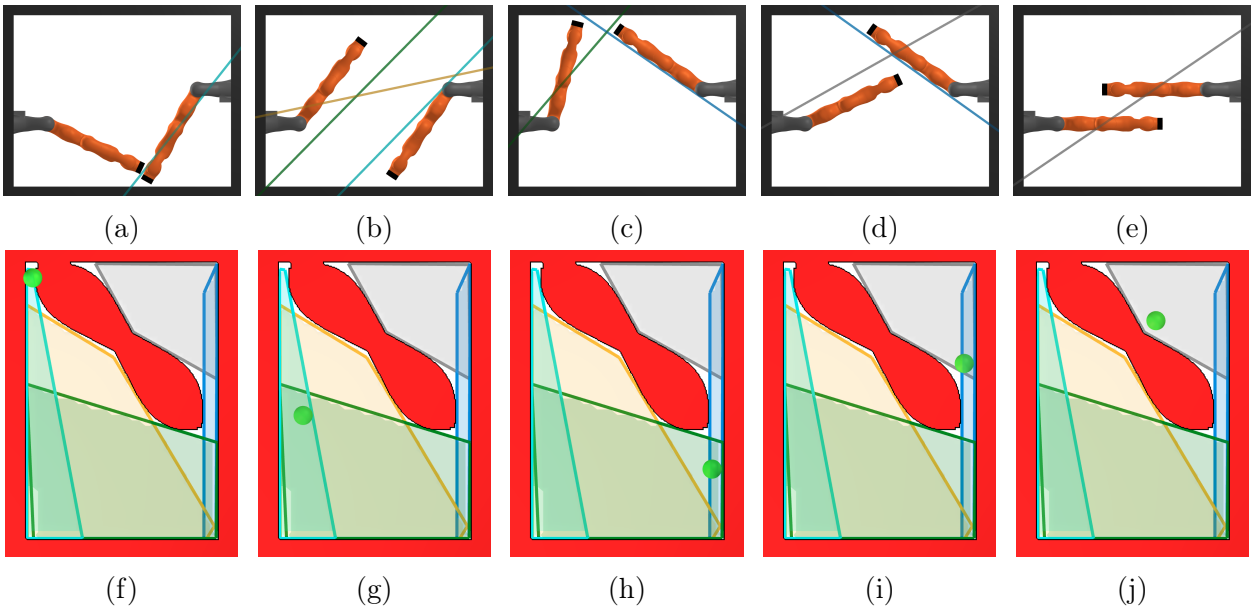


Figure 5.9: We approximate almost the entirety of $\mathcal{T}^{\text{free}}$ for the robot flipper system using 5 polytopic regions. The top panel shows the hyperplanes certifying that the two black tips of the system do not collide. The bottom panel shows the configuration of the robot as a green dot. An example of this system undergoing a trajectory is available here.

5.6.2 KUKA IIWA robot

In this section, we demonstrate our algorithm deployed on the KUKA iiwa arm in two scenes relevant to robot manipulation. The collision geometry of the iiwa is approximated as a union of convex polytopes, as are all obstacles in the scene. We begin by considering a single iiwa to demonstrate the practicality of our algorithm before considering a bimanual manipulator to demonstrate the scalability of our approach.

5.6.2.1 7-DOF IIWA With a Shelf

We apply Algorithm 3 to the scene shown in Figure 5.10: a 7-DOF KUKA iiwa arm reaching into a shelf. Our approach successfully finds many collision-free configurations, and we plot

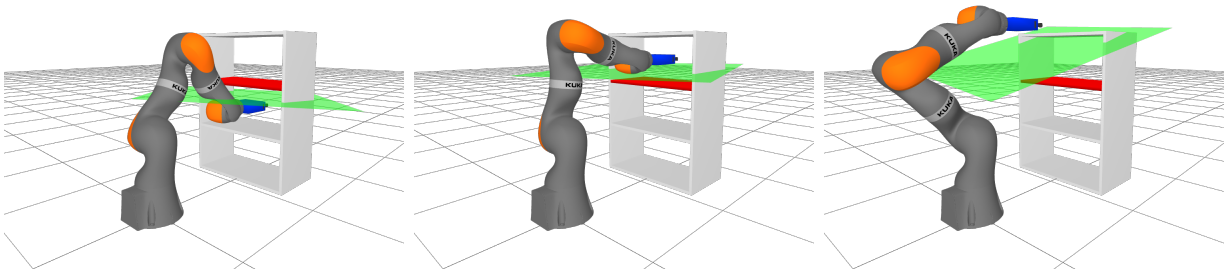
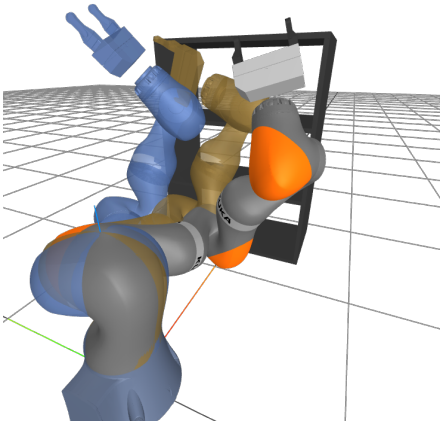
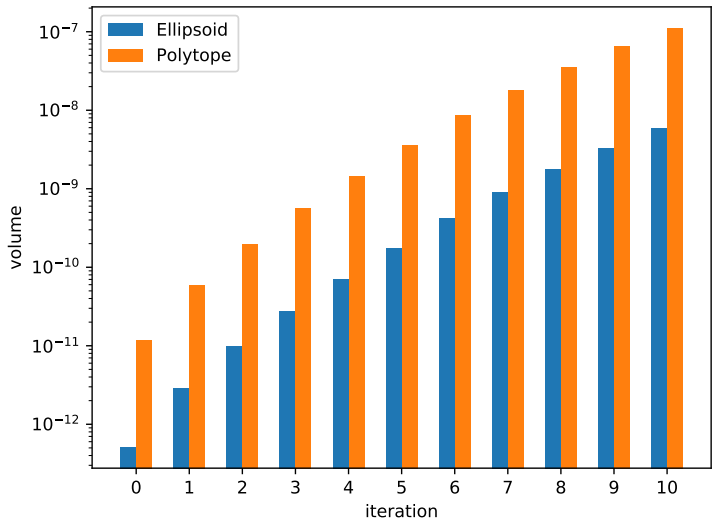


Figure 5.10: 7-DOF iiwa example. We highlight one pair of collision geometries (blue on robot gripper and red on the shelf), together with their separating plane (green).



(a) The configurations in our certified regions correspond to a wide range of task-space positions. We sample three configurations from the same certified region and plot the corresponding task-space position in different colors.



(b) A single region for the 7-DOF KUKA iiwa is grown over the course of 11 iterations of Algorithm 3. We compare the volume of the maximum volume inscribed ellipsoid to the volume of the polytopic region at each iteration and show that the volume improves by a factor of 10,000.

Figure 5.11: Algorithm 3 grows certified regions that contain configurations reaching a large portion of the task space. We show that our algorithm is capable of growing the volume of a certified region by a factor of 10,000 over the course of just 11 iterations.

in green the separating hyperplane certificate between the end-effector, highlighted in blue, and the top shelf highlighted in red.

The run time of Algorithm 3 is dominated by the certification of non-collision between the pairs with the longest kinematic chain, as this leads to the highest degree polynomials and hence the semidefinite variables in programs (5.18) and (5.33). For this program, the largest positive semidefinite matrix variable has 16 rows. Overall, the largest certification program (5.18) takes 54s to solve, while the program (5.33) takes on average 8s to solve.

In Figure 5.11, we demonstrate the behavior of one certified region. In Figure 5.11a, we show that the configurations in one of our certified polytopic regions of $\mathcal{T}^{\text{space}}$ (with 24 faces in the polytope) correspond to many task-space end-effector positions. The configurations from Figure 5.11a are drawn from a region that grows by a factor of $10,000$ using 11 iterations of Algorithm 3. This improvement in volume is reported in Figure 5.11b, where we also compare the volume of the maximum volume inscribed ellipsoid against the volume of the polytopic region.

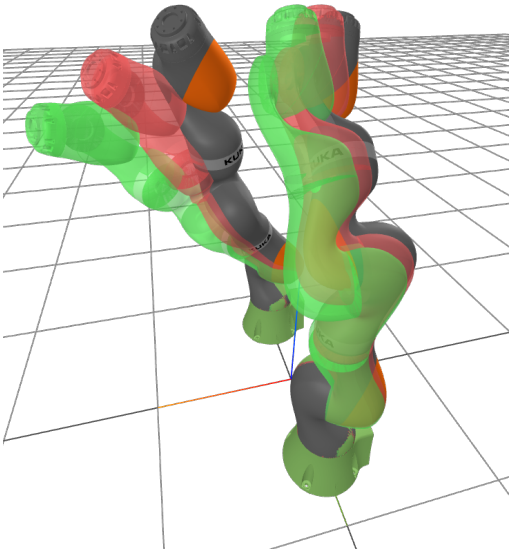
5.6.2.2 12-DOF Bimanual KUKA IIWA Example

We next consider designing regions to avoid self-collision for a robot consisting of two KUKA iiwa arms with the final joint welded (rotation of the final joint does not change the con-

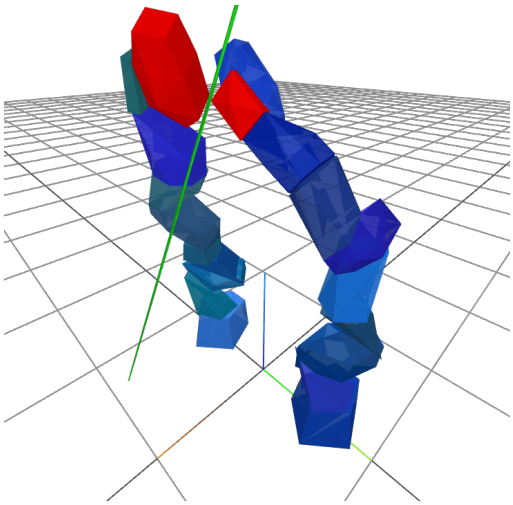
figuration of any geometry for this robot). This robot has 12 DOF. This system tests the scalability of our algorithm due to the degree of the polynomials involved in the forward kinematics, as well as the complexity of the collision geometries.

Solving the largest certification program in (5.18) takes 105 minutes, while the program in (5.33) takes 4 minutes. The increase in solve times compared to the single iiwa environment from Section 5.6.2.1 is best attributed to the increase in the size of the semidefinite variables due to the larger number of DOF. The largest semidefinite matrix in both programs has 64 rows and corresponds to certifying that the two tips of the iiwas do not collide.

Nonetheless, our algorithm again finds certified, 30-face polytopic regions of $\mathcal{T}^{\text{space}}$, which correspond to a wide range of task-space positions as seen in Figure 5.12a. Moreover, the same region is quite tight to the $\mathcal{T}^{\text{space}}$ obstacle; one sampled configuration in the certified region, shown in Figure 5.12b, corresponds to just 7.3mm of separation between the two arms.



(a) Multiple configurations of the 12-DOF, bimanual iiwa manipulator sampled from a single certified region of $\mathcal{T}^{\text{free}}$. Each configuration is shown in a separate color.



(b) The geometries of the bimanual iiwa from Figure 5.12a are tightly approximated using polytopes. At one position in the certified $\mathcal{T}^{\text{free}}$ region, the two geometries highlighted in red are separated by just 7.3mm.

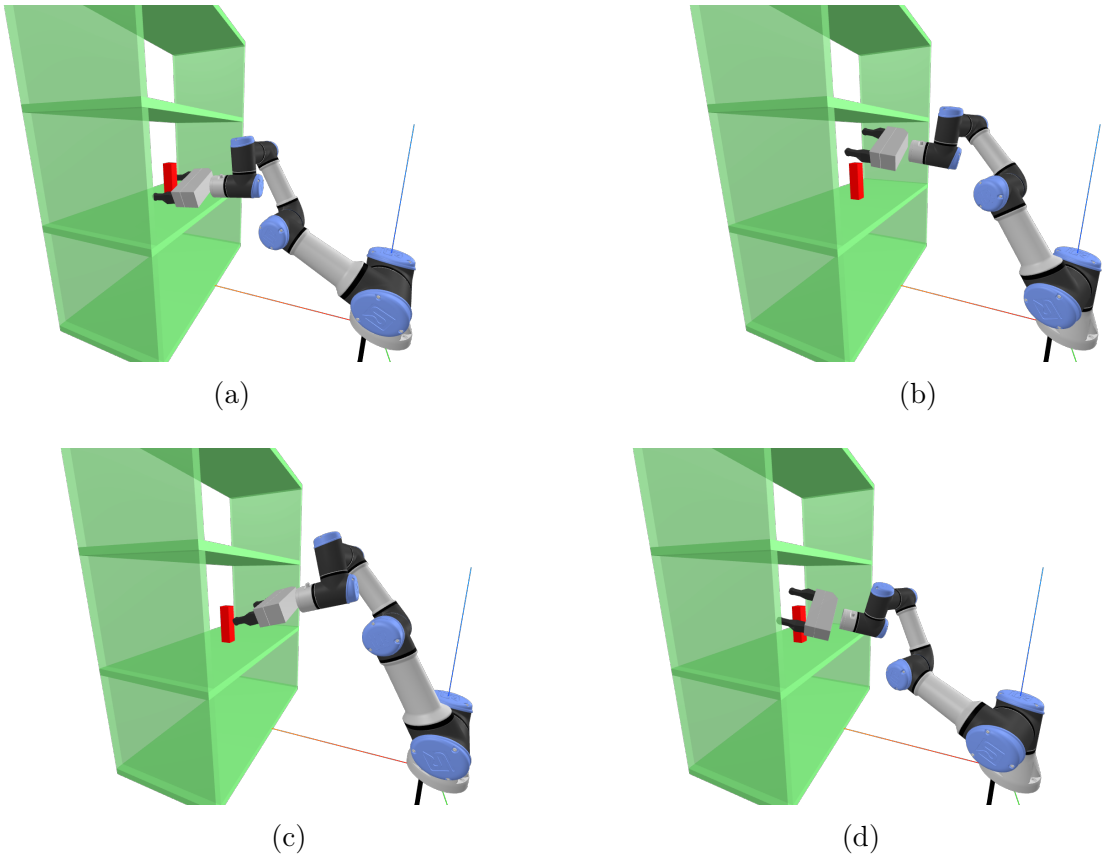
Figure 5.12: Algorithm 3 finds certified polytopic regions of $\mathcal{T}^{\text{free}}$ even for high DOF systems in reasonable times. The algorithm is also not conservative. It finds large regions that correspond to a broad range of task-space positions. Moreover, the regions are very tight to the $\mathcal{T}^{\text{space}}$ obstacle, finding configurations that lead to very small separation between the task-space objects.

5.6.3 UR3e Robot

In this section, we test our algorithm on a UR3e robot with a gripper mounted at the wrist. The robot's links are approximated by cylinders and we weld the gripper's prismatic joints so

that each UR3e has a total of 6 DOFs. This section differs from the KUKA iiwa experiment in Section 5.6.2 due to the introduction of non-polytopic collision geometries into the scene. Similar to Section 5.6.2, we test our approach for a scene where the robot is reaching into a shelf, as well as a bimanual set up.

5.6.3.1 6-DOF UR3e With a Shelf



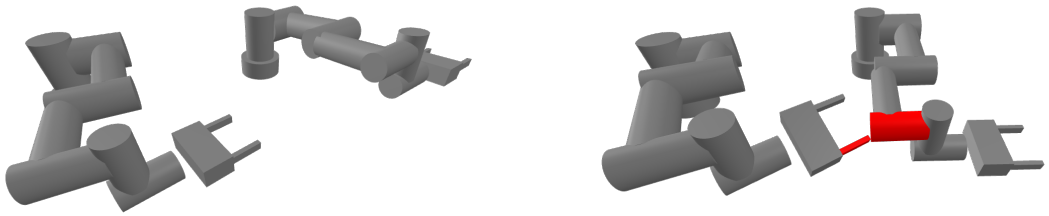
(a) (b) (c) (d)

Figure 5.13: Different postures sampled within one certified $\mathcal{T}^{\text{space}}$ region for a UR3e robot with gripper. The certified region includes the gripper reaching the red box in the center of the shelf (Figure 5.13a), retracting from the shelf (Figure 5.13c), and reaching different regions within the shelf while avoiding the red box (Figure 5.13b and 5.13d). An animation of the range of configurations attainable in this region is available on the project website.

In Figure 5.13, we consider a UR3e robot reaching into a shelf to grasp a small box-shaped object. To simulate a situation where the robot is attempting to pick up the red object, we use Algorithm 3 to grow a certified, $\mathcal{T}^{\text{free}}$ polytope (with 12 faces) near the object. Figure 5.13 shows a variety of postures sampled from the final $\mathcal{T}^{\text{free}}$ polytope and demonstrates that within a single region, our robot is able to reach into the shelf to grasp the object, retract away from the shelf, and maneuver within the shelf while avoiding the object.

Similar to Section 5.6.2.1, the largest semidefinite variables in programs (5.18) and (5.33) have 16 rows with program (5.18) taking about 56s to solve.

5.6.3.2 12-DOF Bimanual UR3e



(a) (b)

Figure 5.14: Top down view of two postures sampled within one certified $\mathcal{T}^{\text{space}}$ region on the dual UR3e platform. In the right figure we highlight the two collision geometries that are separated by only 0.3mm. A dynamic visualization of the range of attainable postures is available by running this notebook

Finally, we demonstrate our algorithm on a dual UR3e platform shown in Figure 5.14. Again, we emphasize that we are able to find large regions of $\mathcal{T}^{\text{free}}$ which correspond to diverse positions in task space with the postures in Figure 5.14a and 5.14b being drawn from the same certified region (a 13-face polytope). Moreover, these regions are very tight to the $\mathcal{T}^{\text{space}}$ obstacle with the two bodies highlighted in red in Figure 5.14b being just 0.3mm apart. For this example, the largest positive semidefinite matrices in (5.18) and (5.33) have 128 rows with the largest program taking about 35 minutes to solve. This program solves faster than the analogous program for the bimanual iiwa from Section 5.6.2.2 because we require fewer polynomial positivity conditions to certify that the UR3e's cylindrical geometries are on a given side of a plane compared to the polytopic approximation used for the iiwa.

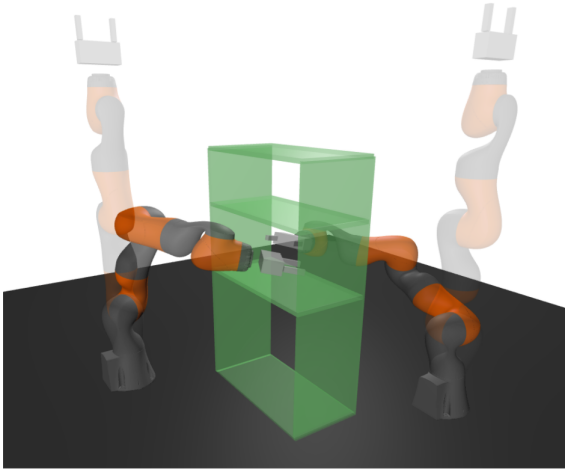
5.6.4 Fast Certification of Trajectories

We now show that the certification of a trajectory can be done substantially faster than the certification of a full dimensional region. All computations in this section are performed on a laptop computer with an 11th Generation Intel I9 CPU and 64GB of RAM. The results of this section are reproduced from [117].

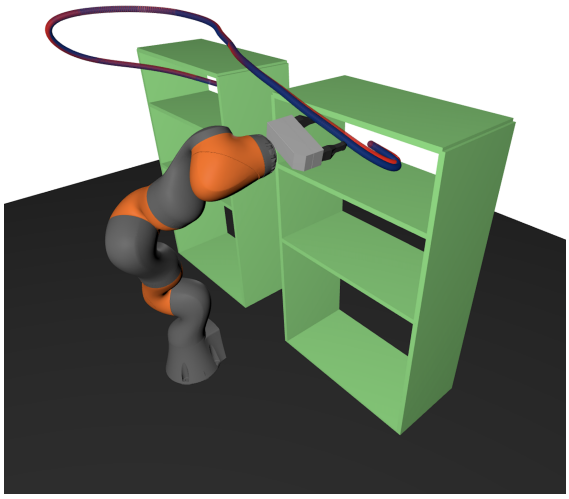
In the case of certifying trajectories, a common practice is to rapidly cull some of the collision pairs which cannot collide over the course of the motion by using, for example, bounding-volume hierarchies [127,166]. In our experiments, we do not perform this optimization which would certainly improve our run times.

5.6.4.1 Certifying an RRT for a Bimanual Manipulator

We consider the task of certifying a motion plan for a 12 degree of freedom (DOF) bimanual KUKA iiwa reaching into the shelf. This is the same robot as in Section 5.6.2.2, placed



(a) A pair of Kuka iiwa reaching from the straight up start configuration (translucent arms) into the shelf (opaque arms). The close confines of this motion plan make checking safety via finite sampling challenging. An animation of the motion plan is available here.



(b) A 7-DOF Kuka iiwa reaching into a pair of shelves. Despite differing by at most 20mm, the blue motion plan is collision-free, while the red motion plan contains minor collision when reaching into each shelf. The proposed certification method is capable of discriminating the safety of these two motion plans. A video describing the method is available at https://youtu.be/oTiDYeptKis.

Figure 5.15: The trajectory certification environments.

in an environment with more obstacles. The start and end configurations are shown in Figure 5.15a. The environment contains 246 total collision pairs.

An RRT is grown in $\mathcal{T}^{\text{space}}$, with edges extended by sampling one hundred intermediate points between tree nodes. We grow the RRT until the task and goal are connected and the motion plan between them is certified as safe by (5.18). This requires 105 edges and so a total of 25,830 instances of (5.18) are solved with the hyperplane parameterized by a linear polynomial. All programs are solved in parallel.

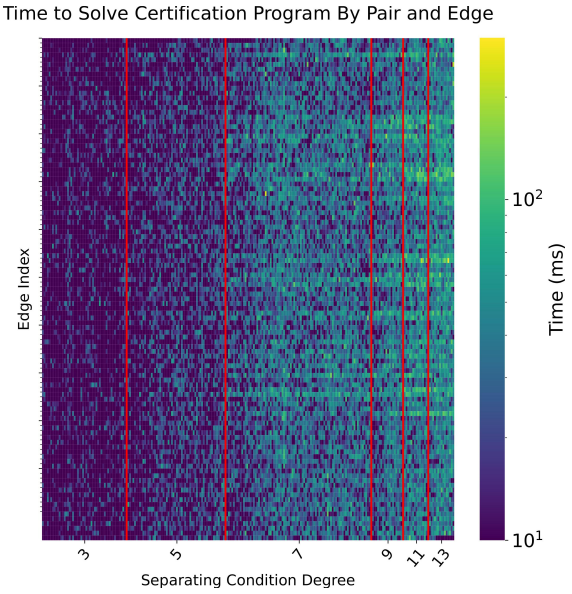
In Figure 5.16a, we plot the entire distribution of solve times for the 25,830 programs. We group the collision geometry pairs by the highest degree condition given by (5.18). The most expensive program takes 297 ms to solve with Mosek [85], while the least expensive program takes 0.66 ms to solve. We note that Figure 5.16a is plotted on a log scale and demonstrates approximately quadratic growth as the degree of the polynomials increases to the right. On the other hand, almost no pattern is observed in the length of time taken to certify an individual edge for a given pair.

We remark on the substantial improvement in the runtimes for this system relative to Section 5.6.2.2 which uses the same robot in a *simpler* environment containing fewer collision pairs. In that section, the most expensive certification program takes 105 minutes, compared to the 297 ms required in this section. This is more than a 21000× speed up.

In Figure 5.16b, we provide some aggregated statistics on the certification procedure.

We recall that an edge of the RRT is considered safe if (5.18) is feasible for *all* 246 collision pairs. Due to the very close proximity of the collision geometries and our very coarse collision checking, we do not expect every edge to be collision-free. Indeed, we see that just under 30% of the edges in our tree are certified as collision free.

We observe that many of the uncertified edges correspond to motions in the tight confines of the shelf and so may in fact contain collisions. Therefore, if (5.18) for a given edge is not feasible for every collision pair, then we resample 10^5, uniformly spaced configurations along that edge. If a collision is found, we confirm that the edge is not safe. Otherwise, we cannot confirm or deny the safety of the edge. We see that in 96% of cases, more dense sampling recovers a collision, which verifies that the infeasibility of (5.18) for these edges is not due to choosing too low a degree for the hyperplane. On the other hand, exactly two edges are neither declared safe, nor confirmed not safe. These two examples correspond to motions which undergo very large task space displacement of all links, but do not appear to contain collisions. It is likely the case that a higher degree polynomial could certify these edges.



(a) The time required to certify each edge and each collision pair in an RRT for the bimanual iiwa system. The y-axis is the index of the edge, and the x-axis is the index of the collision pair, grouped by the degree of the separating hyperplane condition (5.18). The time taken to solve the certification program for a collision pair scales quadratically in the degree of the separating condition (5.18).

Aggregate Statistics For Certifying the Entire RRT

# Instances of (5.18) Solved	25830
# Of Collision Pairs	246
# Of Edges	105
# Edges safe	31
# Edges Confirmed not safe	72
# Edges Unconfirmed not safe	2
Average Solver Time to Certify an Edge	299.9 msec.
Total Solver Time to Certify Goal Plan	1.211 sec.
Total Solver Time to Certify RRT	31.5 sec.

(b) Aggregated statistics for certifying an RRT for the bimanual iiwa system. An edge is safe if all 246 instances of (5.18) are feasible. If (5.18) is infeasible, the corresponding edge is densely sampled with 10^5 points to find a collision. If a collision is found, the edge is confirmed not safe. We see that in two cases, neither (5.18) is feasible for all pairs, nor is a collision found. The final RRT plan is certified as safe in just over 1 second, and the whole tree is certified in about 30 seconds.

Figure 5.16: Timing statistics and aggregated statistics for certifying an RRT for the bimanual iiwa system from Figure 5.15a. An instance of (5.18) is solved for all 246 collision pairs and all 105 edges in just over 30 seconds.

5.6.4.2 Certifying Cubic Polynomial Plans for a 7-DOF iiwa

We demonstrate our method's ability to certify higher-order motion plans. We consider a 7-DOF Kuka iiwa interacting with a pair of shelves shown in Figure 5.15b moving along a piecewise-polynomial plan with thirty pieces. Each piece is parametrized as a cubic Hermite spline.

We consider two such motion plans. The blue plan in Figure 5.15b is collision free, while exactly two of the thirty pieces of the red plan contain minor collisions with the shelf. We solve (5.18) with a linearly parametrized hyperplane for each segment of the plan.

In the case of the blue curve, all thirty pieces of the motion plan are certified as safe in 8.99 seconds. Additionally, the twenty-eight pieces of the red plan which are safe are marked as safe. Meanwhile, the optimizer reported that (5.18) is infeasible for the two unsafe segments. The total time to solve the programs for the unsafe plan was 9.21 seconds, which can be reduced to 6.91 seconds if the two unsafe segments are allowed to terminate as soon as one collision pair fails to certify its safety.

This demonstrates the precision of our method, which is capable of discriminating the safety of two visually indistinguishable motion plans which differ by at most 20mm.

The trajectory-certification examples highlight one use of the certificates developed in this chapter: rapidly verifying a fixed one-dimensional subset of configuration space. We next return to the full-dimensional problem.

Body	$a^T x + b > 0, \forall x \in \mathcal{A}$	$x \in \mathcal{A}$
V-rep Polytope with vertices $\{v_1, \ldots, v_m\}$.	$a^T v_i + b \geq 1, \ \forall\, i \in \{1, \ldots, m\}$	$x = \sum_{i=1}^{m} \mu_i v_i, \ \sum_{i=1} \mu_i = 1,$ $\mu_i \geq 0$
Sphere with center o and radius r.	$a^T o + b \geq r \lVert a \rVert$ $a^T o + b \geq 1$	$\lVert x - o \rVert^2 \leq r^2$
Capsule, the convex hull of two spheres with centers o_1 and o_2 and radii r_1 and r_2.	$a^T o_1 + b \geq r_1 \lVert a \rVert$ $a^T o_2 + b \geq r_2 \lVert a \rVert$ $a^T o_1 + b \geq 1$	$o_\mu = \mu o_1 + (1-\mu) o_2$ $\lVert x - o_\mu \rVert \leq \mu r_1 + (1-\mu) r_2$ $0 \leq \mu \leq 1$
Cylinder, the convex hull of two circles with centers o_1 and o_2, and with radii r_1 and r_2.	$\frac{a_z \lVert o_1 - o_2 \rVert}{2} + b \geq r_1 \left\lVert \begin{bmatrix} a_x & a_y \end{bmatrix} \right\rVert$ $\frac{-a_z \lVert o_1 - o_2 \rVert}{2} + b \geq r_2 \left\lVert \begin{bmatrix} a_x & a_y \end{bmatrix} \right\rVert$ $a^T \left(\frac{o_1 + o_2}{2} \right) + b \geq 1$	$o_\mu = \mu o_1 + (1-\mu) o_2$ $v^T (o_1 - o_2) = 0$ $x = o_\mu + v$ $\lVert v \rVert \leq \mu r_1 + (1-\mu) r_2$ $0 \leq \mu \leq 1$

Table 5.1: Parameterizations of conditions (5.3a) and (5.4a) respectively for particular convex bodies. [4]

5.7 Covering Free Space in Convex Regions

The region-growing method in Section 5.5 shows how to certify and improve a single collision-free region around a seed, but it does not answer where such seeds should be placed if the goal is to cover a useful portion of $\mathcal{C}^{\text{free}}$. The next section addresses this placement problem by using visibility graphs and clique covers to propose seeds for region inflation. Earlier in this chapter, we introduced a new algorithm in the IRIS family for growing a provably collision-free region in (a reparametrization) of configuration space. Many other algorithms in this family have been proposed [145,147,164]. In this section, we turn our attention from growing a single region to covering the region in these convex sets. We propose an efficient method for the approximate decomposition of robot configuration spaces into a few large convex sets, without any human supervision. A guiding illustration is shown in Figure 5.17. The results of this section are based on [118]. In this section, we work directly in $\mathcal{C}^{\text{free}}$, however the method is agnostic to the space, and the method could used in $\mathcal{T}^{\text{free}}$.

Similar to some motion-planning algorithms [167,168], our method constructs a visibility graph by sampling points in $\mathcal{C}^{\text{free}}$. The vertices of this graph are collision-free samples, and the edges connect pairs of points with mutual line of sight. The visibility graph contains rich information about the geometry of $\mathcal{C}^{\text{free}}$. In particular, our key observation is that, as the number of samples grows, cliques (see Definition 37) subgraphs of this visibility graph tend to represent better and better approximations of collision-free convex sets in the underlying configuration space. Our approach is to decompose the visibility graph into a small collection of large cliques. We then circumscribe the points in each clique with an ellipsoid by solving a convex-optimization problem. The center and the principal directions of these ellipsoids are subsequently used to initialize an inflation algorithm analogous to IRIS.

Through a large variety of experiments, we show that our algorithm outperforms previous approaches for seeding and inflating convex regions; both in terms of runtimes and number of regions used to cover the space. As an example, for a robot arm with seven degrees of freedom, and many task-space obstacles, our method requires approximately 46 regions and one hour of computations to cover 70% of $\mathcal{C}^{\text{free}}$, whereas the approach outlined in [164] requires ten times more regions and is ten times slower. The time to compute regions in this section is substantially smaller than in prior section, as we use optimistic approaches to growing regions such as IRISNP2 and IRISNP from [146,147,164] rather than the rigorous method C-IRIS introduced in the prior section.

5.7.1 Related Works

Finding the minimum convex cover of a set is a hard problem; even in the case of two-dimensional polygons, the problem is hard to solve exactly and approximately.[13] Nonetheless, finding low-cardinality convex covers of high-dimensional nonconvex spaces (both polygonal and non-polygonal) remains a problem of practical importance, for which a variety of approximate algorithms have been devised. In the following, we group these algorithms into two categories: ones that require explicit (e.g., analytic) descriptions of $\mathcal{C}^{\text{free}}$, and ones that only use implicit descriptions of $\mathcal{C}^{\text{free}}$ (and are hence suitable for most complex configuration

[13] More formally, the problem is $\exists\mathbb{R}$-complete [140], and therefore NP-hard, as well as APX-hard [142].

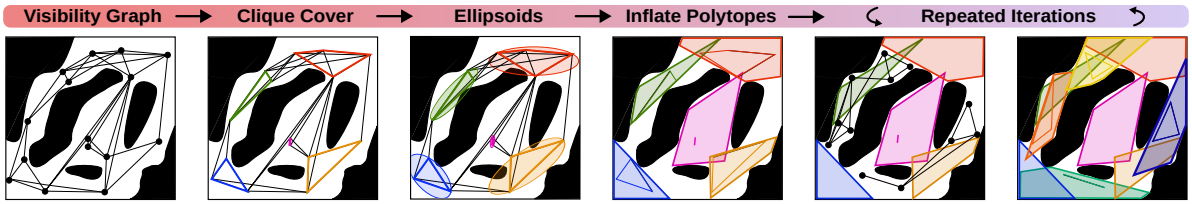


Figure 5.17: Sketch of the proposed algorithm on a simple example. *First four figures*: Samples are drawn uniformly from $\mathcal{C}^{\text{free}}$ to build a visibility graph. The visibility graph is decomposed into five cliques. The principal directions and locations of the cliques are used to direct a region-inflation algorithm. *Remaining two figures*: This process is repeated until sufficient coverage is obtained by drawing new samples from the remaining free space, and repeating the previous steps.

spaces). In this literature review, we particularly focus on IRIS algorithms, due to their efficient scaling to high dimensions.

5.7.1.1 Algorithms Requiring Explicit Obstacle Descriptions

The recent work [169] constructs low-cardinality convex covers of two-dimensional polygons by first decomposing them into small convex pieces (for example triangles), and then by joining the pieces based on a small clique cover of an approximated set-to-set visibility graph. Similarly, the visibility graph of polygons is directly used to construct convex covers in [170]. In three dimensions the approach in [149] can be used to decompose $\mathcal{C}^{\text{free}}$ into sets that are approximately convex, provided that a triangle mesh description is available. Finally, if the configuration-space obstacles are explicitly described as convex sets, the original IRIS algorithm from [145] can be used to inflate a large polytope in $\mathcal{C}^{\text{free}}$ around a specified seed point in arbitrary dimensions.

5.7.1.2 Algorithms Allowing Implicit Obstacle Descriptions

Most commonly, the explicit descriptions of the obstacles are only available in task space; while the collision-free configuration space $\mathcal{C}^{\text{free}}$ is defined implicitly through the robot's inverse kinematics, and is intractable to describe analytically [122, §3]. Some works have focused on approximately describing the configuration space obstacles, rather than $\mathcal{C}^{\text{free}}$ [120,121,171]. Conversely, multiple works have studied directly obtaining convex decompositions of $\mathcal{C}^{\text{free}}$. In [172] visibility graphs and kernels are used to compute convex decompositions of three-dimensional spaces via sample-based collision checking. However, their method represents the convex sets through their vertices, and is inefficient in higher dimensions. In the family of IRIS algorithms, three methods can deal with implicit descriptions of $\mathcal{C}^{\text{free}}$: IRISNP [164], IRISNP2 [146,147], and C-IRIS as described earlier in this chapter and in [115,116]. The first two extend the original IRIS method [145] to arbitrary configuration spaces using nonlinear programming and inflates polytopes that are collision-free with high probability. The latter, as described in this chapter, grows polytopes that are rigorously *certified* to be collision-free using a rational reparametrization of the configuration space and sums-of-squares programming.

5.7.2 Convex Covers, Visibility, and Cliques

In this section, we formally define our main problem: approximating the free configuration space $\mathcal{C}^{\text{free}}$ with a low-cardinality collection of convex sets. We also briefly review the main technical tools that we will use in the development of our algorithm, namely, visibility graphs and clique covers.

5.7.2.1 Problem Statement

Let $\mathcal{C}^{\text{free}} \subseteq \mathbb{R}^n$ be the collision-free subset of an n-dimensional configuration space, which we assume to have a well-defined, finite volume. Let α be a constant in the interval $(0, 1]$.

Definition 41. *An* α-approximate convex cover *of* $\mathcal{C}^{\text{free}}$ *is a collection* $\mathfrak{R} = \{\mathcal{R}_1, \ldots, \mathcal{R}_N\}$ *of potentially overlapping convex sets* $\mathcal{R}_i \subseteq \mathcal{C}^{\text{free}}$ *whose union covers at least an* α*-fraction of the volume of* $\mathcal{C}^{\text{free}}$*:*

$$\mathbf{vol}\left(\bigcup_{i=1}^{N} \mathcal{R}_i\right) \geq \alpha\, \mathbf{vol}\left(\mathcal{C}^{\text{free}}\right).$$

Our problem is to find an α-approximate convex cover of minimum cardinality N.

Problem 5.1: MIN α-APPROXCONVEXCOVER

$$\begin{aligned} &\text{minimize } N \text{ subject to} \\ &\mathbf{vol}\left(\bigcup_{i=1}^{N} \mathcal{R}_i\right) \geq \alpha\, \mathbf{vol}\left(\mathcal{C}^{\text{free}}\right), \\ &\mathcal{R}_i \subseteq \mathcal{C}^{\text{free}}, \quad \forall i = 1, \ldots, N. \end{aligned}$$

In practice, we are interested in solving this problem for values of α that are sufficiently high to accomplish a task of interest, such as collision-free motion planning, but not so large that the cardinality N grows unreasonably. Indeed, when $\alpha = 1$, MIN α-APPROXCONVEXCOVER might not even have a finite solution.[14]

5.7.2.2 Visibility Graphs and Clique Covers

Our algorithm is based on the idea that clusters of points that see each other can approximate convex subsets of $\mathcal{C}^{\text{free}}$. Here, we formally define the notion of visibility as well as introduce some formal tools from graph theory to guide the development of our algorithm.

We begin by defining visibility in $\mathcal{C}^{\text{free}}$.

Definition 42. *Two points* $q, q' \in \mathcal{C}^{\text{free}}$ *are said to* see each other *if the entire line connecting them is collision-free:* $tq + (1-t)q' \in \mathcal{C}^{\text{free}}$ *for all* $t \in [0, 1]$*. Notice that this definition is symmetric in* q *and* q'*.*

[14] For $\alpha < 1$ a finite solution is guaranteed to exist. This can be seen by computing the volume of $\mathcal{C}^{\text{free}}$ with a lower Darboux integral over finite hyperrectangle partitions.

We are now ready to define the visibility graph of a set of collision-free configurations.

Definition 43. *The* visibility graph *of a set of points $q_1, \ldots, q_K \in \mathcal{C}^{\text{free}}$ is an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with vertices $\mathcal{V} = \{1, \ldots, K\}$, and with edges $\{i, j\} \in \mathcal{E}$ for every pair of distinct points q_i and q_j that see each other.*

We show an example of a visibility graph in Figure 5.17. We note that clusters of points that can all see each other form a clique in the visibility graph. We observe that if a cluster of configurations can be placed in the same convex set, then these configurations must form a clique in the visibility graph. The second panel in Figure 5.17 highlights a collection of five cliques in the visibility graph that have this property. These five cliques form what is called a clique cover, which resembles a discrete analog of a convex cover.

Definition 44. *A collection $\mathcal{T}$ of cliques $\mathcal{K}_1, \ldots, \mathcal{K}_N$ is a* clique cover *of a graph $\mathcal{G}$ if every vertex in the graph is contained in at least one clique.*

A natural discrete counterpart of the minimum convex cover is the **MinCliqueCover** problem, where we look for the minimum number of cliques N required to cover a graph. Our observation is that, as the number of samples in the visibility graph increases, a minimum clique cover typically does an increasingly better job of approximating a minimum convex cover. Limitations of this analogy are discussed in Section 5.7.5.

MinCliqueCover is NP-complete [18]. There exist heuristics, such as [173], that attempt to solve **MinCliqueCover** directly. Alternatively, one can greedily construct a clique cover by repeatedly eliminating the largest clique. The problem of finding the largest clique in a graph is called **MaxClique**. Even though this problem is also NP-complete [18], it is often substantially faster to solve in practice. We found the latter approach with exact solutions of **MaxClique** to perform particularly well on our problem instances.

5.7.3 Algorithm

We now present our Visibility Clique Cover (VCC) algorithm, which consists of four main steps. First, we randomly sample a collection of points in $\mathcal{C}^{\text{free}}$ and construct their visibility graph. Second, we compute an approximate clique cover of the graph. Third, we summarize the geometric information of each clique using an ellipsoid. Fourth, we use these ellipsoids to initialize a polytope-inflation algorithm analogous to Iris. This process is repeated until the generated set of polytopes $\mathfrak{R}$ covers a given fraction α of $\mathcal{C}^{\text{free}}$. This procedure is summarized in Algorithm 4, and the remainder of this section details the individual steps.

5.7.3.1 Sampling the Visibility Graph

At the beginning of every iteration of VCC, the subroutine SampleVisibilityGraph samples K configurations uniformly at random from the portion of $\mathcal{C}^{\text{free}}$ that is not already covered by the polytopes in $\mathfrak{R}$. Then, it constructs the visibility graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, by checking for collisions along the line segments connecting each pair of sampled configurations. Currently, this is performed using sampling-based collision checkers. Exact visibility checking is possible using methods such as [133], [174, §5.3.4], or using the method from Section 5.4.4 [117].

Algorithm 4: VisibilityCliqueCover

Input :
α: coverage threshold
K: number of samples per iteration
$s_{\min}$: minimum clique size
Output:
$\mathfrak{R}$: set of convex polytopes approximating $\mathcal{C}^{\text{free}}$
Algorithm:
$\mathfrak{R} \leftarrow \emptyset$
while *CheckCoverage*$(\mathfrak{R}) \leq \alpha$ **do**
  $\mathcal{G} \leftarrow$ SampleVisibilityGraph$(K, \mathfrak{R})$
  $\mathcal{T} \leftarrow$ TruncatedCliqueCover$(\mathcal{G}, s_{\min})$
  $\mathfrak{B} \leftarrow$ MinVolumeEllipsoids$(\mathcal{T})$
  $\mathfrak{R} \leftarrow \mathfrak{R} \cup$ InflatePolytopes$(\mathfrak{B})$
end
return $\mathfrak{R}$

5.7.3.2 Truncated Clique Cover

In the subroutine TruncatedCliqueCover, we approximately cover the visibility graph with a collection of cliques, each of which contains at least $s_{\min}$ vertices. We construct this approximate cover $\mathcal{T}$ greedily, by solving a sequence of **MaxClique** problems. Each instance of **MaxClique** is formulated as an integer linear program

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{K} b_i & & & (5.34a) \\ \text{subject to} \quad & b_i + b_j \leq 1, & & \forall \{i,j\} \in \bar{\mathcal{E}}, & (5.34b) \\ & b_i \in \{0,1\}, & & \forall i = 1, \ldots, K. & (5.34c) \end{aligned}$$

A binary decision variable b_i is added for each vertex. The role of this variable is to take unit value if and only if vertex i is included in the clique. The set $\bar{\mathcal{E}}$ contains all the pairs of vertices $\{i,j\}$ such that $i \neq j$ and $\{i,j\} \notin \mathcal{E}$. Therefore the first constraint ensures that two vertices are selected only if they share an edge.

After solving the integer program (5.34), the clique found is removed from the graph and added to the clique cover. Since small cliques are not informative, we stop this iterative process when the largest clique left in the graph is smaller than a given threshold $s_{\min}$. For this reason, our clique covers will be truncated, i.e., generally, not all vertices will be contained in one of the cliques.

5.7.3.3 Summarizing Cliques with Ellipsoids

In the subroutine MinVolumeEllipsoids, we solve a semidefinite program to enclose each clique with an ellipsoid of minimum volume [42, §8.4.1]. This collection $\mathfrak{B}$ of ellipsoids allows us to summarize the geometry of each clique with a point and a set of principal

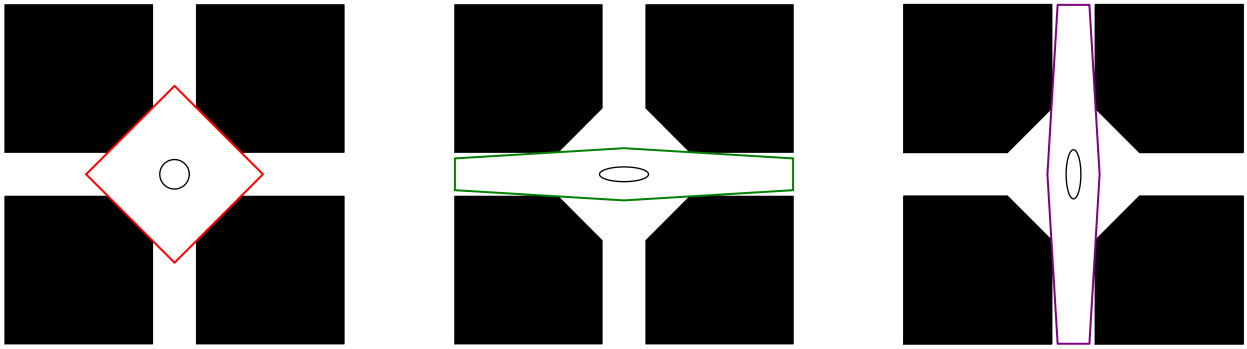


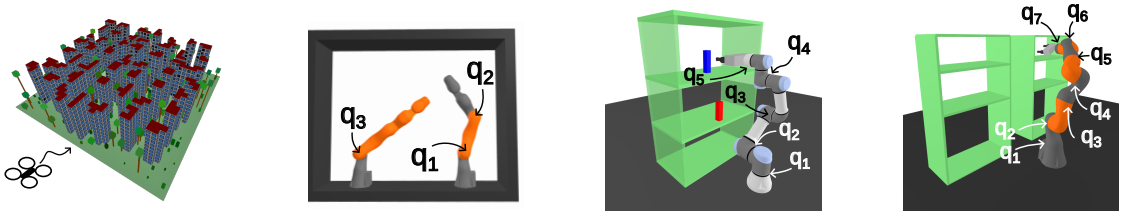
Figure 5.18: The growth direction of an IRIS region can be guided by the initial ellipsoid. We show IRIS initialized with three ellipsoids with the same center but different principal axes, resulting in polytopes that cover different portions of $\mathcal{C}^{\text{free}}$.

directions, which are then used to initialize the region-inflation algorithm. For the upcoming computations, it is necessary that the center of each ellipsoid is not in collision; if this is not the case, we recenter the ellipsoid around the vertex in the clique that is closest to its center.

5.7.3.4 Inflating Polytopes

In the last step of VCC, the subroutine INFLATEPOLYTOPES inflates a large, collision-free polytope around the center of each ellipsoid induced by a clique.

Let us initially assume that the obstacles are convex. Consider a single ellipsoid; using convex optimization, we compute the point in each obstacle that is closest to the center of the ellipsoid, according to the distance metric induced by the ellipsoid. These points anchor separating hyperplanes between the ellipsoid center and the obstacles, which form a polytope of obstacle-free space. These steps are repeated for each ellipsoid (i.e., for each clique) to obtain a collection of collision-free polytopes that we add to the set $\mathfrak{R}$.



(a) Village (b) 3DOF Flipper (c) 5DOF UR3 (d) 7DOF IIWA

These computations correspond to a single iteration of the IRIS algorithm [145], and ensure that the largest uniformly scaled, collision-free version of the ellipsoid is contained in the resulting polytope. Figure 5.18 illustrates how the initial metric is fundamental in guiding the shape of the regions generated by IRIS. Traditionally, the IRIS algorithm is initialized with an uninformed ellipsoidal metric and needs to run for multiple (expensive) iterations in order to expand and cover a larger volume of space; in VCC we use only a single IRIS iteration and the initial ellipsoid is informed by the shape of the clique.

When the obstacles are not convex, we run one iteration of the nonlinear programming variant IRISNP [164] instead. Alternatively, C-IRIS [115,116] could be employed to obtain

Domain	Village		3DOF Flipper		5DOF UR3		7DOF IIWA	
Algorithm	IOS	VCC (ours)	IOS	VCC (ours)	IOS	VCC (ours)	IOS	VCC (ours)
# regions $\lvert\mathfrak{R}\rvert$	198.0±13.6	**93.9**±7.5	10.4±1.9	**6.7**±0.5	90.5±18.8	**35.1**±1.6	482.6±83.8	**46.3**±4.5
runtime [s]	2.7e3±2.9e2	**1.7e3**±3.4e2	2.0e2±2.9e1	**4.9e1**±7.1	1.12e4±2.1e3	**5.5e2**±6.6e1	8.9e4±1.9e4	**5.7e3**± 1.7e3
coverage threshold α	0.8	0.8	0.9	0.9	0.75	0.75	0.7	0.7
# visibility vertices K	-	500	-	500	-	1000	-	1500
min. clique size $s_{\min}$	-	10	-	10	-	10	-	20

Table 5.2: Comparison of our Visibility Clique Cover (VCC) algorithm to the Iterative Obstacle Seeding (IOS) method from [164], across four different environments. All experiments are repeated ten times. The numbers in the first two rows indicate the mean and the empirical standard deviation over the trials. We observe that VCC achieves the given coverage targets with 1.6 to 10.4 times fewer regions, and between 1.4 and 20 times faster than IOS. The environment names are linked to interactive visualizations.

certifiably collision-free regions.

5.7.3.5 Convergence Check

The subroutine CheckCoverage estimates the fraction of $\mathcal{C}^{\text{free}}$ covered by the regions in $\mathfrak{R}$, and terminates our algorithm if this value exceeds the threshold α. Computing this fraction exactly is impractical, and so we resort to randomized methods. The coverage is estimated by drawing a large number of M samples in $\mathcal{C}^{\text{free}}$, and computing the ratio of samples that land in at least one of the regions in $\mathfrak{R}$. More sophisticated checks, such as one-sided Bernoulli hypothesis testing, are possible.

5.7.3.6 Completeness

Analogous to [126,175], VCC is probabilistically complete under mild assumptions. This means as the number of iterations goes to infinity, the probability of completely covering $\mathcal{C}^{\text{free}}$ goes to one.

5.7.4 Results

As there are no direct baseline methods, we compare our VCC algorithm against an extension of the method in [164, §III.D]. The natural extension of this approach is to iteratively grow polytopes around uniformly sampled points from the uncovered free space using Iris, while treating previously computed regions as obstacles. This process is repeated until the desired coverage is met. We call this approach Iterative Obstacle Seeding (IOS).

In the following, all experiments are implemented in `Drake`[15] [69], and all computations are performed on a single desktop computer with an Intel Core i9-10850K CPU and 32 Gb of RAM. We solve all **MaxClique** instances to global optimality using Gurobi [176]. An open-source implementation is available as the `IrisInConfigurationSpaceFromCliqueCover`[16] and supports the use of any algorithm in the Iris family supported by `Drake`.

[15] https://drake.mit.edu/

[16] https://drake.mit.edu/doxygen_cxx/group__planning__iris.html#ga95b21d4800c09233dd7a489238bced19

We evaluate VCC and IOS on four environments: `Village`, `3DOF Flipper`, `5DOF UR3`, and `7DOF IIWA`. The dimension n of these environments ranges from 3 to 7. For each environment, we run the two algorithms ten times and report their performance in Table 5.2. The `Village` environment from [177] contains only convex obstacles and is compatible with the original IRIS [145]. All other examples involve the configuration spaces of robotic manipulators and therefore IRISNP [164] is employed. The number of samples used in the convergence check in Algorithm 4 is $M = 5000$.

VCC meets the required coverage threshold with significantly fewer regions and in a substantially shorter amount of time. Notably, in the most challenging benchmark, `7DOF IIWA`, VCC requires 10 times fewer regions and meets the required coverage of 70% around 16 times faster. This substantial speedup can be attributed to two key factors. First, the region inflation in VCC is parallelized. Second, IRIS is the most computationally expensive step. While VCC only requires a single iteration of IRIS per region, IOS can require around five IRIS iterations per region, due to the initialization with a potentially uninformative spherical metric.

5.7.5 Limitations of Approximating Convex Sets with Cliques

Despite the strong performance of our algorithm, the hardness of solving **MIN α-APPRO XCONVEXCOVER** makes it possible to construct simple examples that highlight pitfalls of our heuristic approach. In this section, we discuss holes in $\mathcal{C}^{\text{free}}$ which lead to one such pitfall that is particularly insightful.

While every convex set in $\mathcal{C}^{\text{free}}$ naturally corresponds to a clique in the visibility graph, a clique in the visibility graph does not necessarily correspond to a convex set in $\mathcal{C}^{\text{free}}$. The convex hull of a clique can enclose obstacles in $\mathcal{C}^{\text{free}}$. This problem persists even if we sample arbitrarily dense visibility graphs, and if we restrict the analysis to maximal cliques.

A visual proof is shown in Figure 5.20, which illustrates a triangular configuration space with a triangular hole of size ε. As ε goes to zero, the largest convex subset of $\mathcal{C}^{\text{free}}$ is the green trapezoid, whose area approaches 5/9 of the total area of $\mathcal{C}^{\text{free}}$. On the other hand, a larger subset of mutually visible points is given by the union of the three red parallelograms, whose area approaches 6/9 of the total area. Therefore, if configurations are sampled uniformly at random, an optimal solution of **MAXCLIQUE** will almost surely enclose a hole as the number of samples goes to infinity. A similar construction can be used to show an analogous discrepancy between the minimum convex cover of $\mathcal{C}^{\text{free}}$ and **MINCLIQUECOVER**.

In principle, this problem can be addressed by solving a modified version of **MAXCLIQUE** that better captures the notion of a convex set. In short, we require that every vertex of the visibility graph that is contained in the convex hull of the maximum clique must be a member of the clique. For an infinitely dense visibility graph, this ensures that the maximum clique cannot enclose holes.

We enforce the contrapositive of the latter condition through linear constraints that separate all non-clique members q_i, with $i \in \{1, \dots, K\}$, from the points in the clique with a hyperplane $\mathcal{H}_i = \{q \mid c_i^T q + d_i = 0\}$, parameterized by the decision variables $(c_i, d_i) \in \mathbb{R}^{n+1}$.

The resulting mixed-integer linear optimization extends (5.34) by adding the additional

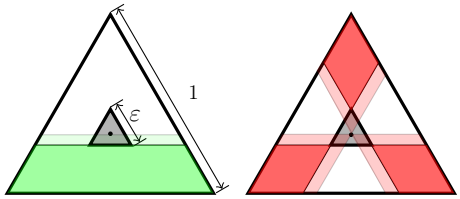


Figure 5.20: Maximum cliques of infinitely dense visibility graphs can enclose holes, and do not necessarily correspond to collision-free convex sets. The largest collision-free convex region (green trapezoid) has a smaller area than the union of red parallelograms when $0 < \varepsilon \leq 1 - \sqrt{5/6}$. In this case, the convex hull of the maximum clique must enclose the hole. Note that the vertices in the red regions form a valid clique because the pairwise convex hulls of the red regions are collision-free.

separation constraints

$$c_i^T q_i + d_i \geq 1 - b_i, \qquad \forall\ i = 1, \ldots, K, \tag{5.35a}$$

$$c_i^T q_j + d_i \leq L(1 - b_j), \qquad \forall i, j = 1, \ldots, K, \tag{5.35b}$$

where L is a large enough constant (e.g. a constant factor times the diameter of the visibility graph).

When the point q_i is not in the clique ($b_i = 0$) and the point q_j is in the clique ($b_j = 1$), these constraints read $c_i^T q_i + d_i \geq 1$ and $c_i^T q_j + d_i \leq 0$. Therefore, the hyperplane $\mathcal{H}_i$ separates q_i from q_j. On the other hand, these constraints are seen to be redundant for all other possible values of the binaries b_i and b_j. In Figure 5.21, we demonstrate how this extension prevents a maximum clique from enclosing holes in a finite-sample regime.

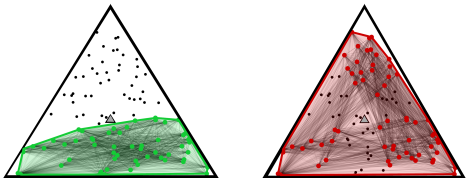


Figure 5.21: A visibility graph with 100 random vertices in the triangular environment from Figure 5.20. Solving the maximum clique problem (5.34) with the additional constraints (5.35) yields a clique with 56 vertices (shown in green), that closely approximates the corresponding convex set in Figure 5.20. Solving only problem (5.34), yields a clique with 63 vertices (shown in red) that, however, encloses the central hole.

In practice, solving (5.34) with constraints (5.35) is too expensive for the problems in Table 5.2. Nonetheless, we observed that in the first MaxClique problem (5.34) only an average of 0.1% of the vertices were excluded from the clique that could not be separated from it with a hyperplane. Subsequent cliques, and improvements to the formulation (5.35), demand a more nuanced discussion and are subject to future work.

5.8 Conclusions

In this chapter, we have demonstrated that SOS programming can be an effective tool for addressing a canonical robotics problem: describing the set of collision-free configurations. We demonstrated a method for verifying that a convex subset of an algebraic reparametrization of $\mathcal{C}^{\text{space}}$ called $\mathcal{T}^{\text{space}}$ is collision free. The certification method can be used to prove that a given trajectory is collision free very quickly, or can be used as part of a subroutine for growing a full-dimensional collision-free region in $\mathcal{T}^{\text{free}}$. Finally, inspired by the many algorithms for finding full-dimensional subsets of $\mathcal{C}^{\text{free}}$ and the utility of such regions, we proposed a method for automatically placing these regions to cover the space.

5.9 Deferred Proofs

5.9.1 The Certification Programs are Necessary and Sufficient

In this section, we expand our discussion on the power of the certification programs presented in Sections 5.4.1 and 5.4.2. As remarked previously, Theorems 17 and 16 are necessary as programs (5.25) and (5.18) are infinite dimensional. It is not immediately obvious that for every robot and every scene, there exists a finite degree at which each program must become feasible when $\mathcal{P}$ truly contains no collisions.

A second subtlety applies specifically to (5.18). When generalizing (5.3), we argued that it was beneficial to search for a parametric hyperplane as a function of our $\mathcal{T}^{\text{space}}$ variable s and asserted that a polynomial parameterization was a good choice. However, it is not obvious that a polynomial parameterization is sufficient, and perhaps we require a rational or even more complicated parameterization of the plane.

These questions about the power of SOS programming arise in other domains. For example, SOS is commonly used to search for polynomial Lyapunov functions to prove the stability of polynomial dynamical systems [178]. However, it is known that not every stable polynomial dynamical system admits a polynomial Lyapunov function [179], and therefore SOS programming is a sufficient, but not necessary tool for proving the stability of dynamical systems.

Fortunately, our certification programs from 5.4.1 and 5.4.2 are indeed *necessary and sufficient*, in the sense that there will always exist a finite degree such that the programs become feasible if $\mathcal{P}$ contains no collisions. The proof of this for the program (5.25) follows immediately from Theorem 15.

Proof 5.2. **(of Theorem 17)** Our assumptions on $\mathcal{P}$, $\mathcal{A}$, and $\mathcal{B}$ imply that $\mathcal{S}_{\mathcal{P},\mathcal{A},\mathcal{B}}$ is an Archimedean set. Therefore, the feasibility of (5.25) for sufficiently high degree ρ follows immediately from "effective" versions of Theorem 15 such as those given in [180,181] which give explicit degree bounds. □

Though the proof of Theorem 16 is more technically involved, the key idea is simple. In short, we construct a family of continuous functions that map each $\mathcal{T}^{\text{space}}$ configuration $s \in \mathcal{P}$ to a separating plane. We then argue that this family of continuous functions must contain hyperplanes that are parametrized as polynomials. Finally, we again appeal to "effective"

versions of Theorem 15 such as those given in [180,181] to show that these polynomials can be found using SOS programming.

We proceed in steps, first establishing that the set of separating planes at a point s in $\mathcal{T}^{\text{free}}$ is open.

Proposition 1. *Let $\Phi(s)$ denote the set of strictly separating hyperplanes at the point s for bodies $\mathcal{A}(s)$ and $\mathcal{B}(s)$ and let $\mathcal{P}$ be a non-empty, polytopic subset of $\mathcal{T}^{\text{free}}$. Then $s \in \mathcal{P}$ implies that $\Phi(s)$ is a non-empty, open set.*

Proof 5.3. By definition, a hyperplane $\begin{bmatrix} a \\ b \end{bmatrix} \in \mathbb{R}^4$ strictly separates $\mathcal{A}(s)$ and $\mathcal{B}(s)$ if and only if there exist positive constants $\varepsilon_{\mathcal{A}}$ and $\varepsilon_{\mathcal{B}}$ such that $a^T x + b \geq \varepsilon_{\mathcal{A}}\ \forall x \in \mathcal{A}(s)$ and $a^T x + b \leq -\varepsilon_{\mathcal{B}}\ \forall x \in \mathcal{B}(s)$. Since the bodies $\mathcal{A}(s)$ and $\mathcal{B}(s)$ are strictly separating for every point $s \in \mathcal{P}$, the Separating Hyperplane theorem guarantees the existence of such a vector and so $\Phi(s)$ is non-empty for every s.

Now consider $\begin{bmatrix} a \\ b \end{bmatrix} \in \Phi(s) \subseteq \mathbb{R}^4$ and its δ neighborhood

$$\mathcal{N}(\delta) = \left\{ \begin{bmatrix} a \\ b \end{bmatrix} + \delta \begin{bmatrix} v_a \\ v_b \end{bmatrix} \middle| \left\| \begin{bmatrix} v_a \\ v_b \end{bmatrix} \right\| \leq 1 \right\},$$

with $\delta > 0$.

We have that for all $x \in \mathcal{A}$

$$(a^T + \delta v_a^T)x + (b + \delta v_b) \geq \varepsilon_{\mathcal{A}} + \delta \min_{\|v\| \leq 1,\ x \in \mathcal{A}} v_a^T x + v_b$$

and similarly for all $x \in \mathcal{B}$

$$(a^T + \delta v_a^T)x + (b + \delta v_b) \leq -\varepsilon_{\mathcal{B}} + \delta \max_{\|v\| \leq 1,\ x \in \mathcal{B}} v_a^T x + v_b.$$

Letting $M_l = \min_{\|v\|=1,\ x \in \mathcal{A}} v_a^T x + v_b$ and $M_u = \max_{\|v\|=1,\ x \in \mathcal{B}} v_a^T x + v_b$, we have that all planes $\mathcal{N}(\delta)$ are separating if

$$0 < \delta < \min \left\{ \frac{\varepsilon_{\mathcal{A}}}{|M_l|},\ \frac{\varepsilon_{\mathcal{B}}}{|M_u|} \right\}$$

and so $\Phi(s)$ is open.

□

Proposition 2. *Define*

$$\mathcal{N}(s, \delta) = \bigcap_{\|v\| \leq 1} \Phi(s + \delta v)$$

For all $s \in \mathcal{P}$ there exists $\delta_{\min}(s) > 0$, not necessarily finite such that, $\mathcal{N}(s, \delta)$ is non-empty and open for every $0 < \delta < \delta_{\min}$.

Proof 5.4. Recall that the position of every point in $\mathcal{A}(s)$ is a continuous function of s and that the distance from a point to a set is a continuous function [182]. Therefore, the distance of every point in $\mathcal{A}(s)$ to every element of $\Phi(s)$ changes continuously. For every $\delta > 0$ and $\begin{bmatrix} a \\ b \end{bmatrix} \in \Phi(s)$ we define:

$$M_\delta(s) \coloneqq \sup_{\|v\| \leq 1} \left| \inf_{x \in \mathcal{A}(s)} a^T x + b - \inf_{x \in \mathcal{A}(s+\delta v)} a^T x + b \right|$$

We have that

$$\inf_{\|v\| \leq 1} \inf_{x \in \mathcal{A}(s+\delta v)} a^T x + b \geq \inf_{x \in \mathcal{A}(s)} a^T x + b - M_\delta(s) \geq \varepsilon_{\mathcal{A}} - M_\delta(s)$$

Moreover, if $\delta_2 < \delta_1$, then $M_{\delta_2}(s) \leq M_{\delta_1}(s)$. By continuity and monotonicity, $M_\delta(s) \to 0$ as $\delta \to 0$ and so there exists δ sufficiently small such that $\varepsilon_{\mathcal{A}} - M_\delta(s) > 0$. A similar argument shows that δ can be chosen sufficiently small such that the plane $\begin{bmatrix} a \\ b \end{bmatrix}$ continues to satisfy the separating plane conditions for $\mathcal{B}$. Therefore $\begin{bmatrix} a \\ b \end{bmatrix} \in \Phi(s + \delta v)$ for all v such that $\|v\| \leq 1$ if δ is chosen sufficiently small. It is clear that choosing δ smaller continues to ensure that $\mathcal{N}(s, \delta)$ is non-empty. Openness is immediate following a similar argument to Proposition 1. □

The above proposition enables us to establish that there exists an open family of continuous functions $f(s)$ such that their outputs are always separating hyperplanes.

Proposition 3. *Let $\mathcal{F}$ be the set of continuous functions mapping*

$$f : s \mapsto \begin{bmatrix} a \\ b \end{bmatrix}$$

such that $f(s) \in \Phi(s)$ for all $s \in \mathcal{P}$. The set $\mathcal{F}$ is non-empty and open under the pointwise metric

$$d(f, g) = \sup_{s \in \mathcal{P}} \|f(s) - g(s)\| .$$

Proof 5.5. Suppose $\mathcal{F}$ were empty. Then every function satisfying $f(s) \in \Phi(s)\ \forall s \in \mathcal{P}$ is not a continuous function. Namely, for every f there exists a point s_0 such that for all $\delta > 0$, $f(s_0) \in \Phi(s_0)$ but $f(s_0) \notin \mathcal{N}(s_0, \delta)$. This contradicts the openness of $\mathcal{N}(s_0, \delta)$ for a sufficiently small δ from Proposition 2 and so $\mathcal{F}$ is non-empty. Openness follows from the fact that if $\delta > 0$ is chosen sufficiently small, then for every continuous g satisfying $d(f, g) < \delta$, g must also separate $\mathcal{A}(s)$ and $\mathcal{B}(s)$ for every $s \in \mathcal{P}$.

□

We are now ready to prove Theorem 16.

Proof 5.6. **(of Theorem 16)** By Proposition 3, $\mathcal{F}$ is a non-empty open subset of continuous functions defined on the compact domain $\mathcal{P}$. The Stone-Weierstrass theorem [182] states that the set of polynomial functions on a compact domain is dense in the set of continuous functions in that domain under the pointwise metric. Therefore, $\mathcal{F}$ must contain a map $p : s \mapsto \begin{bmatrix} a(s) \\ b(s) \end{bmatrix}$ such that each component is a polynomial. This polynomial is of finite degree and is a strictly separating hyperplane and therefore by "effective" versions of Theorem 15 such as [180,181], there exists a Putinar certificate of finite degree certifying that $p(s)$ is a separating hyperplane. □

Chapter 6

Practical Strengthening of Semidefinite Relaxations

In this chapter, we will demonstrate through examples that SOS relaxations are sensitive to how the original POP is formulated. This sensitivity has been observed throughout the literature in various forms and it is well-known that adding additional, redundant constraints to the original POP is an important part of modelling and forming an effective convex relaxation of real-world problems.

To this end, we will explore a method for automatically improving the formulation of POPs containing equality constraints so that their resulting SOS relaxation is stronger. We will do so in an efficient manner; our method will be fast enough to run as a simple pre-processing step, with a runtime negligible relative to solving the SOS relaxation and it will add relatively few additional constraints so as not to overly increase solve times.

We will consider SOS relaxations for programs of the form

$$\begin{aligned} &\min\ p(x) \\ &\text{subject to } h_i(x) = 0,\ i \in [M]. \end{aligned} \tag{6.1}$$

and their degree $2D$ SOS relaxation

$$\max\ \gamma \tag{6.2a}$$

$$\text{subject to } p - \gamma = \sigma + \sum_{i=1}^{M} \lambda_i h_i \tag{6.2b}$$

$$\lambda_i \in \mathbb{R}[x]_{2D-\deg(h_i)},\ \sigma \in \mathbf{\Sigma}[x]_{2D}. \tag{6.2c}$$

Our method will apply to more general polynomial programs with inequalities, but the inequalities will have no impact on our method.

We call a constraint $\hat{h}(x) = 0$ redundant or an implied equality for (6.1) if it does not change the feasible set. Though redundant equalities $\hat{h}_k(x) = 0$ do not change (6.1), they do change (6.2) as their inclusion changes (6.2b) to

$$p - \gamma = \sigma + \sum_{j=1}^{M} \lambda_j h_j + \sum_{k} \hat{\lambda}_k \hat{h}_k.$$

Simple examples show that adding redundant constraints can significantly strengthen SOS relaxations. We give one such example in Example 3, which we revisit here.

Example 9. *Consider the POP*

$$\begin{aligned} \min\ & x \\ \text{subject to } & x^2y - x = 0 \\ & x^d = 0, \end{aligned} \tag{6.3}$$

for a given $d \in \mathbb{N}$. The degree $2D$ SOS relaxation of (6.3) attains the globally optimal value of 0 if $2D \geq 2d-1$ since

$$x - 0 = -\left(\sum_{i=0}^{d-2} x^i y^i\right)(x^2y - x) + y^{d-1}(x^d).$$

If $2D < d$, the relaxation is infeasible and gives the trivial bound $-\infty$.

By contrast, the SOS relaxation of the POP

$$\begin{aligned} \min\ & x \\ \text{subject to } & x^2y - x = 0 \\ & x^d = 0 \\ & x^2 = 0. \end{aligned} \tag{6.4}$$

attains value 0 for every $D > 1$ since

$$x - 0 = -(x^2y - x) + 0(x^d) + y(x^2).$$

The only difference between (6.3) and (6.4) is the redundant equality $x^2 = 0$. This extra equality is useful because it lowers the certificate degree, even though it does not change the feasible set. For a proof of why this is the case, we refer the reader back to Example 3.

Given this sensitivity of SOS relaxations to the formulation of this problem, in this chapter we consider the problem of algorithmically generating useful implied equalities. We provide a simple necessary condition for checking whether an implied equality can strengthen a fixed degree SOS relaxation that amounts to solving a linear system. We combine this with a simple linear-algebraic procedure that generates all implied equalities that can be proven in fixed-degree. Together, this leads to a linear-algebraic procedure for generating candidate implied equalities that can strengthen the relaxation. We illustrate the construction in the quadratic case, where useful candidate equalities can be generated by a nullspace computation followed by a linear projection, turning their design into a lightweight preprocessing step. Finally, we generalize the construction to arbitrary programs of the form (6.1).

6.1 Introduction and Related Work

We denote by $\mathcal{V} \coloneqq \{x \in \mathbb{R}^n \mid h_i(x) = 0,\ i \in [M]\}$ the real affine variety associated to the program (6.1). Similarly, we will denote by $\mathcal{I} \coloneqq \langle h_1, \ldots, h_M \rangle$ the real ideal generated by

the h_i's. We emphasize that $\mathcal{V} \subseteq \mathbb{R}^n$ is a set of points in space while $\mathcal{I} \subseteq \mathbb{R}[x]$ is a subset of polynomials.

The power of the relaxation (6.2) is intrinsically tied to checking whether a polynomial $q(x)$ is nonnegative on $\mathcal{V}$. Following [183], we distinguish two traditional ways for certifying this. The first is known as a degree $2D$ quotient ring SOS certificate. This method checks whether

$$q \equiv \sigma \mod \mathcal{I},\ \sigma \in \mathbf{\Sigma}[x]_{\leq 2D}. \tag{6.5}$$

A Gröbner basis [184] is a special set of generators for the ideal $\mathcal{I}$ and allows us to cast (6.5) as a semidefinite program [52, §3.3.5].

The second kind of certificates are the kind we consider in Chapter 2; we search for polynomials λ_i such

$$q = \sigma + \sum_{i=1}^{M} \lambda_i h_i,\ \sigma \in \mathbf{\Sigma}[x]_{\leq 2D},\ \lambda_i \in \mathbb{R}[x]_{2D-\deg(h_i)}. \tag{6.6}$$

This can also be cast as a semidefinite program (see Chapter 2).

When a Gröbner basis is available, (6.5) is substantially more appealing than (6.6). It produces programs which are both smaller and more powerful than the formulation in (6.6). Unfortunately, computing a Gröbner basis could be extremely expensive, requiring doubly exponential time in general [77,78] and frequently destroys almost all of the sparsity in the program [185]. Cheaper alternatives to computing Gröbner bases include basis-selection methods [186] and sampling-based constructions [183]. The former still similarly destroys the sparsity of the problem as it still relies on polynomial reduction, while the latter is only available if the variety is amenable to sampling. For many robotics problems, even finding a feasible solution can be challenging and so sampling the variety is not possible.

The aforementioned difficulties make (6.6) the default for most practitioners. In this setting, the practical importance of adding redundant equalities to (6.6) has been reported in several robotics works [32,187–190]. In the literature, the additional equalities are often added by hand and constitute a significant part of the paper's contribution, but with limited explanation of why they help or how they were discovered [32,95,191,192].

Automating the search for these constraints has only recently been a topic of interest in robotics. Most closely related to the current work is [190], which uses similar nullspace computations as in our method to generate implied equalities, but relies on sampling which can be difficult to perform efficiently, if at all. Moreover, the method can return constraints that are weak or difficult to interpret. By contrast, our method avoids solution sampling and directly characterizes the implied equalities that can strengthen a SOS relaxation.

Gröbner bases are in a formal sense the best set of implied equalities one could add to (6.6). It is therefore unsurprising that our method is closely related to algorithms which compute Gröbner bases such as Buchberger's algorithm [193] and Faugère's F4 and F5 [194,195]. However, our algorithm is much simpler, at the expense of being weaker. It involves only simple linear algebra and no symbolic algebra.

6.2 Simulation Lemma: When Can a New Equality Help?

In this section, we provide a simple condition for testing whether a redundant equality can add strength to a SOS relaxation. All the results in the remainder of this chapter depend on the set of generators $h_1, \ldots, h_M$ used to describe the ideal $\mathcal{I} = \langle h_1, \ldots, h_M \rangle$, i.e. the set of equations $h_1(x) = 0, \ldots, h_M(x) = 0$, and so we assume these are given and fixed.

First, we recall that by Theorem 9, $\hat{h}(x)$ is redundant if $\hat{h}(x)$ is in the real radical ideal [52, Definition A.51] i.e. we can express it as

$$\hat{h}^{2r} + \sigma \in \langle h_1, \ldots, h_M \rangle, \ \sigma \in \mathbf{\Sigma}[x],$$

for some $r \in \mathbb{N}$. Unfortunately, certificates of this form require semidefinite programming to compute.

Instead, we will use the sufficient condition that $\hat{h} \in \mathcal{I}$. We recall the notion of proving $\hat{h} \in \mathcal{I}$ in fixed degree from Definition 25 and the minimum degree of an ideal membership proof from (1.9).

Definition 45 (Ideal Membership in Fixed Degree)**.** *After fixing a degree $\hat{D}$, we consider the set of polynomials $\hat{h}$ which can be written as*

$$\hat{h}(x) = \sum_{i=1}^{M} \mu_i h_i, \ \mu_i \in \mathbb{R}[x]_{\leq \hat{D} - \deg(h_i)}. \tag{6.7}$$

We call these the polynomials whose membership in the ideal can be proven in degree $\hat{D}$. We denote the set of all such polynomials $\langle h_1, \ldots, h_M \rangle_{\leq \hat{D}}$ or $\mathcal{I}_{\leq \hat{D}}$.

We now turn our attention back to SOS programming. While it is true that every $\hat{h} \in \mathcal{I}$ is redundant, not every $\hat{h}$ helps strengthen a SOS relaxation.

Lemma 2. *Choose $2D$. Let $\hat{h}$ have degree $\hat{d}$ and suppose that $\hat{h} \in \langle h_1, \ldots, h_M \rangle_{\leq \hat{D}}$, and so there exists μ_i such that*

$$\hat{h} = \sum_i \mu_i h_i, \ \mu_i \in \mathbb{R}[x]_{\leq \hat{D} - \deg(h_i)}. \tag{6.8}$$

If there is a degree $2D$ certificate of the form

$$\begin{gathered} p = s + \sum_i \lambda_i h_i + \hat{\lambda}\hat{h} \\ s \in \mathbf{\Sigma}[x]_{\leq 2D}, \ \lambda_i \in \mathbb{R}[x]_{\leq 2D - \deg(h_i)}, \ \hat{\lambda} \in \mathbb{R}[x]_{2D - \hat{d}}, \end{gathered} \tag{6.9}$$

then there is a certificate of degree at most $2D + \hat{D} - \hat{d}$ of the form

$$p = s + \sum_i \lambda_i' h_i. \tag{6.10}$$

Proof 6.1. We expand the certificate (6.9) using the certificate of $\hat{h}$'s membership in the ideal

$$\begin{aligned} p &= s + \sum_i \lambda_i h_i + \hat{\lambda}\hat{h} \\ &= s + \sum_i \lambda_i h_i + \hat{\lambda} \sum_i \mu_i h_i \\ &= s + \sum_i (\lambda_i + \hat{\lambda}\mu_i) h_i \end{aligned}$$

Now we just need to argue that $\deg((\lambda_i + \hat{\lambda}\mu_i)h_i) \leq 2D + \hat{D} - \hat{d}$. The $\deg(\lambda_i h_i) \leq 2D$ by assumption. For the second term, from (6.8) $\deg(\mu_i) \leq \hat{D} - \deg(h_i)$ and from (6.9) we have that $\deg(\hat{\lambda}) \leq 2D - \hat{d}$. Therefore,

$$\begin{aligned} \deg(\hat{\lambda}\mu_i h_i) &\leq 2D - \hat{d} + \hat{D} - \deg(h_i) + \deg(h_i) \\ &= 2D + \hat{D} - \hat{d} \end{aligned}$$

□

Lemma 2 shows that the effect of adding a redundant $\hat{h}$ of degree $\hat{d}$ to the relaxation can be simulated by simply increasing the degree of the relaxation. However, increasing the degree of the relaxation is frequently intractable, so it is usually computationally preferred to include the $\hat{h}$. An immediate corollary of Lemma 2 characterizes which implied constraints *do not* help strengthen the relaxation.

Corollary 1. *Let $\hat{h}$ have degree $\hat{d}$ and suppose there exists a degree $\hat{D}$ certificate that $\hat{h} \in \langle h_1, \ldots, h_M \rangle$. Then if $\hat{d} = \hat{D}$, adding $\hat{h}$ as an implied equality to (6.1) cannot strengthen a SOS certificate.*

The corollary says that only a *degree gap* can matter. An implied equality is only potentially useful when proving its membership in the original ideal requires higher degree than the equality itself. In symbols, redundant equalities $\hat{h}(x) = 0$ satisfying $\deg_{\langle h_1,\ldots,h_M\rangle}(\hat{h}) = \deg(\hat{h})$ don't strengthen the relaxation. Corollary 1 is a necessary condition. It cannot guarantee that a given implied equality will certainly help strengthen a relaxation, but it can discard some which are provably useless.

The condition can be used to rigorously explain why specific implied equalities can help strengthen SOS relaxations of polynomial optimization programs found in the literature.

Example 10 (Contact Planning [95])**.** *In [95], a QCQP model of planning through contact is presented that includes quadratic equalities of the form $Ru = w$, where $R \in SO(n)$ is a rotation matrix. The authors report that including the constraint $u = R^T w$ strengthens their semidefinite relaxation. We can prove that $u = R^T w$ from $Ru = w$ and $R^T R = I$ in degree 3:*

$$R^T w - u = -R^T(Ru - w) + (R^T R - I)u.$$

No lower degree proof exists, verifying the empirical finding that $R^T w = u$ can strengthen their SOS relaxation.

Example 11 (Pose Estimation [32,191]). *In [32, Eq. 23], a QCQP model of robust pose estimation is presented that includes the constraints $x_0^T x_0 = 1$ and $x_i x_i^T = x_0 x_0^T$ for $i \in [N]$. These imply that $x_i x_j^T = x_j x_i^T$ and a degree-4 certificate of this fact is given in Eq. 24 of the same paper. It can be shown that no lower degree certificate can exist, again supporting the empirical finding that the proposed constraint strengthens the relaxation.*

Example 12 (Stereo Vision [190]). *In [190, Eq. 5], a QCQP for 1D camera localization is presented. Again, a redundant constraint is proposed in Eq. 8 of the paper, and a minimal degree 4 certificate for the implied equality is given, validating the claim that the implied constraint can help the relaxation.*

Notice that given the fixed generators $h_1, \ldots, h_M$ of the ideal, condition (6.8) is *linear* in the coefficients of both $\hat{h}$ and the multiplier polynomials μ_i. This means that if one is given a polynomial $\hat{h}$ of degree $\hat{d}$ in n variables, then we can check whether a certificate of degree $\hat{D}$ exists by solving a linear system. This linear system has at most $\sum_{i=1}^{M} \binom{n+\hat{D}-\deg(h_i)}{n}$ unknowns and $\binom{n+\hat{d}}{n}$ equations. In particular, Corollary 1 implies that if (6.8) is feasible when $\hat{D} = \hat{d}$, then the redundant equality $\hat{h}$ does not help, while if it is infeasible, it may help.

Example 13 (Checking ideal membership). *Let $\mathcal{I} = \langle h_1, h_2 \rangle \subseteq \mathbb{R}[x, y]$ where*

$$h_1(x, y) = x + y, \quad h_2(x, y) = x - y.$$

Consider the degree-two polynomial $\hat{h}(x, y) = x^2 - y^2$. To check whether $\hat{h} \in \mathcal{I}$ with proof degree $\hat{D} = 2$, we look for

$$\hat{h} = \mu_1 h_1 + \mu_2 h_2, \quad \mu_1, \mu_2 \in \mathbb{R}[x, y]_{\leq 1}.$$

A degree 2 proof requires searching over polynomials

$$\mu_1(x, y) = a_0 + a_1 x + a_2 y, \quad \mu_2(x, y) = b_0 + b_1 x + b_2 y.$$

We explicitly write out the expression for $\hat{h}$ as a linear map. The columns of the matrix are indexed first by h_1 and its product with the basis elements $[1, x, y]$, then by h_2 and its product with the basis elements. The rows are indexed by the coefficients of the monomials of $\hat{h}$.

Coefficient matching gives the indexed linear system

$$\hat{h}(x) \cong \begin{array}{c} \\ 1 \\ x \\ y \\ x^2 \\ xy \\ y^2 \end{array} \begin{array}{c} \begin{array}{cccccc} (h_1 * 1) & (h_1 * x) & (h_1 * y) & (h_2 * 1) & (h_2 * x) & (h_2 * y) \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & -1 & 1 \\ 0 & 0 & 1 & 0 & 0 & -1 \end{bmatrix} \end{array} \begin{array}{c} \\ \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ b_0 \\ b_1 \\ b_2 \end{bmatrix} \end{array} \begin{array}{l} \\ (\mu_1, 1) \\ (\mu_1, x) \\ (\mu_1, y) \\ (\mu_2, 1) \\ (\mu_2, x) \\ (\mu_2, y) \end{array} = \begin{array}{c} \\ \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ -1 \end{bmatrix} \end{array} \begin{array}{l} 1 \\ x \\ y \\ x^2 \\ xy \\ y^2 \end{array}.$$

One solution is

$$a_0 = a_1 = a_2 = b_0 = 0, \quad b_1 = b_2 = 1,$$

which gives $\mu_1 = 0$ *and* $\mu_2 = x + y$*. Thus*

$$\hat{h}(x, y) = x^2 - y^2 = 0(x + y) + (x + y)(x - y) = \mu_1 h_1 + \mu_2 h_2,$$

so the linear system certifies that $\hat{h} \in \langle x + y, x - y \rangle_{\leq 2}$*.*

By Corollary 1, $\hat{h}$ *would not strengthen a SOS relaxation of a POP with these constraints since* $\deg(\hat{h}) = 2$ *and a degree* 2 *certificate of its membership in* $\langle x + y, x - y \rangle$ *exists.*

We next describe our procedure to generate new, but useful redundant equalities for (6.1). In what follows, we fix a degree $2D$ which is the target degree of the SOS relaxation. We select a degree $\hat{D} \geq 2D$ and search for polynomials $\hat{h}$ which have degree no more than $2D$ and whose membership in the ideal can be proven in degree $\hat{D}$.

The key idea is simple. The set of all redundant equalities of degree $2D$ or less that can be proven in degree $\hat{D}$ is a subspace of $\mathbb{R}[x]$. A basis of this set can be computed. However, this subspace will typically be quite large and contains many polynomials which cannot add any strength to our relaxation. We appeal to the condition in Corollary 1 to remove those that add no strength.

We begin by explaining the procedure when applied to generating new, redundant constraints for degree at most 2 for homogeneous QCQPs. SDP relaxations of QCQPs are becoming increasingly popular in robotics [32,34,95,190,192], and as shown in Examples 10, 11 and 12 frequently require tightening via redundant equalities to be practically useful. This setting allows us to be very concrete with our construction, before moving onto the more general setting.

6.3 Generating New Quadratic Equalities for QCQPs

We begin this section by analyzing the case of QCQPs, as it has become a particularly popular form for expressing polynomial programs in robotics and the notation makes it easier to be concrete.

Frequently in robotics, only the degree 2 relaxation is sufficiently cheap to solve in practice, and therefore we focus on generating new linear and quadratic equalities for this case. These degree 2 relaxations are the same (or the dual of the) relaxation given in Problem 2.3.

6.3.1 Warmup: Linearly Constrained Quadratic Program

Consider the following homogenized QCQP with only linear equality constraints

$$\begin{aligned} &\min\ x^T Q_0 x \\ &\text{subject to } Ax = 0, \quad e_0^T x = 1, \end{aligned} \tag{6.11}$$

and assume that e_0 is not in the rowspace of A[1].

[1]This ensures that $e_0^T x = 1$ is not redundant, i.e. implied by some linear combination of the rows of Ax.

Suppose we have decided a priori that we wish to form a degree 2 relaxation.

$$\begin{aligned} &\max \gamma \\ &\text{subject to } x^T Q_0 x - \gamma \;\; = x^T \Omega x + x^T(\Lambda^T A + A^T \Lambda)x - \lambda_0(x)(e_0^T x - 1) \\ &\qquad\qquad \Omega \succeq 0 \end{aligned} \tag{6.12}$$

Naturally, one may consider whether any additional linear equalities can help such a relaxation. We start by changing coordinates. Let x_0 be a solution to $Ax = 0$ and $e_0^T x = 1$. If no such x_0 exists, then the program is trivially infeasible. We change coordinates to $y = x - x_0$ and instead consider the equations $Ay = 0$ and $e_0^T y = 0$. Let $\tilde{A} = \begin{bmatrix} A \\ e_0^T \end{bmatrix}$. This allows us to consider the homogeneous polynomial equality $\tilde{A}y = 0$.

Consider a vector of polynomials $\mu(y) + \zeta$ where each polynomial in the vector $\mu(y)$ has no constant term and is of arbitrary degree. The set of all affine equalities that can be generated by $\tilde{A}y = 0$ can be parametrized as:

$$\hat{a}^T y = (\mu(y) + \zeta)^T \tilde{A} y \tag{6.13a}$$

$$= \underbrace{\mu(y)^T \tilde{A} y}_{\deg>1} + \underbrace{(\tilde{A}^T \zeta)^T y}_{\deg=1} \tag{6.13b}$$

$$\tilde{A}^T \zeta = \hat{a}, \quad \mu(y)^T \tilde{A} y = 0. \tag{6.13c}$$

From (6.13a) to (6.13b), all we have done is group terms by degree. From (6.13b) to (6.13c), we have matched coefficients by degree on either side of the equality in (6.13b).

These equations tell us two things. First, the set of all linear equalities we can add are $\hat{a} = \tilde{A}^T \zeta \implies \hat{a} \in \mathrm{Im}(\tilde{A}^T)$. Therefore, every equality we could add is already parametrized by $\tilde{A}y = 0$. Second, the equations imply that if we have a higher degree proof that $Ay = 0,\ e_0^T y = 0 \implies \hat{a}^T y = 0$, then we can construct a proof of degree 1. By Lemma 2 this means that every new affine equality constraint we could add does not help our relaxation.

Naturally, we next check if there are any new quadratic equalities we could add to our formulation. We proceed similarly. We parametrize all homogeneous quadratic forms and let $\mu(y)$ be a vector of polynomials with no linear or constant terms. All possible quadratic equalities we could add are parametrized as

$$\begin{aligned} y^T P y &= (\mu(y) + By + \zeta)^T \tilde{A} y \\ &= \underbrace{\mu(y)^T \tilde{A} y}_{\deg>2} + \underbrace{\frac{1}{2} y^T (B^T \tilde{A} + \tilde{A}^T B) y}_{\deg=2} + \underbrace{(\tilde{A}^T \zeta)^T y}_{\deg=1} \end{aligned} \tag{6.14a}$$

$$\tilde{A}^T \zeta = 0, \;\; \frac{1}{2}(B^T \tilde{A} + \tilde{A}^T B) = P. \tag{6.14b}$$

Where again, from (6.14a) to (6.14b) we have simply matched the coefficients of the two sides. This reveals two facts. First, all possible quadratic equalities we can add to (6.11) are $P = \{B^T \tilde{A} + \tilde{A}^T B \mid B \in \mathbb{R}^{(m+1)\times n}\}$. This can be represented as $\tilde{A} y y^T = 0$.

Again notice that the higher degree terms $\mu(y)$ do not factor in, and so we can safely set them to 0. Once again, we have that if $Ay = 0,\ e_0^T y = 0 \implies y^T P y = 0$, this can be proven in degree at most 2 and so by Lemma 2 this *does not help our relaxation.*

We conclude this section by remarking that in general, affine equalities cannot imply any higher degree implied equalities. This can be proven by continuing to argue in the manner presented in this section.

6.3.2 Explicit Low Degree Construction

We now move onto the more general equality constrained QCQP case by allowing quadratic equalities.

$$\begin{aligned} \min\ & x^T Q_0 x \\ \text{subject to } & x^T P_i x = 0 \\ & Ax = 0,\ e_0^T x = 1. \end{aligned}$$

We can replace $Ax = 0$ with $Axx^T = 0$ without affecting the SOS relaxation[2], and so moving forward, we will assume that the linear equalities have been written as quadratic equalities instead. Therefore, we analyze the QCQP

$$\begin{aligned} \min\ & x^T Q_0 x \\ \text{subject to } & x^T P_i x = 0,\ e_0^T x = 1. \end{aligned} \tag{6.15}$$

The first thing we note is that the homogenizing trick we used in Section 6.3.1 cannot be used. Indeed, if we take x_0 to be a solution of $x^T P_i x = 0$ and $e_0^T x = 1$, then performing the substitution $y = x - x_0$ results in

$$(y + x_0)^T P_i (y + x_0) = y^T P_i y + 2x_0^T P_i y + x_0^T P_i x_0,$$

which is not homogeneous. Therefore, we will see that genuinely new, non-trivial redundant constraints can be found.

We begin our search by looking for new homogeneous quadratics. From Corollary 1, the first non-trivial polynomials we could add are homogeneous quadratics which can be proven in degree at least 3. We can parametrize the set of all homogeneous quadratics with a degree at most 3 proof as

$$x^T \hat{P} x = \sum_i (a_i^T x + b_i)(x^T P_i x) + (x^T S_0 x + a_0^T x + b_0)(e_0^T x - 1). \tag{6.16}$$

Note that this also implicitly allows for a search of implied linear constraints via quadratics parametrized as

$$\hat{P} = \begin{bmatrix} 0 & q^T \\ q & 0 \end{bmatrix}$$

since $e_0^T x = 1$.

[2]For the degree-2 relaxation this is degree-preserving. Each entry of Axx^T is obtained from Ax by multiplying by a coordinate of x. Conversely, $Ax = Axx^T e_0 - Ax(e_0^T x - 1)$ so any affine-multiplier term involving Ax can be rewritten using constant multipliers on Axx^T and an affine multiplier on $e_0^T x - 1$. The same argument generalizes for the degree-$2D$ relaxation.

Matching coefficients (6.16) will result in the conditions

$$\sum_i a_i \otimes_{\text{sym}} P_i + e_0 \otimes_{\text{sym}} S_0 = 0 \tag{6.17a}$$

$$\sum_i b_i P_i + \frac{1}{2}(a_0 e_0^T + e_0 a_0^T) - S_0 = \hat{P} \tag{6.17b}$$

$$a_0 = 0,\ b_0 = 0. \tag{6.17c}$$

where (6.17a) collects the degree three terms of (6.16), (6.17b) the quadratic terms, and (6.17c) the linear terms. We recall the notation $\otimes_{\text{sym}}$ from Section 1.4.1 denoting symmetric tensor product.

Simplification of (6.17) results in the condition that $x^T \hat{P} x$ is provably redundant for (6.15) in degree *at most* three if

$$\hat{P} = \sum_i b_i P_i - S_0 \tag{6.18a}$$

$$\sum_i a_i \otimes_{\text{sym}} P_i + e_0 \otimes_{\text{sym}} S_0 = 0. \tag{6.18b}$$

While these are all redundant, by Corollary 1 not all of them actually add strength. There are two cases. The first is that $\hat{P}$ genuinely encodes a quadratic. From Lemma 2 all quadratics $\hat{P}$ which we can express as

$$\begin{aligned} x^T \hat{P} x &= \sum_i x^T(\mu_i P_i)x + (\mu_0^T x)(e_0^T x - 1) \\ &= x^T \left(\sum_i \mu_i P_i + \frac{1}{2}(\mu_0 e_0^T + e_0 \mu_0^T) \right) x - \mu_0^T x \end{aligned} \tag{6.19}$$

$$\implies \mu_0 = 0,\ \hat{P} = \sum_i \mu_i P_i \tag{6.20}$$

$$\implies \hat{P} \in \mathbf{span}(P_i). \tag{6.21}$$

add no strength. Immediately, this shows that we can remove the $\sum_i b_i P_i$ component from (6.18a) as it cannot add strength.

The second case is that $\hat{P}$ is actually encoding a linear function. In this case, $\hat{P}$ adds no strength if (6.18) can be satisfied with $S_0 = 0$ and $a_i = 0$. This implies that every coefficient is zero, and so it cannot happen. This does not mean that no linear implied equalities exist. It simply means that none can be disqualified by Corollary 1.

The following example shows a linear equality that can be discovered in degree 3.

Example 14. *Let $R \in SO(2)$ be a rotation matrix. One way to parametrize the set of rotation matrices is that $R^T R = I$ and $\det(R) = 1$, the latter of which is a degree 2 polynomial for $SO(2)$. We consider checking whether this parametrization implies any linear equalities.*

Homogenizing with t and letting $R = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix}$, leads to

$$R^T R = t^2 I, \quad \textbf{\textit{det}}(R) = t^2 \\ t^2 = 1 \iff \begin{cases} h_1 \coloneqq r_{11}^2 + r_{21}^2 - t^2 = 0, \\ h_2 \coloneqq r_{12}^2 + r_{22}^2 - t^2 = 0, \\ h_3 \coloneqq r_{11} r_{12} + r_{21} r_{22} = 0, \\ h_4 \coloneqq r_{11} r_{22} - r_{12} r_{21} - t^2 = 0. \\ h_5 \coloneqq t^2 - 1 \end{cases}.$$

A search over degree 3 certificates shows that

$$r_{11} - r_{22} = r_{12} h_3 + r_{22} h_4 - r_{11} h_2 - (r_{11} - r_{22}) h_5, \\ r_{12} + r_{21} = r_{11} h_3 - r_{21} h_4 - r_{12} h_1 - (r_{12} + r_{21}) h_5.$$

Thus, the parametrization implies the linear equalities $r_{11} = r_{22}$ and $r_{21} = -r_{12}$. This is the familiar parametrization that

$$R = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}.$$

A simple linear algebra computation shows that a lower degree proof does not exist.

Summary: We summarize the key steps from this section. To check if an implied constraint $x^T \hat{P} x \in \langle x^T P_1 x, \ldots, x^T P_M x, e_0^T - 1 \rangle$, we first choose a proof degree; in this section we chose 3. Next, we set up a linear system matching the coefficients of $x^T \hat{P} x$ to the proof that it is in the ideal. This is (6.16).

By matching the coefficients on both sides, we are able to set up a linear system that parametrizes all possible $\hat{P}$ that we could add. This is (6.17). Parametrizing the set of $\hat{P}$ amounts to computing the nullspace of a subset of these equations, (6.17a), (6.17b), and (6.17c).

After recovering all possible $\hat{P}$, we note that by Corollary 1 some of these new equalities will not actually add strength. Therefore, we reduce the set of $\hat{P}$ by projecting away this useless part.

We formalize these steps to search for new $\hat{P}$ using higher degree proofs next.

6.3.3 High Degree Search

Any degree-$\hat{D}$ proof that $x^T \hat{P} x$ lies in the ideal $\langle x^T P_1 x, \ldots, x^T P_M x, e_0^T x - 1 \rangle$ has the form

$$x^T \hat{P} x = \sum_{i=1}^{M} \mu_i(x) \left(x^T P_i x \right) + \mu_0(x) \left(e_0^T x - 1 \right), \tag{6.22}$$

with $\mu_i \in \mathbb{R}[x]_{\leq \hat{D}-2}$, $\mu_0 \in \mathbb{R}[x]_{\leq \hat{D}-1}$. Similar to (6.17), matching the terms of the same degree on the left and right-hand sides of Equation (6.22) gives a linear equation between the coefficients of $\mu_i(x)$ and $\hat{P}$. It also requires all terms which are not homogeneous of

degree 2 to vanish on the right-hand side, forcing the coefficients of $\mu_i(x)$ to satisfy certain linear equalities.

Because (6.22) is linear, we can encode it as a linear map:

$$H_2^{\langle P_i \rangle}(u) = \hat{P} \tag{6.23a}$$

$$N_{\hat{D}}^{\langle P_i \rangle}(u) = 0. \tag{6.23b}$$

In (6.23), u stacks all the coefficients of the multiplier μ_i, $H_2^{\langle P_i \rangle}$ is a linear map keeping the degree 2 part of the right-hand side of (6.22), and $N_{\hat{D}}^{\langle P_i \rangle}$ is the linear map to all other terms. The maps $H_2^{\langle P_i \rangle}$ and $N_{\hat{D}}^{\langle P_i \rangle}$ depend on the coefficients of the matrices P_i and the vector e_0. $N_{\hat{D}}^{\langle P_i \rangle}$ also depends on the chosen $\hat{D}$. The size of the linear operator in (6.23) depends on the choice of the proof degree $\hat{D}$ and is at most $\binom{n+\hat{D}}{n} \times \left(M\binom{n+\hat{D}-2}{n} + \binom{n+\hat{D}-1}{n}\right)$.

This conversion from a polynomial equality to a system of linear equations is standard and is how a SOS program is converted into a standard form SDP. Mapping a basis for the solutions to (6.23b) through (6.23a) constructs a spanning set for all redundant equalities that could be added to (6.15). We denote the space of implied quadratic equalities proven in degree $\hat{D}$ as

$$E_{\hat{D}}^{\langle P_i \rangle} \coloneqq H_2^{\langle P_i \rangle}\left(\ker(N_{\hat{D}}^{\langle P_i \rangle})\right). \tag{6.24}$$

This completes the generation part. It requires computing the nullspace of $N_{\hat{D}}^{\langle P_i \rangle}$ and then mapping the nullspace through $H_2^{\langle P_i \rangle}$. The first step can be computed in a variety of standard ways including using a QR or SVD factorization.

6.3.3.1 Removing the Redundant Equalities

Once again, the parametrization in (6.24) may contain many polynomials which do not strengthen our degree 2 SOS relaxation. These useless redundant constraints are parametrized in (6.21).

To remove these from the subspace $E_{\hat{D}}^{\langle P_i \rangle}$, we compute the quotient

$$U_{\hat{D}}^{\langle P_i \rangle} = E_{\hat{D}}^{\langle P_i \rangle} / \mathbf{span}\{P_1, \ldots, P_M\}. \tag{6.25}$$

This can be done in several ways. Given a basis of $E_{\hat{D}}^{\langle P_i \rangle}$, we can project the vectors onto the orthogonal complement of $\mathbf{span}\{P_1, \ldots, P_M\}$. Alternatively, given a basis of $\mathbf{span}\{P_1, \ldots, P_M\}$, we can complete it to a basis of $E_{\hat{D}}^{\langle P_i \rangle}$. Generically, we should expect there to be relatively few implied equalities[3] and so we expect the space of implied equalities to be relatively small compared to the dimension of $E_{\hat{D}}^{\langle P_i \rangle}$. When this is the case, the latter procedure can be much more efficient.

Finally, we emphasize computation of the space $U_{\hat{D}}^{\langle P_i \rangle}$ can still be achieved with basic linear algebra regardless of the degree $\hat{D}$. Moreover, the scaling as we increase $\hat{D}$ is polynomial in a fixed number of variables n. In our experiments, we show that we can allow $\hat{D}$ to increase to much higher degree that the SOS degree can be set to.

[3]This is true when there are fewer equations than variables [196, §17.1] and conjectured by the Fröberg's Conjecture when there are more equations than variables [197].

Proof Deg.	dim (6.23b)	dim (6.25)	New constraints	t (s)
3	60	3	$R^T w = u$	0.002
4	457	4	$\|w\|^2 = \|u\|^2$	0.013
5	3040	4	Same as degree 4	0.087
6	16512	9	$RR^T = I$	0.647

Table 6.1: Quadratic implied equalities recovered for $Ru = w$ and $R^T R = I$. This encodes $R \in O(3)$.

Proof Deg.	dim (6.23b)	dim (6.25)	New constraints	time (s)
3	68	8	$R^T w = u$ $\tilde{r}_1^T \tilde{r}_2 = 0, \;\; \tilde{r}_2^T \tilde{r}_3 = 0$ $\tilde{r}_1 \times u = w_2 \tilde{r}_3 - w_3 \tilde{r}_2$	0.003
4	520	16	$\|w\|^2 = \|u\|^2$ $\tilde{r}_1^T r_1 = 1$ $r_2 = r_3 \times r_1, \;\; r_3 = r_1 \times r_2$	0.026
5	3805	24	$\tilde{r}_2^T \tilde{r}_2 = 1, \;\; \tilde{r}_2^T \tilde{r}_3 = 0$ $\tilde{r}_2 \times u = w_3 \tilde{r}_1 - w_1 \tilde{r}_3$ $\tilde{r}_3 \times u = w_1 \tilde{r}_2 - w_2 \tilde{r}_1$	0.16
6	22151	24	Same as degree 5	1.337

Table 6.2: Quadratic implied equalities recovered for $Ru = w$, $R^T R = I$, and $r_1 = r_2 \times r_3$. This encodes $R \in SO(3)$ using a quadratic cross-product representation.

6.4 Results

We consider the computation of implied equalities for various ideals. All experiments are run on an 11th Gen Intel Core i9-11980HK CPU with 64 GiB RAM using `Julia`. For all examples, we report the dimension of the nullspace in (6.23b), the dimension of (6.25), the time required to compute the new candidate equalities, and which new constraints are found by the procedure at a given degree. The dimension of (6.25) will be the cumulative sum of the dimension of the new equations in the rows preceding it.

6.4.1 Ideal of Rotated Vectors

In this section, we consider the ideal from Example 10 in dimension 3, where vectors u and w are related by a matrix R. We consider two cases.

1. We only constrain $R \in O(3)$ by using the constraint $R^T R = I$. The results are presented in Table 6.1.

2. We constrain R to be a rotation $R \in SO(3)$. We use the quadratic parametrization $R^T R = I$ and $r_1 = r_2 \times r_3$, where r_i is a column of R and $\times$ is the cross product. The results are presented in Table 6.2.

We denote the columns of R as r_i and the rows as $\tilde{r}_i$.

We note that in both cases, the computation times are negligible. Moreover, it is interesting to track the evolution of proving that $RR^T = I$ in the case of $O(3)$ versus $SO(3)$. In the $O(3)$ case, the quadratic equality $RR^T = I$ is not proven until degree 6. However, in the $SO(3)$ case, we are able to prove that rows $\tilde{r}_i^T \tilde{r}_j = 0$ at degree 3 and the full relation $RR^T = I$ is proven at degree 4.

Additionally, we note that at degree 6 for $O(3)$ and degree 5 for $SO(3)$, our procedure has already found a basis for all possible implied constraints provable in any degree. Though this cannot be proven by the proposed method, it can be certified by computing a Gröbner basis of the considered ideals.

6.4.2 Cloned-Quaternion Pose Estimation

Proof Deg.	dim (6.23b)	dim (6.25)	New constraints	time (s)
3	21	0	None	0.001
4	394	18	$x_0x_1^T = x_1x_0^T$ $x_0x_2^T = x_2x_0^T$ $x_1x_2^T = x_2x_1^T$	0.009
5	3770	18	Same as degree 4	0.084

Table 6.3: Quadratic implied equalities recovered for Example 11.

We consider the ideal from Example 11, namely

$$x_0^T x_0 = 1, \quad x_i x_i^T = x_0 x_0^T,$$

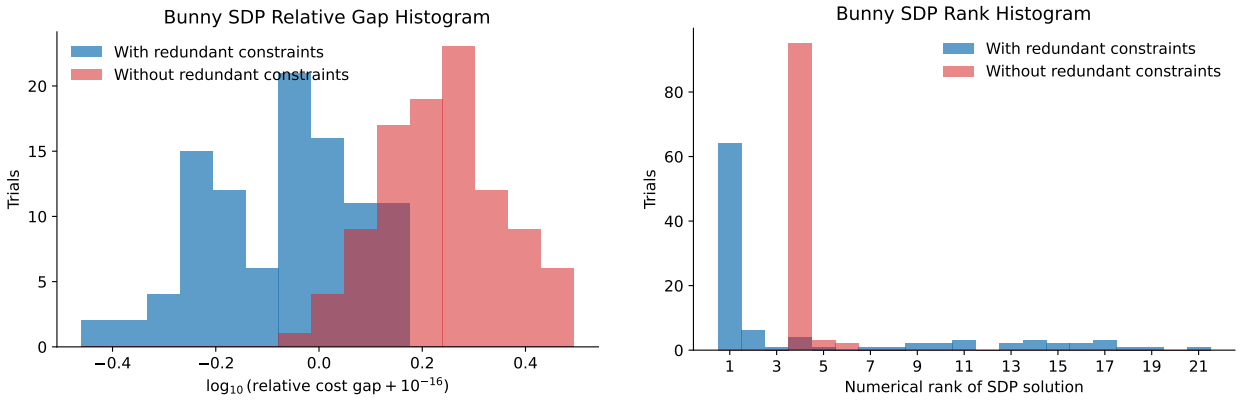
with $i \in [2]$, where each $x_i \in \mathbb{R}^4$. We report the results of running our procedure in Table 6.3.

Our procedure recovers the implied constraints of [32,191] at degree 4. A Gröbner basis computation confirms that these are all quadratic equalities obtainable in this example.

To verify the importance of adding these constraints, we run a small ablation on robust registration instances generated from the Stanford bunny point cloud.

Each trial samples 20 correspondences, applies a random rotation, adds noise with bound 0.01, and replaces 20% of the correspondences with outliers. We use `TEASER++` to export the rotation-stage program from their procedure and a reference candidate solution, then solve the primal SDP relaxation associated with [32, Eq. 23] both with and without the implied equalities $x_i x_j^T = x_j x_i^T$.

The results of the ablation are plotted in Figure 6.1 and confirm that the algebraically generated equalities strengthen the relaxation in the intended way. Across 100 trials, adding the equalities reduces the median relative gap between the `TEASER++` candidate objective and the SDP optimum from 1.71 to 0.90. It also changes the rank profile of the SDP solution: with the equalities, 64 of 100 solutions are numerically rank one and the median rank is 1, while without the equalities the median rank is 4 with no solutions being rank 1. The tighter formulation has a modest additional cost when solved with the `Clarabel` implementation, with median solve time increasing from 5.7 s to 6.5 s.



(a) Relative objective gap.

(b) Numerical SDP rank.

Figure 6.1: Effect of the redundant equality $x_i x_j^T = x_j x_i^T$ on bunny-derived robust registration instances. The relative gap compares the TEASER++ candidate objective to the SDP optimum, so smaller values indicate a tighter relaxation.

6.4.3 Generating New Equalities: The General POP Case

In this section, we generalize the construction in Section 6.3 to constructing implied constraints for (6.1) more generally.

After choosing relaxation degree $2D$, we select a maximum ideal membership degree $\hat{D}$. To construct our implied equalities, we proceed in a bottom up fashion. For proof degree $K \leq \hat{D}$ we define the linear map

$$\mathcal{L}_K : \bigoplus_{i=1}^{M} \mathbb{R}[x]_{\leq K-d_i} \to \mathbb{R}[x]_{\leq K}, \quad (\mu_1, \ldots, \mu_M) \mapsto \sum_{i=1}^{M} \mu_i h_i, \tag{6.26}$$

where $d_i := \deg(h_i)$ and we take $\mathbb{R}[x]_{\leq K-d_i} = \{0\}$ if $K < d_i$. To define our analog of (6.23) in this more general setting, we define the map ϕ_r to denote projection onto the homogeneous degree-r part of a polynomial. We use $\phi_{\neq r}$ to denote the projection onto the non-degree r part. Every polynomial f can be written as $f = \phi_r(f) + \phi_{\neq r}(f)$.

A degree r homogeneous polynomial $\hat{h}$ has a degree-K proof of membership in the ideal if there exists multiplier coefficients u such that

$$H_{r,K}^{\langle h_i \rangle}(u) = \hat{h} \tag{6.27a}$$

$$N_{r,K}^{\langle h_i \rangle}(u) = 0, \tag{6.27b}$$

where

$$H_{r,K}^{\langle h_i \rangle} := \phi_r \circ \mathcal{L}_K, \quad N_{r,K}^{\langle h_i \rangle} := \phi_{\neq r} \circ \mathcal{L}_K.$$

The space of homogeneous degree-r equalities with membership proofs of degree at most K is therefore

$$E_{r,K}^{\langle h_i \rangle} := H_{r,K}^{\langle h_i \rangle} \left(\ker N_{r,K}^{\langle h_i \rangle} \right) \subseteq \mathbb{R}[x]_r.$$

This is exactly the same construction as in the QCQP case. There, $r = 2$, $H_{2,K}$ keeps the quadratic part, and $N_{2,K}$ enforces cancellation of the non-quadratic part. We emphasize that after choosing the bases, every map here can be realized as a matrix and every operation can be computed using standard tools from linear algebra.

It remains to remove the equalities that Corollary 1 says cannot help. For degree r, the useless subspace is precisely the part already provable in degree r:

$$E_{r,r}^{\langle h_i \rangle} = H_{r,r}^{\langle h_i \rangle} \left(\ker N_{r,r}^{\langle h_i \rangle} \right).$$

Thus the potentially useful new degree-r equalities proven by degree K are represented by the quotient

$$U_{r,K}^{\langle h_i \rangle} := E_{r,K}^{\langle h_i \rangle} / E_{r,r}^{\langle h_i \rangle}.$$

Computationally, one obtains a spanning set for $E_{r,K}^{\langle h_i \rangle}$ by computing a basis of $\ker N_{r,K}^{\langle h_i \rangle}$ and applying $H_{r,K}^{\langle h_i \rangle}$. One then projects this basis onto the orthogonal complement of $E_{r,r}^{\langle h_i \rangle}$, or equivalently completes a basis of $E_{r,r}^{\langle h_i \rangle}$ to a basis of $E_{r,K}^{\langle h_i \rangle}$.

Running this for every $r \leq 2D$, with a proof degree K_r chosen for each r, gives all candidate implied equalities that can strengthen the degree-$2D$ SOS relaxation

$$U_{\leq 2D}^{\langle h_i \rangle} := \bigoplus_{r=0}^{2D} U_{r,K_r}^{\langle h_i \rangle}.$$

Choosing $K_r = \hat{D}$ sets the proof degree to be the same value for every homogeneous part of degree r.

Our procedure remains efficient in the general polynomial case, still only requiring basic linear algebra operations. The main complication is the requirement to sequentially compute the series of subspace $U_{r,K}^{\langle h_i \rangle}$ rather than a single subspace.

6.5 Conclusion

Adding redundant equalities to a QCQP before taking the semidefinite relaxation has proven essential to the successful application of SOS programming in robotics. In this chapter, we have described a simple linear-algebraic procedure for rapidly and automatically generating these candidate redundant equalities. Importantly, our procedure avoids adding candidate equalities which provably cannot strengthen the relaxation, avoiding an unnecessary increase in program size. We show that our method successfully recovers known, stronger formulations of QCQPs from the robotics literature, is fast enough to serve as a simple preprocessing step before solving a semidefinite relaxation, and can be generalized to arbitrary polynomial programs.

Part III

Tailored Conic Solvers

Chapter 7

CCosmo: A General and Customizable First-Order Convex Solver

Convex optimization appears throughout robotics, but in markedly different forms: small, dense quadratic programs (QPs) arise in online whole-body control, inverse-kinematics controllers, and reduced-order MPC controllers [198–202]; moderately sized, sparse and dense quadratic and second-order cone programs (SOCPs) arise naturally in grasp selection and contact planning problems due to friction constraints [95,203,204]; large, structured, and sparse problems arise in the context of trajectory optimization [205–207], with complicated semidefinite constraints becoming increasingly popular in the localization, perception, and motion planning communities [34,95,188,190–192,208]. In this thesis, we have been particularly interested in convex programs of various scales arising in the context of relaxations of polynomial problems.

In many of these applications, a solver is part of a larger planning, control, or estimation loop that repeatedly solves closely related problems to moderate accuracy at high speeds. This setting favors first-order methods, which have low memory requirements, are naturally warm started, and often converge rapidly to the accuracy required in the outer loop. Moreover, the relative simplicity of first-order methods makes them easy to customize to different classes of instances, sizes, and hardware such as massive programs on compute clusters [23,111], tiny programs on embedded hardware [202], or batches of programs on GPUs [209]. One of our main motivations for such a solver will be the developments in Chapter 9, where we will use a convex optimization solver as a subroutine in a larger solver.

In this chapter, we introduce `CCosmo`, a family of convex optimization solvers based on the alternating direction method of multipliers (ADMM) [39,101]. `CCosmo` is a `C++` implementation of a standard algorithm described in [23,40] with a variety of heuristic enhancements drawn broadly from the literature. Its basic form can be seen as a `C++` implementation of `COSMO.jl`. We further implement two variants of the method targeting specific cases. The first targets a matrix standard form which will become relevant in our work in Chapter 9. The second is a `Cuda` implementation of `CCosmo` which targets solving large batches of very small problems in parallel.

7.1 Related Work

Convex optimization using first-order methods has seen an explosion of research interest in the past 10 years. The earliest, modern instance of a general convex solver based on ADMM was the `SCS` solver from [210]. Implemented in `C`, `SCS` is able to detect optimality, primal infeasibility, and dual infeasibility by solving a homogeneous, self-dual embedding (HSDE) [211] of the original convex optimization problem. Due to the favorable numerical properties of the HSDE [212] and the symmetric way in which the HSDE handles optimality and infeasibility, `SCS` demonstrates predictable and robust convergence across a range of difficult instances and was a catalyst for renewed interest in first-order methods. On the other hand, low-accuracy solutions of an HSDE can in general be very far from being feasible, much less optimal, points [163,211,213] and so a low-accuracy solution to the HSDE may not yield good solutions.

Following `SCS`, the `OSQP` solver of [23] is an ADMM solver for quadratic programs (QPs) written in `C`. Rather than solving an HSDE, `OSQP` runs ADMM directly on the primal formulation of the QP, converging to an optimal solution if one exists and detecting primal or dual infeasibility using properties of diverging iterates [214]. `OSQP` was widely adopted [69,215,216] and used for solving both large-scale programs [217] and small programs on embedded systems [218]. A solver based on similar ideas, but targeted at higher-accuracy solves and denser problems is the `ProxQP` solver of [219].

Building on the success of `OSQP`, [40] introduced `COSMO.jl`, a `Julia` solver which implements the same base algorithm as `OSQP`. `COSMO.jl` handles a broader range of cones. Similar to `OSQP` on QPs, `COSMO.jl` demonstrates competitive performance in attaining low-accuracy solutions to convex optimization problems compared to more established interior point methods. Moreover, `COSMO.jl` natively handles convex programs with quadratic costs, which at the time gave it superior performance on problems with this type of objective over conic solvers such as `Mosek` [85] and `SCS` [210], though `SCS` has since been extended to natively handle quadratic costs [220]. Lastly, [40] also demonstrated both the ease with which their method could be adapted to special types of cones and the substantial computational benefits of doing so. Our `CCosmo` solver is largely based on the ideas presented in `COSMO.jl`, with additional enhancements and specializations.

In the search of ever more performance, extensions and variations of the original ADMM algorithm have been proposed [104,221–223]. See [112] for a unifying viewpoint on many of these extensions. The recent `GeNIOS` solver of [224] incorporates many of these ideas, solving convex programs with second-order differentiable costs and general convex constraints via a sequence of inexact solves of the ADMM subproblems. Another recent popular variation of the ADMM algorithm is the primal-dual hybrid-gradient (PDHG) method, also known as the Chambolle-Pock method, introduced in [225–228]. As this method relies only on sparse matrix-vector products and projection onto cones, it has become a particularly popular method for solving massive-scale convex optimization problems on GPUs. This direction has resulted in several solvers targeting linear programs, such as `PDLP` [111], `cuPDLP.jl` [229], `cuPDLPx.jl` [230], and `cuPDLP-C` [231], as well as `PDCS` [232] for general convex programs.

In practice, first-order methods are known to be both extremely sensitive to their algorithmic hyperparameters [233] and also may exhibit poor convergence to higher accuracy solutions [101,104]. To circumvent these issues, researchers have incorporated a wide variety

of heuristic strategies for adaptively selecting hyperparameters [23,111] and various acceleration schemes from the fixed-point literature. For example, `SCS` and `COSMO.jl` both implement a safeguarded Anderson Acceleration scheme [220,234,235]. In the PDHG literature, accelerations based on restarts [106], Halpern iteration [108,110,236], and reflection/relaxation [103] are popular. `CCosmo` optionally allows users to use many of these proposed schemes in a single solver, giving users fine grained control of the underlying algorithm.

Finally, we note the recent attention on using accelerated hardware to solve convex optimization problems. These efforts have focused primarily on using GPUs to solve a single, massive instance [111,217,229]. In the robotics setting, we are frequently more interested in the ability to solve a large batch of very small to moderately sized programs. This direction has seen less attention, but is beginning to emerge as a research direction [237–239].

7.2 Problem Formulation

In this section, we formally introduce our problem formulation.

Let $\mathbb{V}$ and $\mathbb{W}$ be vector spaces. Let $x \in \mathbb{V}$, $\mathcal{A} : \mathbb{V} \to \mathbb{W}$ be a linear operator, $b \in \mathbb{W}$, and $\mathcal{K} \subseteq \mathbb{W}$ be a convex set. We let $q \in \mathbb{V}^*$ be a linear functional on $\mathbb{V}$ and $\mathcal{P} : \mathbb{V} \to \mathbb{R}_+$ be a positive semidefinite quadratic form.

`CCosmo` solves problems in the following form.

$$\begin{aligned} \min\ & \mathcal{P}(x) + \langle q, x \rangle && \text{(7.1a)} \\ \text{subject to}\ & \mathcal{A}(x) - b \in \mathcal{K} && \text{(7.1b)} \\ & \mathcal{C}(x) - d = 0, && \text{(7.1c)} \end{aligned}$$

where $\mathcal{K}$ is a closed convex set.

The dual of (7.1) is

$$\begin{aligned} \max\ & \langle \lambda_{\mathcal{K}}, b \rangle + \langle \lambda_0, d \rangle - \mathcal{P}(x) + \inf_{w \in \mathcal{K}} \langle \lambda_{\mathcal{K}}, w \rangle && \text{(7.2a)} \\ \text{subject to}\ & \nabla \mathcal{P}(x) = \mathcal{A}^*(\lambda_{\mathcal{K}}) + \mathcal{C}^*(\lambda_0) - q && \text{(7.2b)} \\ & \lambda_{\mathcal{K}} \in (\mathcal{K}^\infty)^*. && \text{(7.2c)} \end{aligned}$$

We remind the reader that $\mathcal{K}^\infty$ denotes the recession cone of $\mathcal{K}$ (see Definition 6) and $\mathcal{K}^*$ denotes the dual cone (see Definition 7). The term $\inf_{w \in \mathcal{K}} \langle \lambda_{\mathcal{K}}, w \rangle$ can alternatively be written as $-\sigma_{\mathcal{K}}(-\lambda_{\mathcal{K}})$ where σ is the support function of $\mathcal{K}$ (see Definition 8). A proof of the duality between (7.1) and (7.2) is deferred to Section 7.8.1.

The formulation in (7.1) provides a flexible description of convex optimization problems. We list two examples of interest.

Example 15 (Quadratic Convex Standard Form)**.** *Choosing $\mathbb{V} = \mathbb{R}^n$, $\mathbb{W} = \mathbb{R}^m$ and providing explicit matrix representations of $\mathcal{P}$, $\mathcal{A}$, and $\mathcal{C}$ results in the typical standard form described in* (QCSF)*.*

$$\begin{aligned} \min\ & \frac{1}{2} x^T P x + q^T x \\ \text{subject to}\ & Ax - b \in \mathcal{K} \\ & Cx - d = 0. \end{aligned} \tag{7.3}$$

This is the standard form input of many modern solvers including `SCS`, `COSMO.jl`, *and* `Clarabel` *[22].*

Example 16 (Matrix Standard Form)**.** *Choosing* $\mathbb{V} = \mathbb{R}^{n\times p}$ *and* $\mathbb{W} = \bigoplus_{i=1}^{N} \mathbb{R}^{m\times q}$ *results in the matrix standard form of* (7.1)*:*

$$\begin{aligned} \min \ & \frac{1}{2}\,\boldsymbol{tr}(X^T P_l X P_r) + \boldsymbol{tr}(Q^T X) \\ \text{subject to } & A_i X B_i - G_i \in \mathcal{K}_i,\ i \in [M]. \end{aligned} \tag{7.4}$$

where $\mathcal{K}_i$ *are convex sets.*

We take particular interest in the following convex linear programming specialization of (7.4)*.*

$$\begin{aligned} \min \ & \boldsymbol{tr}(Q^T X) \\ \text{subject to } & AXB - G \in \mathcal{K}_1 \\ & XC - F \in \mathcal{K}_2 \end{aligned} \tag{7.5}$$

which is a natural form for the vertex part of the GCS convex relaxation in Chapter 3.

Remark 10. *Since* $\mathcal{K}$ *is just assumed to be a convex set, in principle we can append all the constraints* $\mathcal{C}(x) = d$ *onto the constraint* $\mathcal{A}(x) - b \in \mathcal{K}$*. We will see that the algorithm described in Section 7.3 can handle constraints of the form* (7.1c) *directly. Absorbing* $\mathcal{C}(x) = d$ *as a generic convex set will result in a slightly different algorithm. Our solver supports either form; how* $\mathcal{C}(x) = d$ *is handled is a user-specified choice.*

7.2.1 Optimality and Infeasibility Conditions

Assuming strong duality, the optimality conditions of (7.1) are given by

$$\mathcal{A}(x) - b \in \mathcal{K},\ \mathcal{C}(x) - d = 0 \tag{7.6a}$$

$$\nabla\mathcal{P}(x) = \mathcal{A}^*(\lambda_{\mathcal{K}}) + \mathcal{C}^*(\lambda_0) - q,\ \lambda_{\mathcal{K}} \in (\mathcal{K}^\infty)^* \tag{7.6b}$$

$$\langle \lambda_{\mathcal{K}}, \mathcal{A}(x) - b\rangle = \inf_{w\in\mathcal{K}} \langle \lambda_{\mathcal{K}}, w\rangle. \tag{7.6c}$$

Equations (7.6a) and (7.6b) encode primal and dual feasibility respectively, while (7.6c) encodes complementary slackness and is frequently written as requiring that $-\lambda_{\mathcal{K}}$ be in the normal cone of $\mathcal{A}(x) - b$.

Our problem is dual unbounded if $\exists \lambda_{\mathcal{K}},\ \lambda_0$ such that

$$\mathcal{A}^*(\lambda_{\mathcal{K}}) + \mathcal{C}^*(\lambda_0) = 0,\ \lambda_{\mathcal{K}} \in (\mathcal{K}^\infty)^* \tag{7.7a}$$

$$\langle \lambda_{\mathcal{K}}, b\rangle + \langle \lambda_0, d\rangle + \inf_{w\in\mathcal{K}} \langle \lambda_{\mathcal{K}}, w\rangle > 0, \tag{7.7b}$$

and the problem is primal unbounded if $\exists x$ such that

$$\mathcal{A}(x) \in \mathcal{K}^\infty,\ \mathcal{C}(x) = 0 \tag{7.8a}$$

$$\mathcal{P}(x) = 0,\ \langle q, x\rangle < 0. \tag{7.8b}$$

Under appropriate regularity assumptions guaranteeing the corresponding strong alternatives, (7.7) is equivalent to primal infeasibility and (7.8) is equivalent to dual infeasibility. We call λ (resp. x) a dual (resp. primal) improving ray. These infeasibility conditions follow from e.g. [42, §5.8] or [44, §4.2 and §4.3].

7.3 General Solution Method

We now turn our attention to solving (7.1). This is done by applying the ADMM algorithm from Chapter 4 to a reformulation of (7.1). To employ Algorithm 1, we use the same splitting method as in `OSQP` [23] and `COSMO.jl` [40] to cast (7.1) into the form (4.1). First, we introduce slack variable s and consensus copies $(\tilde{x}, \tilde{s})$ and write (7.1) as

$$\begin{aligned} \min\ & \mathcal{P}(x) + \langle q, x \rangle && \text{(7.9a)} \\ \text{subject to}\ & \mathcal{A}(x) - b = s && \text{(7.9b)} \\ & \mathcal{C}(x) - d = 0 && \text{(7.9c)} \\ & (\tilde{x}, \tilde{s}) \in (\mathbb{V}, \mathcal{K}) && \text{(7.9d)} \\ & (x, s) = (\tilde{x}, \tilde{s}). && \text{(7.9e)} \end{aligned}$$

This corresponds to the following ADMM standard form.

$$\begin{aligned} \min\ & \mathcal{P}(x) + \langle q, x \rangle + \mathbb{1}(\mathcal{A}(x) - b = s) + \mathbb{1}(\mathcal{C}(x) = d) + \mathbb{1}((\tilde{x}, \tilde{s}) \in (\mathbb{V}, \mathcal{K})) && \text{(7.10a)} \\ \text{subject to}\ & (x, s) = (\tilde{x}, \tilde{s}), && \text{(7.10b)} \end{aligned}$$

where $\mathbb{1}(\cdot)$ is the indicator function that is 0 if the expression is satisfied and ∞ otherwise. In this case, the f and g are given by $f(x, s) = \mathcal{P}(x) + \langle q, x \rangle + \mathbb{1}(\mathcal{A}(x) - b = s) + \mathbb{1}(\mathcal{C}(x) = d)$ and $g(\tilde{x}, \tilde{s}) = \mathbb{1}((\tilde{x}, \tilde{s}) \in (\mathbb{V}, \mathcal{K}))$.

We apply Algorithm 1 to the augmented Lagrangian

$$\begin{aligned} \mathcal{L}_{\mathcal{D}_x, \mathcal{D}_s}((x, s), (\tilde{x}, \tilde{s}), u) = \mathcal{P}(x) + \langle q, x \rangle + \mathbb{1}(\mathcal{A}(x) - b = s) + \mathbb{1}(\mathcal{C}(x) = d) + \\ \mathbb{1}((\tilde{x}, \tilde{s}) \in (\mathbb{V}, \mathcal{K})) + \mathcal{D}_x(x - \tilde{x}) + \mathcal{D}_s(s - \tilde{s} + u) \end{aligned} \quad \text{(7.11)}$$

It can be shown that if $x^0 = \tilde{x}^0$, then no dual variable is needed for the constraint $x = \tilde{x}$ as it will always be zero.

In this form, we can directly apply Algorithm 1. Due to the particular split we have chosen, Line 1.3 and Line 1.4 can be written in closed-form. We describe how to efficiently implement Line 1.3 and Line 1.4 in the next sections.

7.3.1 Form of $\mathcal{D}_x$ and $\mathcal{D}_s$

We choose $\mathcal{D}_x$ and $\mathcal{D}_s$ to be compatible with the underlying vector-space structure. For problems of the form (7.3), we use

$$\mathcal{D}(z) = \frac{1}{2} z^T D z, \quad D \succ 0. \tag{7.12}$$

For problems of the form (7.4), we use

$$\mathcal{D}(Z) = \frac{1}{2} \mathbf{tr}(Z^T D_l Z D_r), \quad D_l, D_r \succ 0. \tag{7.13}$$

7.3.2 Affine Subspace Projection

Line 1.3 for (7.10) amounts to solving the following equality-constrained QP:

$$\min \mathcal{P}(x) + \langle q, x\rangle + \mathcal{D}_x(x - \tilde{x}^k) + \mathcal{D}_s(s - \tilde{s}^k + u^k) \tag{7.14}$$

$$\text{subject to } \mathcal{A}(x) - b = s \tag{7.15}$$

$$\mathcal{C}(x) = d. \tag{7.16}$$

The KKT system of (7.14) is given by

$$\begin{bmatrix} \nabla\mathcal{D}_s & 0 & -I & 0 \\ 0 & \nabla\mathcal{P} + \nabla\mathcal{D}_x & \mathcal{A}^* & \mathcal{C}^* \\ -I & \mathcal{A} & 0 & 0 \\ 0 & \mathcal{C} & 0 & 0 \end{bmatrix} \begin{bmatrix} s \\ x \\ \omega_{\mathcal{K}} \\ \omega_0 \end{bmatrix} = \begin{bmatrix} \nabla\mathcal{D}_s(\tilde{s}^k - u^k) \\ \nabla\mathcal{D}_x(\tilde{x}^k) - q \\ b \\ d \end{bmatrix}, \tag{7.17}$$

and so performing Line 1.3 amounts to solving (7.17). Solving (7.17) is a standard problem and an extremely common subroutine in many optimization algorithms; see [240, §16.2] and [241]. Depending on the structure of the data, we solve (7.17) after eliminating s or eliminating all of s, $\omega_{\mathcal{K}}$, and ω_0.

Eliminating s yields the system

$$\begin{bmatrix} \nabla\mathcal{P} + \nabla\mathcal{D}_x & \mathcal{A}^* & \mathcal{C}^* \\ \mathcal{A} & -(\nabla\mathcal{D}_s)^{-1} & 0 \\ \mathcal{C} & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \omega_{\mathcal{K}} \\ \omega_0 \end{bmatrix} = \begin{bmatrix} \nabla\mathcal{D}_x(\tilde{x}^k) - q \\ b + \tilde{s}^k - u^k \\ d \end{bmatrix}. \tag{7.18}$$

When $\mathcal{C}$ is surjective, then (7.18) is quasidefinite [242] and so admits a unique solution [240, §16.2].

If $\mathcal{C}$ is not surjective, then the perturbed systems

$$\begin{bmatrix} \nabla\mathcal{P} + \nabla\mathcal{D}_x & \mathcal{A}^* & \mathcal{C}^* \\ \mathcal{A} & -(\nabla\mathcal{D}_s)^{-1} & 0 \\ \mathcal{C} & 0 & -\delta I \end{bmatrix} \begin{bmatrix} x \\ \omega_{\mathcal{K}} \\ \omega_0 \end{bmatrix} = \begin{bmatrix} \nabla\mathcal{D}_x(\tilde{x}^k) - q \\ b + \tilde{s}^k - u^k \\ d \end{bmatrix} \tag{7.19}$$

is quasidefinite for every $\delta > 0$ and admits a unique solution. We choose $\delta \approx \sqrt{\epsilon_{mach}}$ where ϵ_{mach} is the floating-point precision[1] [12] and solve (7.19) using iterative refinement [243, §3.37].

Elimination of $\omega_{\mathcal{K}}$ and ω_0 from (7.19) yields the system

$$\left(\nabla\mathcal{P} + \nabla\mathcal{D}_x + \mathcal{A}^*(\nabla\mathcal{D}_s)\mathcal{A} + \delta^{-1}\mathcal{C}^*\mathcal{C}\right) x = \left[\nabla\mathcal{D}_x(\tilde{x}^k) - q + \mathcal{A}^*(\nabla\mathcal{D}_s)(b + \tilde{s}^k - u^k) + \delta^{-1}\mathcal{C}^* d\right]. \tag{7.20}$$

In this case, (7.20) is positive definite.

Solving either (7.19) or (7.20) is sufficient as we can recover $s = \mathcal{A}(x) - b$, and the dual variables if needed from (7.17). This solve is the most expensive part of Algorithm 1 (unless $\mathcal{K}$ contains large PSD cones). Both can be solved using Krylov subspace iterative methods

[1] for double precision $\epsilon_{mach} = 10^{-16}$ and for float precision $\epsilon_{mach} = 10^{-8}$

such as GMINRES [244] for (7.19) and Conjugate Gradients [245] or MINRES [246] for (7.20).

By assuming a concrete form to (7.1), we can also use a direct factorization approach to solve (7.19) and (7.20). When $\mathcal{D}_x$ and $\mathcal{D}_s$ do not change between iterations, the left-hand sides of (7.19) and (7.20) do not change. Therefore, their factorizations can be cached between iterations, substantially speeding up the iteration and making the solve times very predictable compared to using an iterative method. This is in contrast to interior point methods which solve a similar system, yet must recompute the factorization at every step [22,85].

We next outline the solution strategies we consider for problems of the form (7.3) and (7.4).

7.3.2.1 Convex Standard Form

When solving an instance of (7.1) given in the form (7.3), with penalty parameter functions chosen as (7.12), then (7.19) takes the form

$$\begin{bmatrix} P + D_x & A^T & C^T \\ A & -D_s^{-1} & 0 \\ C & 0 & -\delta I \end{bmatrix} \begin{bmatrix} x \\ \omega_{\mathcal{K}} \\ \omega_0 \end{bmatrix} = \begin{bmatrix} D_x \tilde{x}^k - q \\ b + \tilde{s}^k - u^k \\ d \end{bmatrix}, \tag{7.21}$$

while (7.20) takes the form

$$\left(P + D_x + A^T D_s A + \delta^{-1} C^T C\right) x = D_x \tilde{x}^k - q + A^T D_s (b + \tilde{s}^k - u^k) + \delta^{-1} C^* d. \tag{7.22}$$

When (7.3) is given as sparse data, we solve (7.21) rather than (7.22) as the latter results in a much denser system. To solve (7.21), we perform a sparse LDL^T factorization after applying a fill-reducing permutation, offering the option to use either `QDLDL`[2] [23] or `SSIDS`[3] [247,248].

If the problem data is dense, we solve (7.22) since it is the smallest linear system. We use a dense pivoted Cholesky factorization to factor the left-hand side.

7.3.2.2 Matrix Standard Form

When solving instantiations of (7.1) in the form (7.4), then (7.18) takes the form

$$\begin{aligned} P_l X P_r + D_{lx} X D_{rx} \quad &+ \sum_{i=1}^{M} A_i^T \Lambda_i B_i^T \quad &&= D_{lx} \tilde{X}^k D_{rx} - Q \\ A_i X B_i \quad &- D_{lS_i}^{-1} \Lambda_i D_{rS_i}^{-1} \quad &&= G_i + \tilde{S}_i^k - U_i^k, \quad i \in [M]. \end{aligned} \tag{7.23}$$

[2] https://github.com/osqp/qdldl

[3] https://ralna.github.io/spral/_build/html/C/ssids.html

while (7.20) results in

$$P_l X P_r + D_{lx} X D_{rx} + \sum_{i=1}^{M} A_i^T D_{lS_i} A_i X B_i D_{lS_i} B_i^T = \\ D_{lx} \tilde{X}^k D_{rx} - Q + \sum_{i=1}^{M} A_i^T D_{lS_i} \left(G_i + \tilde{S}_i^k - U_i^k \right) D_{rS_i} B_i^T. \quad (7.24)$$

Both (7.23) and (7.24) are highly structured matrix equations. In principle, both can be solved using Kronecker vectorization (see Theorem 1), but the memory cost of forming these matrices can be prohibitive even on a modern computer.

The controls community has historically developed methods for solving Lyapunov and Sylvester equations of a similar form and there has been a modern resurgence of interest. See the recent survey by [249]. In the case of multiple terms $A_i X B_i - G_i \in \mathcal{K}_i$, direct factorization methods exploiting the matrix structure of (7.23) and (7.24) seem difficult to obtain. Therefore, most research has focused on developing more effective iterative methods [250–254].

If we have a problem of the form (7.5), then (7.24) takes the form

$$D_{lX} X \begin{bmatrix} D_{rX}^{1/2} & C^T D_{lS_0}^{1/2} \end{bmatrix} \begin{bmatrix} D_{rX}^{1/2} \\ D_{lS_0}^{1/2} C \end{bmatrix} + A^T D_{lS_\mathcal{K}} A X B D_{lX} B^T = \\ D_{lX} \tilde{X}^k D_{rX} - Q + A^T D_{lS_\mathcal{K}} \left(G + \tilde{S}_\mathcal{K}^k - U_\mathcal{K}^k \right) D_{rS_\mathcal{K}} B^T + \\ D_{lS_\mathcal{K}} \left(F + \tilde{S}_0^k - U_0^k \right) D_{rS_0} C^T \quad (7.25)$$

A very efficient, direct factorization method for solving (7.25) is described in Section 8.1.

In the current implementation, we assume the data matrices P, Q, A_i, B_i, G_i are all dense and solve (7.24). In the case of many matrix convex constraints, we use the method of [254] to solve (7.24). When there is only one convex constraint, or the special form given in (7.5), we use our method from Section 8.1 to solve (7.25).

7.3.3 Cone Projection

Line 1.4 partitions into two subproblems:

$$\min \mathcal{D}_x(\hat{x}^{k+1} - \tilde{x}). \quad (7.26)$$

where $\hat{x}^{k+1} = \alpha x^{k+1} + (1-\alpha)\tilde{x}^k$ and

$$\begin{aligned} &\min \mathcal{D}_s(\hat{s}^{k+1} - \tilde{s} + u^k) \\ &\text{subject to } \tilde{s} \in \mathcal{K}, \end{aligned} \quad (7.27)$$

where $\hat{s}^{k+1} = \alpha s^{k+1} + (1-\alpha)\tilde{s}^k$.

The solution to (7.26) is trivially $\tilde{x} = \hat{x}^{k+1}$. The problem in (7.27) is the weighted projection of $\hat{s}^{k+1} + u^k$ onto a convex set.

In practice, the set $\mathcal{K}$ is given in a factored form $\mathcal{K} = \mathcal{K}_1 \times \cdots \times \mathcal{K}_N$ and so we assume that the quadratic penalty $\mathcal{D}_s$ partitions according to this product structure

$$\mathcal{D}_s(s) = \sum_{i=1}^{N} \mathcal{D}_{s_i}(s_i),$$

allowing (7.27) to also partition per set.

Now (7.27) corresponds to weighted projection onto the convex set $\mathcal{K}_i$. If $\mathcal{D}_{s_i}$ is a scaled multiple of the identity, then this reduces to Euclidean projection onto the convex set $\mathcal{K}_i$:

$$\tilde{s}_i = \Pi_{\mathcal{K}_i}\left(\hat{s}^{k+1} + u^k\right).$$

Many sets admit efficient or closed-form solutions in this case such as "linear" sets (i.e. bounding boxes, the positive orthant, and the zero set), the Lorentz cone, and the PSD cone.

More generally, if

$$\mathcal{D}_{s_i} = \rho_i^2 \mathcal{T}_i^* \mathcal{T}_i \tag{7.28}$$

with $\rho_i > 0$ a scaling parameter and $\mathcal{T}_i$ an automorphism of $\mathcal{K}_i$, (7.27) admits the solution

$$\tilde{s}_i = \mathcal{T}_i^{-1} \Pi_{\mathcal{K}_i}\left(\mathcal{T}_i\left(\hat{s}^{k+1} + u^k\right)\right). \tag{7.29}$$

This is proven in Section 7.8.2.

The linear sets also admit closed-form solutions under anisotropic, coordinate-wise scalings (i.e. diagonal $\mathcal{D}_s$). The Lorentz cone admits efficient iterative solutions to problems of the form (7.27) with general $\mathcal{D}_{s_i}$ [232,255].

To ensure that Line 1.4 has an efficient solution, we will assume that $\mathcal{D}_s$ has the form prescribed by (7.28), unless it is a linear set in which case we allow for anisotropic diagonal scalings.

In Appendix B.1, we describe the sets currently supported by `CCosmo`, their projection operators, and their automorphism groups.

7.3.4 Termination Criterion

We apply Algorithm 1 to (7.10) until our iterates satisfy one of the following conditions. To terminate, we recover our dual variables as

$$\lambda_{\mathcal{K}} = -\nabla \mathcal{D}_s(u^k) \qquad \lambda_0 = -\omega_0^k \tag{7.30}$$

where u^k is the current ADMM dual iterate and ω_0^k is the equality-constraint multiplier from the most recent solve of (7.17).

7.3.4.1 Optimality

We define our primal, dual, and gap residuals as

$$r_p = \mathcal{A}(\tilde{x}) - b - \tilde{s}, \quad r_d = \nabla \mathcal{P}(\tilde{x}) + q - \mathcal{A}^*(\lambda_{\mathcal{K}}) - \mathcal{C}^*(\lambda_0), \quad r_g = \langle \lambda_{\mathcal{K}}, \tilde{s} \rangle - \inf_{w \in \mathcal{K}} \langle \lambda_{\mathcal{K}}, w \rangle.$$

Note that $Cx - d = 0$ and $C\tilde{x} - d = 0$ by construction at every iteration.

After choosing thresholds ϵ_{abs} and ϵ_{rel}, we terminate at optimality if all three conditions

$$\|r_p\|_\infty \leq \epsilon_{abs} + \epsilon_{rel} \max\left\{\|\mathcal{A}(\tilde{x})\|_\infty, \|b\|_\infty, \|\tilde{s}\|_\infty\right\}, \tag{7.31a}$$
$$\|r_d\|_\infty \leq \epsilon_{abs} + \epsilon_{rel} \max\left\{\|\nabla\mathcal{P}(\tilde{x})\|_\infty, \|q\|_\infty, \|\mathcal{A}^*(\lambda_\mathcal{K})\|_\infty, \|\mathcal{C}^*(\lambda_0)\|_\infty\right\}, \tag{7.31b}$$
$$\|r_g\|_\infty \leq \epsilon_{abs} + \epsilon_{rel} \max\left\{\|\langle\lambda_\mathcal{K}, \tilde{s}\rangle\|_\infty, \left\|\inf_{w\in\mathcal{K}}\langle\lambda_\mathcal{K}, w\rangle\right\|_\infty\right\}. \tag{7.31c}$$

7.3.4.2 Infeasibility

By [214], if (7.1) is primal infeasible, then

$$\delta\lambda_\mathcal{K}^k := \lambda_\mathcal{K}^k - \lambda_\mathcal{K}^{k-1}, \qquad \delta\lambda_0^k := \lambda_0^k - \lambda_0^{k-1} \tag{7.32}$$

converge to a certificate of primal infeasibility satisfying (7.7). If (7.1) is dual infeasible, then

$$\delta x^k := \tilde{x}^k - \tilde{x}^{k-1}$$

converges to a certificate of dual infeasibility satisfying (7.8).

After choosing tolerances $\epsilon_{pinf} > 0$ and $\epsilon_{dinf} > 0$, we declare primal infeasibility if

$$\left\|\mathcal{A}^*(\delta\lambda_\mathcal{K}^k) + \mathcal{C}^*(\delta\lambda_0^k)\right\|_\infty \leq \epsilon_{pinf}, \tag{7.33a}$$
$$\left\|\delta\lambda_\mathcal{K}^k - \Pi_{(\mathcal{K}^\infty)^*}(\delta\lambda_\mathcal{K}^k)\right\|_\infty \leq \epsilon_{pinf}, \tag{7.33b}$$
$$\langle\delta\lambda_\mathcal{K}^k, b\rangle + \langle\delta\lambda_0^k, d\rangle + \inf_{w\in\mathcal{K}}\langle\delta\lambda_\mathcal{K}^k, w\rangle \geq \epsilon_{pinf}. \tag{7.33c}$$

Similarly, we declare dual infeasibility if

$$\left\|\nabla\mathcal{P}(\delta x^k)\right\|_\infty \leq \epsilon_{dinf}, \tag{7.34a}$$
$$\left\|\mathcal{A}(\delta x^k) - \Pi_{\mathcal{K}^\infty}(\mathcal{A}(\delta x^k))\right\|_\infty \leq \epsilon_{dinf}, \tag{7.34b}$$
$$\left\|\mathcal{C}(\delta x^k)\right\|_\infty \leq \epsilon_{dinf}, \tag{7.34c}$$
$$\langle q, \delta x^k\rangle \leq -\epsilon_{dinf}. \tag{7.34d}$$

7.3.5 Summary of the Base Algorithm

We summarize the specialization of Algorithm 1 to the problem (7.1) in Algorithm 5. We choose $\mathcal{D}_x = 10^{-6}I$ because its role is to regularize the solution of (7.17). Given the product of sets $\mathcal{K} = \mathcal{K}_1 \times \cdots \times \mathcal{K}_N$, we choose per-set penalty parameter ρ_i and automorphisms $\mathcal{T}_i$ and compose $\mathcal{D}_s = \oplus_{i=1}^N \rho_i \mathcal{T}_i^* \mathcal{T}_i$. We discuss how ρ_i and $\mathcal{T}_i$ are chosen in Section 7.4.2.

Using the previous subsections, the specialization of Algorithm 1 to the `CCosmo` split (7.10) can be written explicitly as Algorithm 5. The affine-subspace step in Line 5.3 is implemented by solving one of the linear systems (7.19) or (7.20), and then recovering the equality multiplier ω_0. The cone projection step in Line 5.5 is evaluated blockwise over the product decomposition $\mathcal{K} = \mathcal{K}_1 \times \cdots \times \mathcal{K}_N$.

Algorithm 5: Base ADMM algorithm specialized to (7.1)

Input: Penalty functions $\mathcal{D}_x$, set automorphisms $\mathcal{T}_i$, initial penalties ρ_i, so that $\mathcal{D}_s = \oplus_{i=1}^{N}(\rho_i \mathcal{T}_i^* \mathcal{T}_i)$, relaxation parameter $\alpha \in (0, 2)$, initial iterates $(\tilde{x}^0, \tilde{s}^0, u^0)$ with $\tilde{s}^0 \in \mathcal{K}$, optimality termination tolerances ϵ_{abs} and ϵ_{rel}, and primal and dual infeasibility tolerances ϵ_{pinf} and ϵ_{dinf}.

Output: Either an approximate primal-dual solution or a certificate of infeasibility

5.1 $k \leftarrow 0$
5.2 **while** *no termination criterion has fired* **do**
5.3 Solve (7.14) for (x^{k+1}, s^{k+1}), and recover ω_0^{k+1}
5.4 $\tilde{x}^{k+1} \leftarrow \alpha x^{k+1} + (1-\alpha)\tilde{x}^k$ and $\delta x^{k+1} \leftarrow \delta x^{k+1} - \delta x^k$
5.5 $\tilde{s}^{k+1} \leftarrow \mathcal{T}_{\mathcal{K}}^{-1} \Pi_{\mathcal{K}} \mathcal{T}_{\mathcal{K}}(\alpha s^{k+1} + (1-\alpha)\tilde{s}^k + u^k)$
5.6 $u^{k+1} \leftarrow u^k + \alpha s^{k+1} + (1-\alpha)\tilde{s}^k - \tilde{s}^{k+1}$
5.7 $\lambda_{\mathcal{K}}^{k+1} \leftarrow -\nabla \mathcal{D}_s(u^{k+1})$ and $\delta\lambda_{\mathcal{K}}^{k+1} \leftarrow \lambda_{\mathcal{K}}^{k+1} - \lambda_{\mathcal{K}}^k$
5.8 $\lambda_0^{k+1} \leftarrow -\omega_0^{k+1}$ and $\delta\lambda_0^{k+1} \leftarrow \lambda_0^{k+1} - \lambda_0^k$
5.9 $k \leftarrow k+1$
5.10 **if** (7.31) **then**
5.11 **return** *optimal* $(\tilde{x}^{k+1}, \tilde{s}^{k+1}, \lambda_{\mathcal{K}}^{k+1}, \lambda_0^{k+1})$
5.12 **else if** (7.33) **then**
5.13 **return** *Primal infeasible certificate* $(\lambda_{\mathcal{K}}^{k+1}, \lambda_0^{k+1})$
5.14 **else if** (7.34) **then**
5.15 **return** *Dual infeasible certificate* $(\tilde{x}^{k+1}, \tilde{s}^{k+1})$

7.4 Enhancements

First-order methods are known to exhibit slow convergence to high-accuracy solutions and to be very sensitive to their parameters and problem data. It is therefore standard to augment a first-order method in a variety of ways. We describe the enhancements implemented in `CCosmo` here.

7.4.1 Using Acceleration

The restarted anchoring schemes from Algorithm 2 apply directly to the fixed-point map induced by Algorithm 5. In our setting, Algorithm 1 only converges if (7.1) has an optimal solution and so the accelerated convergence rates of [103] only apply in the case where (7.1) is feasible. However, it has also been shown that Halpern iteration on Algorithm 1 achieves accelerated convergence to certificates of infeasibility for (7.1) as well and it is likely that such results can be extended to infeasibility detection for the scheme from [103].

For the `CCosmo` split, we take the ADMM state to be

$$w^k := (x^k, s^k, \tilde{x}^k, \tilde{s}^k, u^k),$$

and apply (4.7) to this state. When applying acceleration, we choose by default $\alpha = 1.0$, $\gamma = 1.8$, and $\beta = 15$.

7.4.1.1 Restarts

When restarts are enabled, we restart any time we see sufficient decrease in the fixed-point penalty. Concretely, let

$$r_{\mathrm{p}}^{t,k} = \mathcal{D}_x^t(x^{t,k+1} - \tilde{x}^{t,k+1}) + \mathcal{D}_s^t(s^{t,k+1} - \tilde{s}^{t,k+1}), \tag{7.35}$$

$$r_{\mathrm{d}}^{t,k} = \mathcal{D}_x^t(\tilde{x}^{t,k+1} - \tilde{x}^{t,k}) + \mathcal{D}_s^t(\tilde{s}^{t,k+1} - \tilde{s}^{t,k}), \tag{7.36}$$

$$r^{t,k} = r_{\mathrm{p}}^{t,k} + r_{\mathrm{d}}^{t,k}. \tag{7.37}$$

We choose $\eta \in (0, 1)$ and restart the epoch if

$$r^{t,k+1} \leq \eta r^{t,k}.$$

In `CCosmo`, we choose $\eta = 0.4$.

7.4.2 Adaptive Penalty Function Selection

The choice of penalty function $\mathcal{D}_x$ and $\mathcal{D}_s$ dramatically affects the convergence of Algorithm 1 and Algorithm 2. In `CCosmo`, we will choose $\mathcal{D}_x = 10^{-6}\mathcal{D}_{xb}$ and $\mathcal{D}_s = \mathcal{D}_{sa} \circ \mathcal{D}_{sb}$ as the composition of a base scaling which does not change and an adaptive scaling.

7.4.2.1 Static Base Scaling

First-order methods are not affine invariant; rescaling the problem data will result in different iterations. It is therefore common to rescale (7.1) via an invertible map $\mathcal{S}_x : \mathbb{V} \to \mathbb{V}$, an automorphism $\mathcal{S}_\mathcal{K}$ of $\mathcal{K}$, and an invertible map $\mathcal{S}_0$ on the equality-constraint space. Writing $x = \mathcal{S}_x(\hat{x})$, the rescaled problem is

$$\min \hat{\mathcal{P}}(\hat{x}) + \langle \hat{q}, \hat{x} \rangle \tag{7.38a}$$

$$\text{subject to } \hat{\mathcal{A}}(\hat{x}) - \hat{b} \in \mathcal{K} \tag{7.38b}$$

$$\hat{\mathcal{C}}(\hat{x}) - \hat{d} = 0. \tag{7.38c}$$

where

$$\hat{\mathcal{P}} = \mathcal{P} \circ \mathcal{S}_x, \qquad \hat{q} = \mathcal{S}_x^*(q),$$

$$\hat{\mathcal{A}} = \mathcal{S}_\mathcal{K} \circ \mathcal{A} \circ \mathcal{S}_x, \qquad \hat{b} = \mathcal{S}_\mathcal{K}(b),$$

$$\hat{\mathcal{C}} = \mathcal{S}_0 \circ \mathcal{C} \circ \mathcal{S}_x, \qquad \hat{d} = \mathcal{S}_0(d).$$

Since $\mathcal{S}_x$, $\mathcal{S}_\mathcal{K}$, and $\mathcal{S}_0$ are invertible and $\mathcal{S}_\mathcal{K}$ preserves $\mathcal{K}$, (7.1) and (7.38) have the same feasible points up to the change of variables $x = \mathcal{S}_x(\hat{x})$.

Proposition 4. *Let $x = \mathcal{S}_x(\hat{x})$ and $s = \mathcal{S}_\mathcal{K}^{-1}(\hat{s})$. Applying Algorithm 1 to the ADMM standard-form split of (7.38) with penalty functions $\hat{\mathcal{D}}_x$ and $\hat{\mathcal{D}}_s$ is equivalent, under the induced change of variables on the primal and dual iterates, to applying Algorithm 1 to (7.10) with penalty functions*

$$\mathcal{D}_x(x) = \hat{\mathcal{D}}_x(\mathcal{S}_x^{-1}(x)),$$

$$\mathcal{D}_s(s) = \hat{\mathcal{D}}_s(\mathcal{S}_\mathcal{K}(s)).$$

Consequently, the effect of static rescaling in ADMM can be realized by modifying the penalty functions.

The proof is deferred to Section 7.8.3.

A standard choice for scaling maps [22,23,111] is to compute positive diagonal maps $\mathcal{S}_{x,\text{init}}$ and $\mathcal{S}_{s,\text{init}}$ such that the operator

$$\begin{bmatrix} \mathcal{S}_{x,\text{init}} & \\ & \mathcal{S}_{s,\text{init}} \end{bmatrix} \begin{bmatrix} \nabla\mathcal{P} & \mathcal{A}^* \\ \mathcal{A} & 0 \end{bmatrix} \begin{bmatrix} \mathcal{S}_{x,\text{init}} & \\ & \mathcal{S}_{s,\text{init}} \end{bmatrix}, \tag{7.39}$$

is balanced in some way. Following [22,23,40], we use Ruiz scaling [256] to compute $\mathcal{S}_{x,\text{init}}$ and $\mathcal{S}_{s,\text{init}}$ such that all the columns of (7.39) have infinity norm of approximately 1.

This raw Ruiz scaling output need not have the form required by (7.28), which is essential for preserving the efficiency of Line 5.5. Therefore, after computing $\mathcal{S}_{s,\text{init}} = \oplus_{i=1}^{N} \mathcal{S}_{s_i,\text{init}}$, we rectify the scaling on a per-cone basis by replacing each block with its average scaling, denoted by $\overline{\mathcal{S}_{s_i,\text{init}}}$:

$$\begin{aligned} \mathcal{S}_{s_i} &= \overline{\mathcal{S}_{s_i,\text{init}}} I \\ \mathcal{S}_s &= \oplus_{i=1}^{N} \mathcal{S}_{s_i}. \end{aligned}$$

If $\mathcal{K}_i$ is a linear set, i.e. the positive orthant, zero set, or a bounding box, we do not rectify it as Line 5.5 remains efficient under general diagonal scaling. Moving a constraint of the form $\mathcal{A}_i(x) - b_i = 0$ from the block (7.1b) to (7.1c) corresponds to choosing $\mathcal{S}_{s_i} = \infty \cdot \mathcal{S}_{s_i,\text{init}}$.

Our base penalty functions are given by:

$$\mathcal{D}_{xb} = \frac{1}{2}\langle \mathcal{S}_x^{-1}(x), \mathcal{S}_x^{-1}(x) \rangle \tag{7.40}$$

$$\mathcal{D}_{sb} = \frac{1}{2}\langle \mathcal{S}_s(s), \mathcal{S}(s) \rangle \tag{7.41}$$

7.4.2.2 Dynamic Scaling

Selecting $\mathcal{D}_{sa}$ adaptively to improve the convergence is a well-established and successful heuristic in practice [23,40,101,111,257,258]. We select $\mathcal{D}_{sa}$ on a per-set basis. It will be computed either as a scaling ρ_i or a set automorphism $\mathcal{T}_i$.

Residual-Balanced Rescaling By default, `CCosmo` uses the same dynamic penalty scheme as [23,40]. We choose a global scalar $\bar{\rho}$ and a relative scale ρ_0 and set:

$$\mathcal{D}_{s_i}(s) = \begin{cases} \rho_0 \bar{\rho} \langle s, s \rangle & \mathcal{K}_i = \mathbb{0} \\ \bar{\rho} \langle s, s \rangle & \text{else} \end{cases}.$$

That is to say, the zero constraints included in the set block are scaled up by ρ_0 relative to the other set.

By default, we choose $\bar{\rho} = 10^{-1}$ and $\rho_0 = 10^3$. Over the course of the iterations, we allow only $\bar{\rho}$ to vary. This is done by computing

$$\bar{\rho}^{k+1} \leftarrow \bar{\rho}^k \sqrt{\frac{\|s^{k+1} - \tilde{s}^{k+1}\|_\infty \,/\, \max\{\|s^{k+1}\|_\infty, \|\tilde{s}^{k+1}\|_\infty\}}{\|\tilde{s}^{k+1} - \tilde{s}^k\|_\infty \,/\, \max\{\|\tilde{s}^{k+1}\|_\infty, \|\tilde{s}_i^k\|_\infty\}}}. \tag{7.42}$$

As every penalty update triggers a refactorization in Line 5.3, we only accept the new penalties if $\bar{\rho}^{k+1} \geq \tau \bar{\rho}^k$ or if $\bar{\rho}^{k+1} \leq \frac{\bar{\rho}^k}{\tau}$ with $\tau > 1$. We choose $\tau = 5$ by default.

A more aggressive scheme introduces a separate ρ_i for each set and adapts it on a per-set basis.

$$\rho_i^{k+1} \leftarrow \rho_i^k \sqrt{\frac{\left\|s_i^{k+1} - \tilde{s}_i^{k+1}\right\|_\infty / \max\{\left\|s_i^{k+1}\right\|_\infty, \left\|\tilde{s}_i^{k+1}\right\|_\infty\}}{\left\|\tilde{s}_i^{k+1} - \tilde{s}_i^k\right\| / \max\{\left\|\tilde{s}_i^k\right\|, \left\|s_i^k\right\|\}}}. \tag{7.43}$$

For linear sets such as $\mathbb{R}_+$, bounding boxes, and the zero cone, this can be done on a per-coordinate basis. Similar to the global penalty adaptation, we only update the penalties if

$$\left\|\rho^{k+1}\right\|_\infty \geq \tau \left\|\rho^k\right\|_\infty \text{ or } \left\|\rho^{k+1}\right\|_\infty \leq \frac{\left\|\rho^k\right\|_\infty}{\tau}$$

where ρ is the vector all the penalties.

Primal Barrier Rescaling for Symmetric Cones In this section, we propose an alternative adaptive penalty scheme inspired by interior point methods. This should be seen as an alternative to the schemes in the prior section, and should not be composed with them.

As noted in Section 7.4.2.1, first-order methods are not affine invariant and are sensitive to the local geometry of the problem [106,259]. Moreover, it has recently been shown that the local geometry of certain convex programs can force ADMM to converge sublinearly no matter which scalar ρ_i is chosen [260]. For symmetric cone blocks, we therefore propose adapting the penalty to the local geometry of the current iterates.

Let $\mathcal{K}_i$ be a symmetric cone with identity element $e_i \in \operatorname{int}(\mathcal{K}_i)$. For the cones considered here, e_i is the all-ones vector for $\mathbb{R}^n_+$, the vector $(1, 0, \ldots, 0)$ for the Lorentz cone, and the identity matrix for $\mathbb{S}_+$. Given the projected slack iterate $\tilde{s}_i^k \in \mathcal{K}_i$, we define

$$\begin{aligned}
\mu_i^k &= \max\left\{\mu_{\min}, \frac{\langle \tilde{s}_i^k, e_i \rangle}{\langle e_i, e_i \rangle}\right\}, \\
\theta_k &= \max\left\{\theta_{\min}, \frac{1}{k+1}\right\}, \\
s_i^{\sharp,k} &= (1 - \theta_k)\tilde{s}_i^k + \theta_k \mu_i^k e_i.
\end{aligned}$$

Here $\mu_{\min} > 0$ and $\theta_{\min} \in (0, 1)$ are safeguarding parameters. The scalar μ_i^k is the projection of $\tilde{s}_i^k$ onto the identity ray $\{\mu e_i : \mu \geq 0\}$, clipped away from zero, while θ_k controls how strongly the iterate is pulled into the interior. Since $\tilde{s}_i^k \in \mathcal{K}_i$ and $e_i \in \operatorname{int}(\mathcal{K}_i)$, we have $s_i^{\sharp,k} \in \operatorname{int}(\mathcal{K}_i)$ for every k.

Nesterov and Todd [261,262] show that symmetric cones admit canonical automorphisms associated with interior points. Concretely, let $f_{\mathcal{K}}$ be the standard logarithmic barrier function for the symmetric cone $\mathcal{K}$. By choosing $\mathcal{T}_i^k = \left(\nabla^2 f_{\mathcal{K}_i}(s_i^{\sharp,k})\right)^{1/2}$ we get that $s_i^{\sharp,k}$ maps to the identity element:

$$\mathcal{T}_i^k(s_i^{\sharp,k}) = e_i.$$

This yields the block penalty

$$\mathcal{D}_{s_i}^k(h) = \frac{1}{2}\langle \mathcal{T}_i^k(h), \mathcal{T}_i^k(h)\rangle. \tag{7.44}$$

For the standard symmetric cones, these scalings have familiar forms. For $\mathbb{R}_+^n$ we obtain $\mathcal{T}_i^k = \operatorname{diag}((s_i^{\sharp,k})^{-1})$. For $\mathbb{S}_+$, writing $S_i^{\sharp,k} \succ 0$, we use the congruence map $X \mapsto (S_i^{\sharp,k})^{-1/2} X (S_i^{\sharp,k})^{-1/2}$. For the Lorentz cone, we use a Lorentz-cone automorphism sending $s_i^{\sharp,k}$ to the identity element [263].

The scheme extends naturally to bounding boxes $[\ell, u]$. In this case we set $\mathcal{T}_i^k = \operatorname{diag}\left(\frac{1}{s_j^{\sharp,k}-\ell_j}, \frac{1}{u_j - s_j^{\sharp,k}}\right)$. This is not strictly an automorphism of the box, but as any diagonal scaling of the box still admits a closed form expression for Line 5.5, we can use this scaling efficiently.

This construction captures local cone geometry using only the primal projected slack. We leave incorporation of dual information from the residual of the projection to future work.

Once again, to avoid triggering a refactorization unnecessarily, we only accept the update when the candidate penalty metric differs substantially from the currently accepted penalty metric. Letting

$$\mathcal{M}_i^k = (\mathcal{T}_i^k)^* \mathcal{T}_i^k,$$

we compute

$$\Delta_i^k = \max \left|\log\left(\lambda_i\left((\mathcal{M}_i^{k-1})^{-1/2}\mathcal{M}_i^k(\mathcal{M}_i^{k-1})^{-1/2}\right)\right)\right|, \tag{7.45}$$

and accept the update if

$$\left\|\Delta^k\right\|_\infty \geq \log(\tau)$$

In (7.45), we use the matrix logarithm, and the distance (7.45) is the Thompson-Part metric which measures the distance between two positive definite matrices in an affine invariant way [264, §6.4].

We note that computing (7.45) can be done relatively efficiently given the factors $\mathcal{T}_i$ and does not require new eigenvalue factorizations. Nonetheless, as computing the scaling matrices is much more computationally expensive than (7.42), we only do this very occasionally, e.g. every 500 iterations. Moreover, while the metric (7.44) is diagonal in the case of the bounding box and nonnegative orthant, it is not diagonal in the general case of the Lorentz and PSD cones. This may slow down Line 5.3 substantially due to loss of sparsity.

7.4.2.3 Final Penalty Selection

We initialize $\mathcal{D}_x = 10^{-6}\mathcal{D}_{xb}$ where $\mathcal{D}_{xb}$ is given in (7.40). We set

$$\mathcal{D}_s = (\oplus_{i=1}^N \mathcal{D}_{s_i}) \circ \mathcal{D}_{sb},$$

where $\mathcal{D}_{sb}$ is given in (7.41) and $\mathcal{D}_{s_i}$ is one of the proposed penalty methods, Residual-Balanced Rescaling or Primal Barrier Rescaling for Symmetric Cones. We update the penalty according to the update rules in those sections.

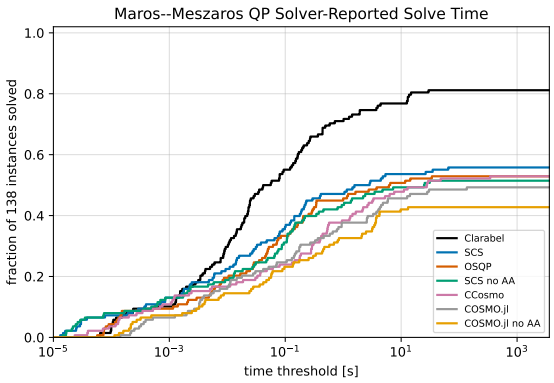


Figure 7.1: Maros–Meszaros QP absolute performance profile.

Solver	Fraction Solved	Shifted GM [s]
`Clarabel`	112/138 (81.2%)	4.93
`SCS`	77/138 (55.8%)	44.1
`OSQP`	73/138 (52.9%)	54.9
`SCS` no AA	71/138 (51.4%)	62.8
`CCosmo`	73/138 (52.9%)	66.5
`COSMO.jl`	68/138 (49.3%)	84.5
`COSMO.jl` no AA	59/138 (42.8%)	132

Table 7.1: Shifted geometric mean and the fraction solved in the time/iteration limit of each solver on the Maros–Meszaros QPs sorted by shifted geometric mean.

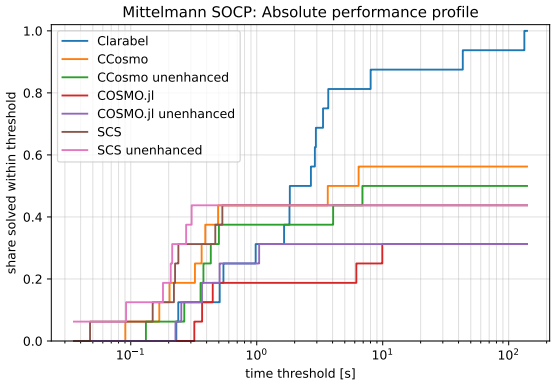


Figure 7.2: Mittelmann SOCP absolute performance profile.

Solver	Fraction Solved	Shifted GM [s]
`Clarabel`	16/16 (100.0%)	3.20
`CCosmo`	9/16 (56.2%)	3.14×10^1
`CCosmo` unenhanced	8/16 (50.0%)	5.03×10^1
`SCS` unenhanced	7/16 (43.8%)	6.06×10^1
`SCS`	7/16 (43.8%)	6.23×10^1
`COSMO.jl` unenhanced	5/16 (31.2%)	1.57×10^2
`COSMO.jl`	5/16 (31.2%)	1.95×10^2

Table 7.2: Shifted geometric mean and the fraction solved in the time/iteration limit of each solver on the Mittelmann SOCP sorted by shifted geometric mean.

7.5 Results

7.5.1 Standard Form Benchmark

We compare `CCosmo` against `Clarabel` [22], `SCS` [210,220], and `COSMO.jl` [40], and include `OSQP` [23] on QP benchmarks. To compare the base algorithms, we also compare to `SCS` and `COSMO.jl` with acceleration disabled; we call these the unenhanced versions. All problem instances are parsed using `CVXPY`, except some GCS problems which are parsed using `Drake`. Experiments are run on an 11th Gen Intel Core i9-11980HK CPU with 64 GiB RAM to absolute and relative error tolerances of 10^{-3} and 10^{-4} with a 10k iteration limit and a timeout of 1 hour.

In `CCosmo`, we use the static-penalty scheme from Static Base Scaling, the scalar residual balancing scheme of Section 7.4.2.2, and run without any acceleration, setting $\alpha = 1.7$. This is done to configure the solver to be in a similar default setting as `OSQP` and `COSMO.jl`. Moreover, as shown in Section 7.5.3, the additional enhancements did not show conclusively better results.

We begin by evaluating the solvers' performance on two challenging benchmarks: 138

QPs from the Maros–Meszaros benchmark [265] and the 16 smallest SOCP instances[4] from the 7th DIMACS challenge [266] hosted on the Mittelmann benchmark [267]. All solvers are run with absolute and relative tolerances of 10^{-3} and 10^{-4}. When a solver returns a solution, we check whether the returned cost is within 5 times the relative tolerance of the known solution, otherwise we declare the result a failure. If a solver fails, its reported time to solve an instance is set to the time limit.

We report absolute performance profiles for both the Maros–Meszaros benchmark in Figure 7.1 and the Mittelmann benchmark in Figure 7.2. These profiles show the fraction of instances a solver succeeds in solving within a specified time window [22]. We also report the shifted geometric mean [267], a standard metric for comparing solver performance, for the Maros–Meszaros benchmark in Table 7.1 and the Mittelmann SOCP benchmark Table 7.2.

The results show that `CCosmo` is competitive with other open-source first-order solvers, as expected since it implements a very similar algorithm. `SCS` is typically slightly faster than `CCosmo` on the Maros–Meszaros and Mittelmann benchmarks as `SCS` takes fewer iterations. This is consistent with the observations of [220]. `OSQP` similarly reports faster times on the Maros–Meszaros benchmark as it typically takes fewer iterations. However, `OSQP` does not include the duality gap (7.31c) as part of its convergence criterion and so returns slightly worse solutions. Finally, `CCosmo` demonstrates both better robustness and performance than `COSMO.jl`. In this case, the robustness gap is primarily due to `COSMO.jl` failing to return a solution within 5 times the relative tolerance of the known solution.

Due to the challenging nature of these problems, the more robust second-order interior point solver `Clarabel` outperforms the first-order solvers in terms of problems solved. However, on the easier instances in the benchmarks, first-order solvers typically return solutions in comparable time, or more quickly.

7.5.2 Fixed Contact-Schedule Centroidal Locomotion MPC

One of the primary advantages of first-order solvers is the ease with which they can be warm-started. This is in contrast to interior point methods which are notoriously difficult to warm-start.

To demonstrate the advantages of this, we consider a repeated fixed contact-schedule centroidal locomotion MPC benchmark similar to [268] and [156, Chpt. 5, Chpt. 7]. We model friction at the feet using second-order cones, which yields a sequence of related SOCPs. Across a rollout, consecutive problems differ only through the shifted state and reference data, making this a natural setting for warm starting. Each problem in the benchmark has 1340 variables and 1556 constraints, and we evaluate both a nominal-tracking rollout and a push-recovery rollout. For details of the problem formulation, see Appendix B.2.

We compare the performance of `CCosmo`, `SCS`, and `Clarabel` on this task, warm starting `CCosmo` and `SCS` from the previous solution. In the nominal-tracking rollout, `CCosmo` attains a median solve time of 2.13 ms with interquartile range 0.05 ms, compared with 3.10 ms and 1.04 ms for `SCS` and 4.78 ms and 1.84 ms for `Clarabel`. In the push-recovery rollout, `CCosmo` attains a median of 5.40 ms with interquartile range 0.18 ms, compared with 6.24 ms and 0.58 ms for `SCS` and 10.66 ms and 4.13 ms for `Clarabel`. These results are summarized in

[4]Problems `nql180` and `qssp180` are too large for the hardware of the test machine

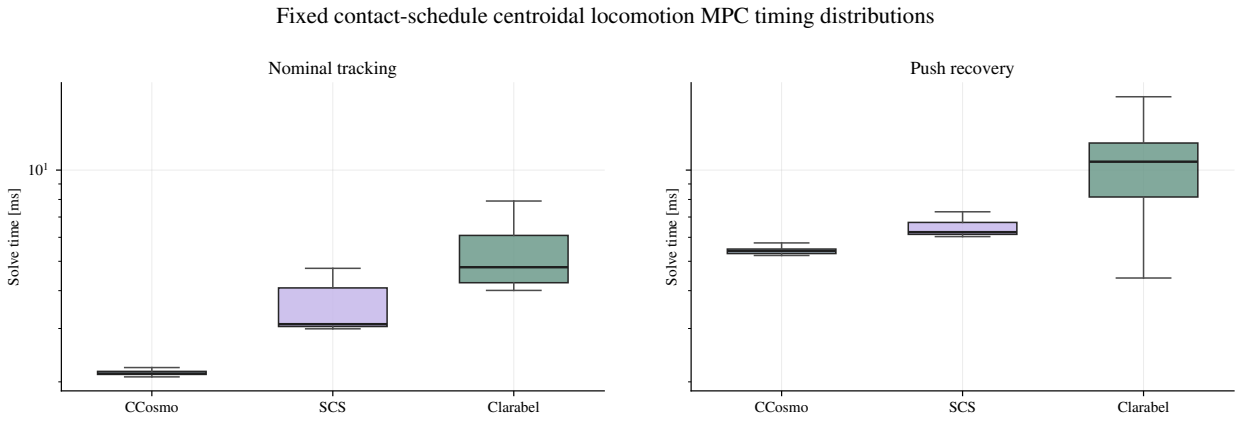


Figure 7.3: Timing distribution for the warm-starting, fixed-contact-schedule locomotion MPC benchmark. Each box summarizes all repeated solves in one rollout regime, excluding the first solve.

Figure 7.3.

7.5.3 Enhancements Ablation

In this section, we consider the effect of each of the enhancements proposed in Section 7.4 when used on the full Maros–Meszaros benchmark. We begin by comparing the various penalty adaptation schemes when run on the base algorithm Algorithm 5. The main conclusion is that the scalar residual-balancing penalty update is the most reliable individual enhancement on this benchmark. Next, we consider the effects of using the accelerated scheme in Algorithm 2 compared to Algorithm 1. It is a well-known phenomenon that accelerated rates in theory do not always translate to accelerated rates in practice, and on this benchmark the KM/Halpern recurrence is not a robust standalone replacement for penalty adaptation. Finally, we consider the effects of composing the best penalty adaptation scheme with the best acceleration schemes. In all experiments, we turn the static scaling off.

The strongest KM composition solves slightly more problems, but the unaccelerated scalar-balancing rows have slightly better shifted geometric mean. On instances where both KM acceleration and simple over-relaxation succeed, the actual wall-clock time from using the simpler over-relaxation scheme is better.

Remark 11. *To better compare between different ablations, we plot all instances where at least one solver succeeds across all ablations in this section. If a column is blank, it means that a solver from a different ablation succeeded on that instance, but none in the current cross view did. This is done to ensure that the ith problem instance in e.g. Figure 7.4 corresponds to the ith problem instance in e.g. Figure 7.6.*

7.5.3.1 Penalty Adaptation Ablations

To isolate the effect of the penalty rule, we set $\alpha = 1$ and disable acceleration throughout this ablation and report the results in Table 7.3. Only solver status `optimal` which returns

variant	strict	inacc.	max iters	SGM [s]	median penalty adaptations	common strict	cand. only	base only	median common solved iter ratio
fixed ρ	32	1	76	565	0	32	0	0	1.000
scalar balancing	63	3	49	97.40	1	30	33	2	1.000
coordinatewise	37	9	76	428	2	21	16	11	1.000
barrier	50	8	73	217	2	28	22	4	1.000

Table 7.3: Penalty adaptation ablation on the full Maros–Meszaros benchmark. The baseline for the median iteration ratio and base only columns uses a fixed $\rho = 0.1$ with $\alpha = 1$. Strict solves are those with solver status `optimal` which return a solution within 5 times the relative tolerance of the known solution; all other statuses are charged as failures in the shifted geometric mean.

a solution within 5 times the relative tolerance of the known solution counts as a strict solve; `optimal_inaccurate`, max-iteration, error, and objective-failure statuses are charged as failures in the shifted geometric mean. We also report the median number of times the penalty is adapted over the course of the solve, the number of instances each variant solved in common compared to the fixed-penalty solver, the number the variant only solved, and the number the base solver only solved. Finally, for each variant we consider all the instances commonly solved by that variant and the base solver. For these common instances, we consider the ratio of the number of iterations taken by the variant versus the base solver. Ratios lower than 1 indicate that the variant took fewer iterations on the commonly solved instances.

The fixed-ρ baseline solves (32/138) instances with a shifted geometric mean of 565 seconds. The clear winner is the scalar residual-balancing update, which solves (63/138) instances with a shifted geometric mean of 97.40 seconds. This rule solves 33 instances that the fixed-ρ baseline fails to solve, while failing on 2 instances solved by the fixed-ρ baseline.

Barrier metric rescaling is safer than coordinatewise rescaling, but it does not compete with scalar residual balancing on this benchmark. It solves (50/138) instances with a shifted geometric mean of 217 seconds. It solves 22 instances that the fixed-ρ baseline does not, while failing on 4 instances solved by the baseline.

We see that on instances where both the variant and the fixed-penalty solver succeeded, both solvers take the same number of steps, indicating that adaptation is only needed and used when the problem requires it.

Coordinatewise rescaling appears too aggressive: it solves only (37/138) instances, with a shifted geometric mean of 428 seconds.

In Figures 7.4 and 7.5, we compare iteration count and wall-clock time across instances solved by at least one plotted variant. We plot the log of the ratio of each metric against the best performing variant on that instance. The easy instances do not strongly separate the penalty rules. The harder instances do: scalar residual balancing is the only penalty adaptation that improves both coverage and shifted geometric mean at benchmark scale.

7.5.3.2 KM/Halpern Acceleration

To isolate the effect of the acceleration recurrence, we next disable dynamic penalty adaptation and compare the accelerated KM/Halpern variants and over-relaxation variants with

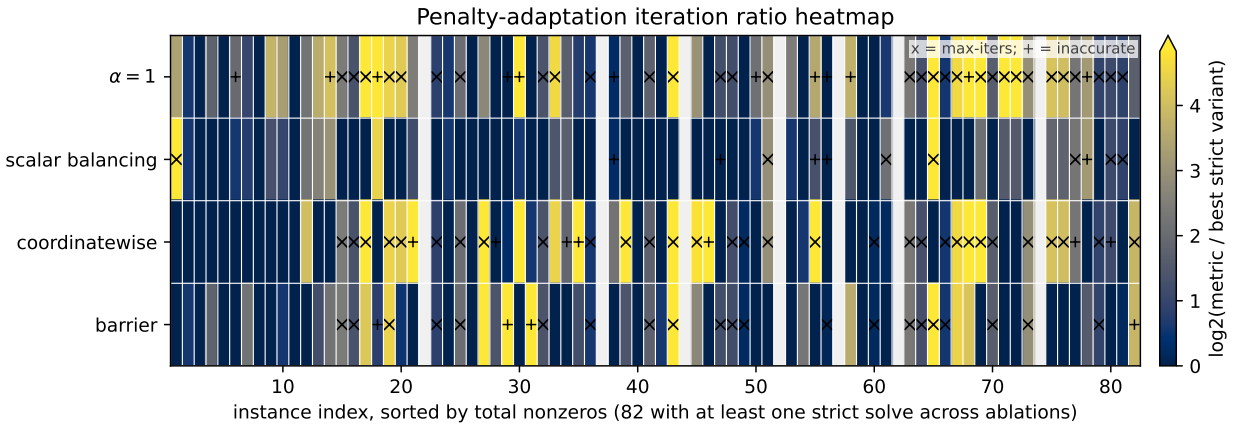


Figure 7.4: Impact of different penalty adaptation schemes on iteration count for Maros–Meszaros instances solved by at least one plotted variant. We show the log of the iteration count of an instance normalized by the best variant on that instance.

a fixed penalty parameter and $\alpha = 1$. Table 7.4 reports the results of the two types of acceleration. The first runs Algorithm 5, but sweeps the relaxation parameter $\alpha \in \{0.8, 1.2, 1.6\}$. The second wraps Algorithm 5 in the acceleration loop of Algorithm 2 while potentially allowing for restarts. We fix $\alpha = 1$, and sweep pairs of $\gamma \in \{1.5, 1.8\}$ and $\beta \in \{10, 15, 20\}$ with restarts both on and off. We also compare to the accelerated variant $\alpha = 1, \gamma = 1, \beta = 2$, which corresponds to pure Halpern iteration.

Unlike in Section 7.5.3.1, no acceleration scheme substantially improves coverage over the fixed-ρ baseline. Simple over-relaxation is the most useful acceleration-only mechanism: $\alpha = 1.6$ solves (36/138) instances, compared with (32/138) for $\alpha = 1$. Pure Halpern iteration performs poorly without restarts, solving only (11/138) instances, though restarts recover much of the lost performance.

In Figures 7.6 and 7.7 we compare the baseline to the acceleration schemes. These figures have far more blank columns, compared to Figures 7.4 and 7.5 because many problems require penalty adaptation to solve at all. Restarting improves the robustness of pure Halpern iteration, but it does not make Halpern competitive with simple over-relaxation.

The best fixed-ρ acceleration-only story is therefore not KM/Halpern acceleration, but the classical over-relaxation choice $\alpha = 1.6$. Perhaps the most important finding is that over-relaxation with $\alpha = 1.6$ is the only method which consistently reduces the iteration count relative to the baseline on commonly solved instances. All accelerated variants take more iterations, with the Halpern variants taking substantially more.

Nonetheless, the overall conclusion is that penalty adaptation is much more important than acceleration on this benchmark.

7.5.3.3 Composing Enhancements

Finally, we consider whether the adaptation of the penalty parameters composes well with the acceleration schemes of the prior section. We compare composing scalar residual balancing, barrier metric rescaling, and coordinatewise rescaling with the best performing accelerated

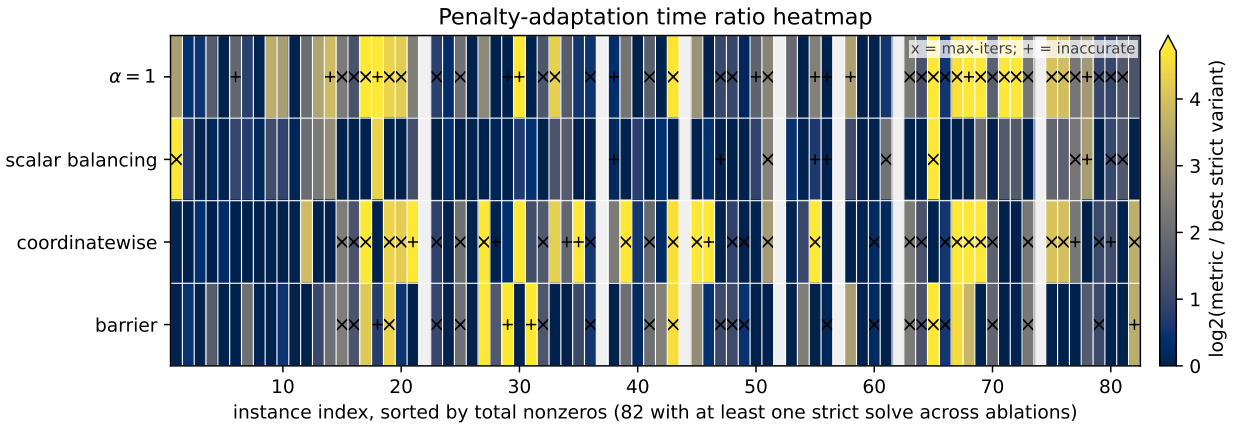


Figure 7.5: Impact of different penalty adaptation schemes on wall-clock time for Maros–Meszaros instances solved by at least one plotted variant. We show the log of the solve time of an instance normalized by the best variant on that instance.

variants. We use the unaccelerated algorithm with scalar residual balancing and $\alpha = 1.6$ as the baseline, and compare against KM variants with $\gamma = 1.8, \beta = 15$ and $\gamma = 1.5, \beta = 20$, both with and without restarts. The results are summarized in Table 7.5.

Composing KM/Halpern acceleration with scalar residual balancing improves strict solve coverage slightly. The best row, KM with $\gamma = 1.8, \beta = 15$ and restarts, solves (64/138) instances with a shifted geometric mean of 98.95 seconds. The scalar residual-balancing baseline with $\alpha = 1.6$ solves (63/138) instances with shifted geometric mean 100.88 seconds. This is a coverage improvement, but not a clean timing improvement. On common solved instances, the accelerated row is 1.589× slower and has a median iteration ratio of 1.000 against the $\alpha = 1.6$ scalar-balancing baseline, indicating that it is not taking fewer iterations, but is iterating almost 60% slower.

The barrier and coordinatewise compositions do not show the same benefit. Barrier metric rescaling alone solves (50/138) instances, while the best barrier-composed row solves (48/138). Coordinatewise rescaling solves (37/138) instances, while the best coordinatewise-composed rows solve (31/138). Thus the composition result is best understood as a robustness-speed tradeoff for scalar residual balancing, not as evidence that KM/Halpern acceleration is a generally useful replacement for over-relaxation.

We note that the best performing variant of `CCosmo` in this section solves fewer instances than in Section 7.5.1 because we have disabled static-scaling in this ablation.

variant	strict	inacc.	max iters	SGM [s]	median solved iters	median restarts	common strict	cand. only	base only	GM time ratio	median common solved iter ratio
$\alpha = 1$	32	1	76	565	226	0	32	0	0	1.000	1.000
$\alpha = 0.8$	31	2	77	589	226	0	30	1	2	1.095	1.250
$\alpha = 1.2$	33	2	72	526	201	0	32	1	0	0.821	1.000
$\alpha = 1.6$	36	3	70	449	164	0	31	5	1	0.825	0.669
pure Halpern (none)	11	2	97	1879	376	0	10	1	22	11.10	10.67
pure Halpern (restart)	28	0	77	731	401	3	28	0	4	4.683	2.471
KM $\alpha = 1$, $\gamma = 1.5$, $\beta = 20$ (restart)	33	0	77	545	251	2	31	2	1	2.125	1.358
KM $\alpha = 1$, $\gamma = 1.8$, $\beta = 15$ (none)	32	0	77	572	264	0	30	2	2	1.570	1.076
KM $\alpha = 1$, $\gamma = 1.5$, $\beta = 10$ (restart)	32	0	77	580	288	2	30	2	2	2.198	1.395
KM $\alpha = 1$, $\gamma = 1.5$, $\beta = 2$ (restart)	32	0	78	588	451	3	29	3	3	3.460	1.906
KM $\alpha = 1$, $\gamma = 1.8$, $\beta = 15$ (restart)	32	0	76	589	238	2	30	2	2	1.812	1.054
KM $\alpha = 1$, $\gamma = 1.8$, $\beta = 20$ (restart)	31	0	75	604	226	2	29	2	3	1.868	1.109
KM $\alpha = 1$, $\gamma = 1.8$, $\beta = 2$ (restart)	31	0	77	609	376	3	28	3	4	3.239	1.757
KM $\alpha = 1$, $\gamma = 1.5$, $\beta = 20$ (none)	31	0	77	609	251	0	29	2	3	1.769	1.483
KM $\alpha = 1$, $\gamma = 1.8$, $\beta = 10$ (restart)	31	0	76	610	226	2	29	2	3	1.914	1.150
KM $\alpha = 1$, $\gamma = 1.5$, $\beta = 15$ (restart)	31	0	77	613	251	2	30	1	2	2.227	1.362
KM $\alpha = 1$, $\gamma = 1.2$, $\beta = 15$ (restart)	31	0	76	615	276	2	30	1	2	2.759	1.699
KM $\alpha = 1$, $\gamma = 1.2$, $\beta = 20$ (restart)	31	0	76	622	301	2	30	1	2	2.753	1.657
KM $\alpha = 1$, $\gamma = 1.8$, $\beta = 20$ (none)	30	0	77	641	201	0	29	1	3	1.580	1.000
KM $\alpha = 1$, $\gamma = 1.5$, $\beta = 15$ (none)	30	0	77	647	276	0	29	1	3	2.000	1.490
KM $\alpha = 1$, $\gamma = 1.5$, $\beta = 10$ (none)	30	0	77	649	364	0	29	1	3	2.187	1.490
KM $\alpha = 1$, $\gamma = 1.2$, $\beta = 20$ (none)	30	0	76	654	338	0	29	1	3	2.168	1.794
KM $\alpha = 1$, $\gamma = 1.2$, $\beta = 15$ (none)	30	0	77	654	326	0	29	1	3	2.259	1.794
KM $\alpha = 1$, $\gamma = 1.2$, $\beta = 10$ (restart)	30	0	76	661	351	2	28	2	4	2.644	1.706
KM $\alpha = 1$, $\gamma = 1.8$, $\beta = 10$ (none)	29	0	77	684	326	0	28	1	4	1.718	1.175
KM $\alpha = 1$, $\gamma = 1.2$, $\beta = 10$ (none)	28	1	77	728	438	0	27	1	5	2.498	1.962
KM $\alpha = 1$, $\gamma = 1.2$, $\beta = 2$ (restart)	28	1	78	739	488	3	27	1	5	3.970	2.147
KM $\alpha = 1$, $\gamma = 1.8$, $\beta = 2$ (none)	13	1	95	1667	226	0	12	1	20	8.860	8.042
KM $\alpha = 1$, $\gamma = 1.2$, $\beta = 2$ (none)	13	0	96	1668	326	0	12	1	20	11.51	12.54
KM $\alpha = 1$, $\gamma = 1.5$, $\beta = 2$ (none)	13	1	95	1668	276	0	12	1	20	11.31	9.654

Table 7.4: Fixed-ρ KM/Halpern and over-relaxation rows on the full Maros–Meszaros benchmark. The pairwise columns compare against fixed ρ with $\alpha = 1$. These rows isolate acceleration from penalty adaptation, pin the pure Halpern rows, and include the over-relaxed $\alpha = 1.6$ fixed-ρ control.

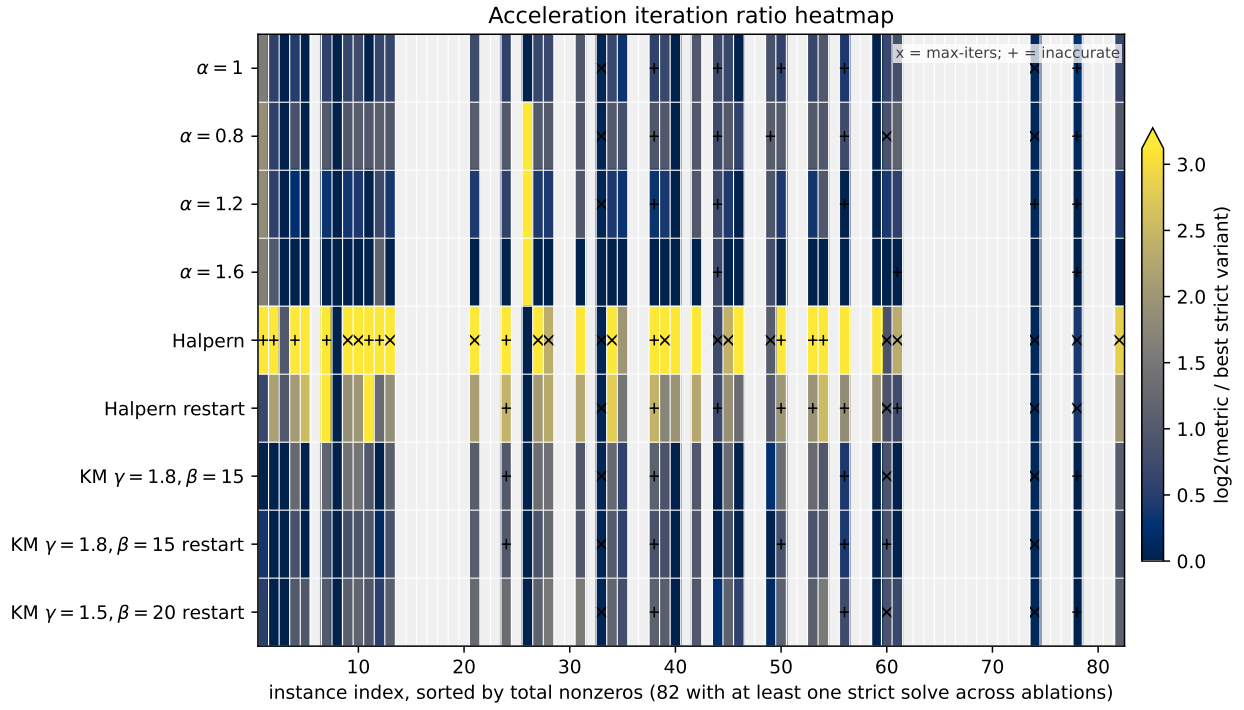


Figure 7.6: Impact of different acceleration schemes on iteration count for Maros–Meszaros instances solved by at least one plotted variant.

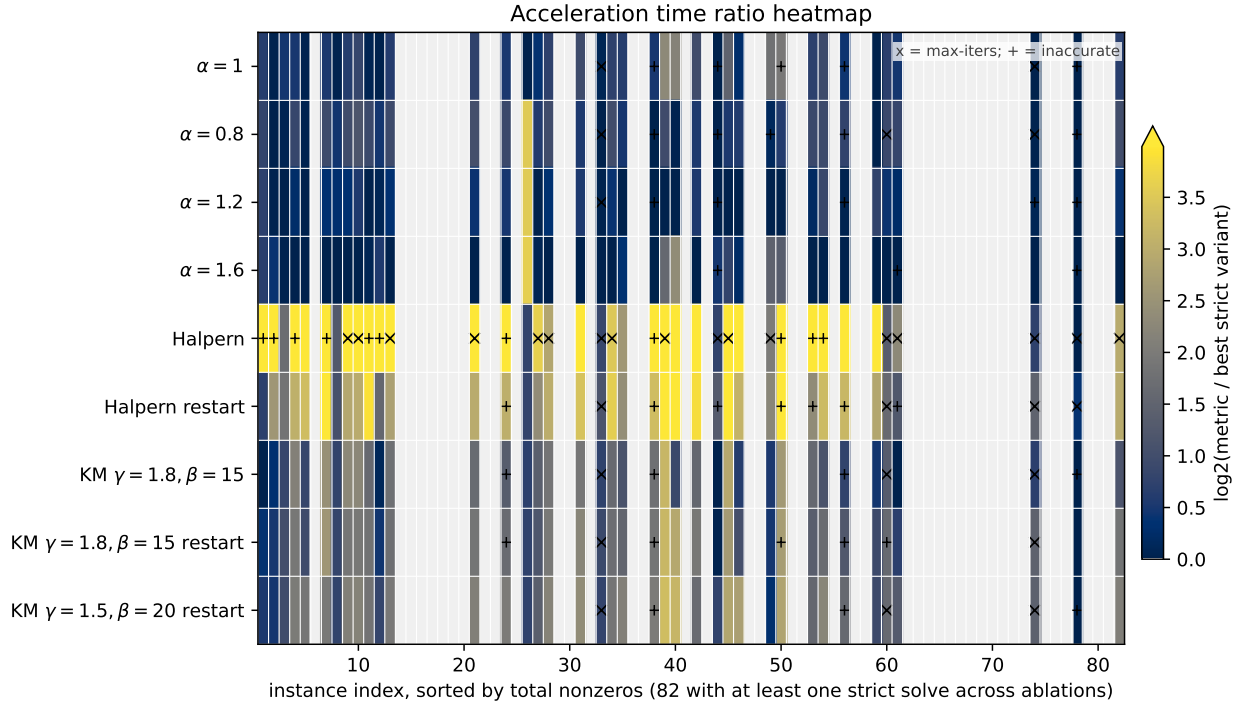


Figure 7.7: Impact of different acceleration schemes on wall-clock time for Maros–Meszaros instances solved by at least one plotted variant.

variant	strict	inacc.	max iters	SGM [s]	median restarts	common strict	cand. only	base only	GM time ratio	median common solved iter ratio
$\alpha = 1.6$ scalar balancing	63	3	46	101	0	63	0	0	1.000	1.000
scalar balancing	63	3	49	97.40	0	57	6	6	0.91	1.000
KM $\alpha = 1$, $\gamma = 1.8$, $\beta = 15$ (scalar balancing, restart)	64	0	48	98.95	2	54	10	9	1.589	1.000
KM $\alpha = 1$, $\gamma = 1.5$, $\beta = 20$ (scalar balancing, restart)	63	1	49	104	2	55	8	8	1.637	1.124
KM $\alpha = 1$, $\gamma = 1.8$, $\beta = 15$ (scalar balancing)	63	1	47	105	0	54	9	9	1.480	1.006
KM $\alpha = 1$, $\gamma = 1.5$, $\beta = 20$ (scalar balancing)	62	2	46	110	0	53	9	10	1.485	1.154
barrier	50	8	73	217	0	41	9	22	1.050	1.000
KM $\alpha = 1$, $\gamma = 1.8$, $\beta = 15$ (barrier, restart)	48	5	80	248	1	39	9	24	2.378	1.307
KM $\alpha = 1$, $\gamma = 1.5$, $\beta = 20$ (barrier, restart)	46	11	73	277	1	37	9	26	2.019	1.000
KM $\alpha = 1$, $\gamma = 1.8$, $\beta = 15$ (barrier)	44	7	81	298	0	38	6	25	2.388	1.432
KM $\alpha = 1$, $\gamma = 1.5$, $\beta = 20$ (barrier)	41	11	76	354	0	37	4	26	2.080	1.284
coordinatewise	37	9	76	428	0	30	7	33	1.067	1.000
KM $\alpha = 1$, $\gamma = 1.8$, $\beta = 15$ (coordinatewise)	31	5	96	585	0	29	2	34	2.376	1.000
KM $\alpha = 1$, $\gamma = 1.8$, $\beta = 15$ (coordinatewise, restart)	31	5	96	586	1	29	2	34	2.553	1.000
KM $\alpha = 1$, $\gamma = 1.5$, $\beta = 20$ (coordinatewise)	29	3	98	654	0	27	2	36	1.880	1.000
KM $\alpha = 1$, $\gamma = 1.5$, $\beta = 20$ (coordinatewise, restart)	29	3	98	659	1	27	2	36	2.233	1.000
pure Halpern (coordinatewise)	28	9	85	691	0	26	2	37	1.626	1.000
pure Halpern (barrier)	27	13	91	752	0	25	2	38	2.232	1.198
pure Halpern (scalar balancing)	17	3	88	1323	0	17	0	46	5.637	5.809

Table 7.5: Composition ablation on the full Maros–Meszaros benchmark. The pairwise columns compare against scalar residual balancing with $\alpha = 1.6$.

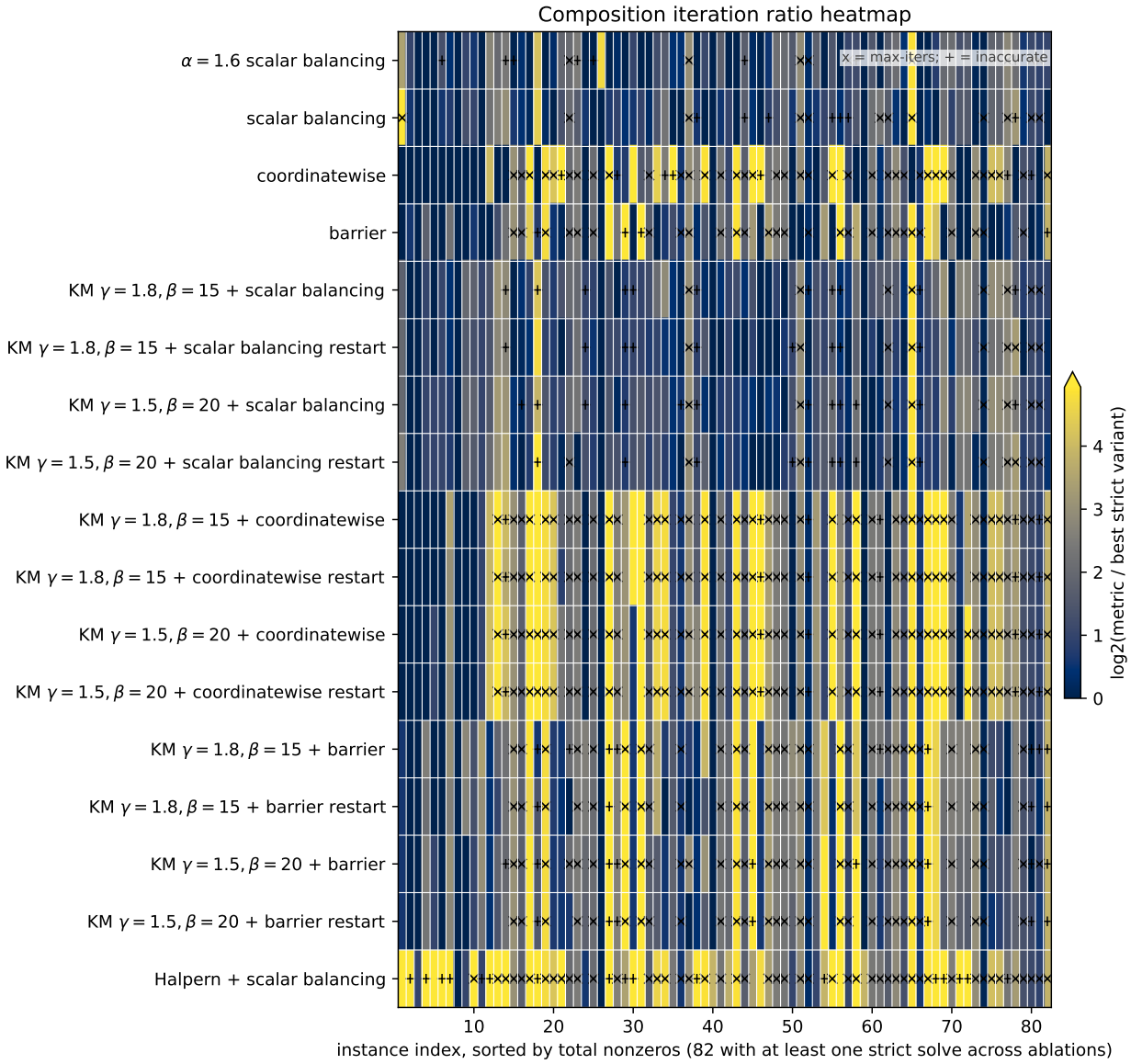


Figure 7.8: Impact of composed penalty adaptation and acceleration schemes on iteration count for Maros–Meszaros instances solved by at least one plotted variant.

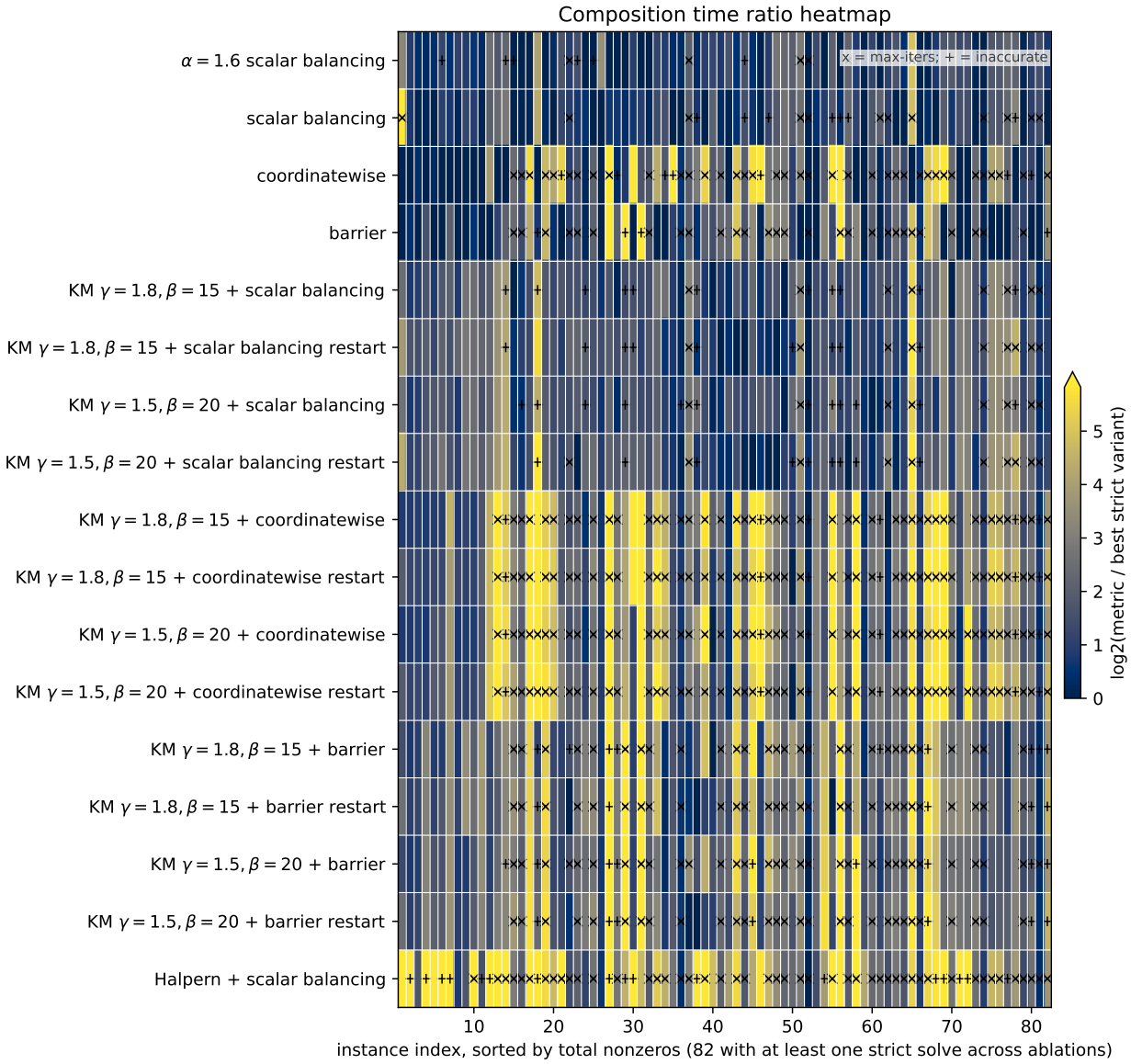


Figure 7.9: Impact of composed penalty adaptation and acceleration schemes on wall-clock time for Maros–Meszaros instances solved by at least one plotted variant.

7.5.4 Matrix Standard Form Results

We consider using the matrix standard form implementation to solve the shortest path in a GCS convex relaxation of various vertex programs described in Section 9.3. These programs have the form

$$\min \frac{1}{2} \|X - X_{\text{ref}}\|_F^2 \tag{7.46a}$$

$$\text{subject to } AXB \in \mathcal{K} \tag{7.46b}$$

$$XC = 0. \tag{7.46c}$$

We report the performance on four different families which show dramatically different sparsity. We consider three instances from the `IiwaShelf` environment which all have 11 vertices, two instances from the `Bimanual` each having 15 vertices, the `cg_simple_4` environment which has 194 vertices, and three instances of `Maze` which have 225, 100, and 25 vertices each, respectively. In Table 7.6, we report the structure of each instance and make note of the number of non-zero entries and the density of the constraint matrix after vectorizing the problem.

In Table 7.7, we report the shifted geometric mean with a one-second shift for solving the problems in vector standard form versus matrix standard form. We show the absolute performance profiles in Figure 7.10. In these instances, we note that the density of the program substantially affects whether the matrix-standard form outperforms the vector standard form. This is due to the fact that the matrix standard form solver necessarily uses dense linear algebra (see Chapter 8) while the vector standard form can use either sparse or dense linear algebra depending on the problem.

On the relatively dense instances of `IiwaShelf`, `Bimanual`, and `Maze`, we see that the matrix-algebra allows the solver to improve its solve time. However, on the extremely sparse `cg_simple_4`, the price of performing the dense linear algebra does not outweigh the advantage of the structured solve. In Figure 7.11 we show that the entirety of the discrepancy between the two solvers is due to the different backends used to solve the linear systems. We exclude `cg_simple_4` due to the memory requirements of running the fully instrumented solver on this large family.

The sparse vector backend is surprisingly effective on these programs because the B factor in (7.46b) is very sparse and the C factor is very small. For larger programs, the sparse vector backend is expected to be much less effective. The resulting vectorization of the program

Family	Vertices	Vars	Rows	A_{kron} nnz	A_{kron} density
`IiwaShelf`	33	207	8740	106087	6.07%
`Bimanual`	30	259	3168	30536	3.72%
`cg_simple_4`	194	1012	7586	31964	0.416%
`Maze`	350	98	725	3220	4.37%

Table 7.6: Structural summary for the GCS vertex matrix-program benchmark. Size, nonzero, and density columns report medians over vertices. The density column is $\text{nnz}(A_{\text{kron}})/(mn)$, where $A_{\text{kron}} \in \mathbb{R}^{m \times n}$ is the vectorized affine operator for the two matrix constraints.

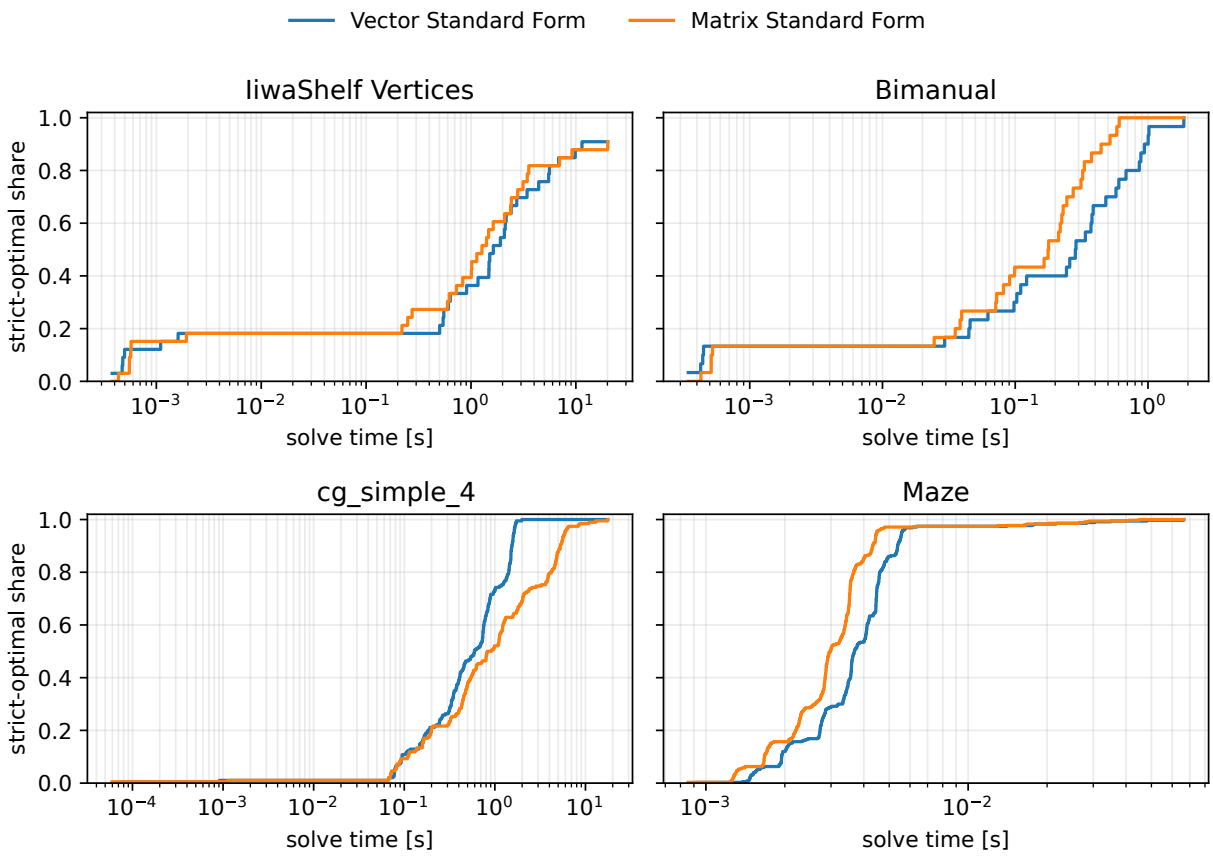


Figure 7.10: Absolute performance profiles for the vertex families. The result is family-dependent: `Maze` and `Bimanual` favor matrix standard form, `IiwaShelf` is close by shifted geometric mean, and the `cg_simple_4` favors vector standard form.

is therefore only modestly larger and extremely sparse as well. To demonstrate that even a mild disruption of this phenomenon leads to performance gains, we consider adding two dense rows of the form $h^T(y_v, y_{\mathcal{E}_v}) \geq 0$ to every vertex local constraint. We choose $h \geq 0$, so this constraint is completely redundant for the program: it does not change the optimal value of the feasible set. These dense rows lead to a dense column in the vectorization of (7.46b). We compare the effects of adding these rows in Figure 7.12. Including these rows does almost nothing to the matrix backend as it is necessarily in dense linear algebra. On the other hand, even this minor disruption to the sparsity leads to a more than 10% change in the runtimes of the sparse backend.

7.6 `CCosmo` on the GPU

An emerging trend in convex optimization is the utilization of accelerated hardware such as GPUs to increase the speed at which convex optimization programs can be solved. The vast majority of this work has been focused on using the massive parallelism in a GPU to accelerate the solution of a single, very large, sparse program [84,111,229,232,269,270]. Recently,

Family	Vertices	Vector SGM [s]	Matrix SGM [s]	Median Ratio	Geom. ratio	Matrix faster
`IiwaShelf` vertices	33	2.82	2.49	0.695	0.815	18/30
`Bimanual`	30	0.350	0.187	0.648	0.634	26/30
`cg_simple_4`	194	0.637	1.30	1.475	1.751	15/194
`Maze`	350	0.00437	0.00356	0.812	0.816	349/350

Table 7.7: Comparison of solve times for the vertex programs of GCS convex relaxations in both vector standard form and matrix standard form. The median ratio is the median of matrix standard form solve time divided by vector standard form solve time. Ratios below 1 indicate that the matrix solve was faster. We see that for the denser families, the matrix backend outperforms the vector backend.

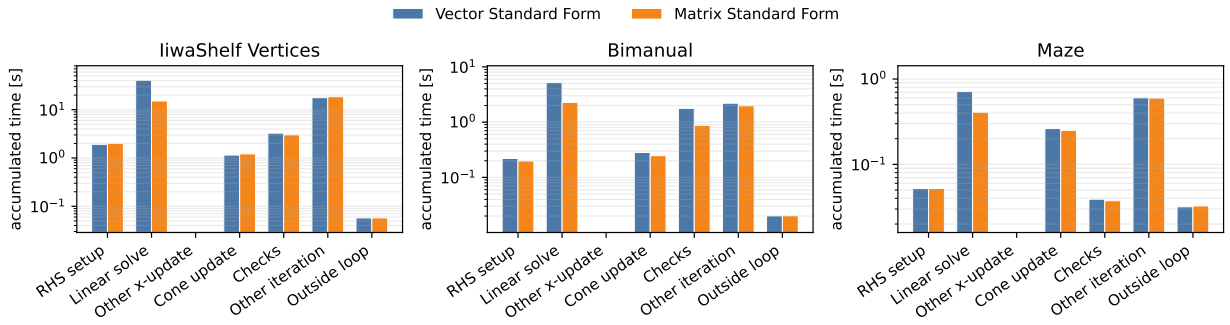


Figure 7.11: Accumulated time in each step of Algorithm 5 across all vertices in different families. We compare solving the programs from the GCS convex relaxations using the matrix standard form backend against the vector standard form backend. Almost the entirety of the discrepancy in solve times is accounted for by the difference in performing the linear solve.

there has been interest in using the parallel capabilities of the GPU for solving batches of optimization programs [237,238]. These batched settings occur naturally in robotics such as when performing trajectory optimization [271], collision detection [117,152], or when embedding optimization problems as a layer in a neural network [272]. In some instances, a batch of optimization programs is *structurally* identical: the operators $\mathcal{P}$, $\mathcal{A}$ and $\mathcal{C}$ have the same shape and non-zeros, but different values, and the set $\mathcal{K}$ is the same. In other instances, such as in collision detection, the programs in the batch may differ structurally. The heterogeneous batch setting is typically handled by simply padding smaller instances up to the size of the largest instances.

GPUs achieve their superior numeric throughput compared to CPUs by combining much higher on-chip parallelism with a memory system built to sustain that parallelism. A CPU is designed primarily to minimize the latency of a relatively small number of threads. To do this, it spends a large fraction of its chip area on deep cache hierarchies, branch prediction, out-of-order execution, and other control mechanisms that help a single thread make progress quickly even on irregular code. A GPU makes the opposite tradeoff. It devotes much less area to sophisticated per-core control and caching and instead uses that silicon budget for substantially more, but simpler cores and a much wider memory system. When an algorithm exposes significant independent work, a GPU is able to perform substantially more calculations than a CPU, hiding latency by switching between threads that are ready to run.

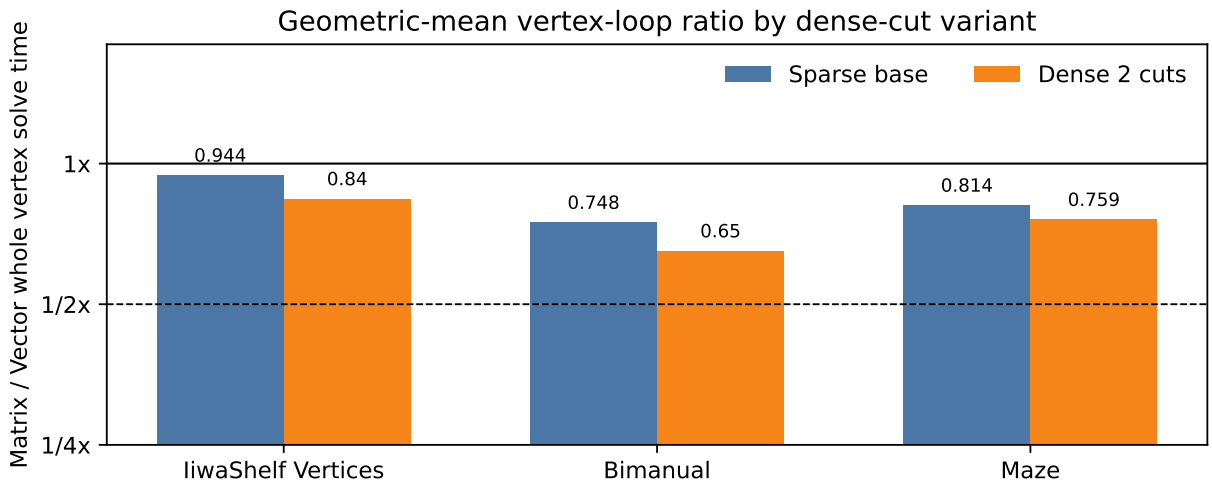


Figure 7.12: Ratio of solve times between the matrix-standard to vector forms before and after adding two dense rows to the right factor of AXB. Ratios below one favor the matrix standard form solver. This dense cut affects the efficiency of the vector standard form solver much more than the matrix standard form solver, leading to a relative speedup of the matrix standard form solver.

The GPU pairs substantially more cores with substantially higher memory bandwidth to ensure that the cores can stream data quickly enough to avoid stalling.

The result is that GPUs excel at processing highly regular, massively parallel workloads with regular memory accesses and uniform control flow. In principle, this makes trivially parallelizable batch workflows, such as solving batches of convex optimization problems, ideal candidates for acceleration using a GPU. The most natural way to parallelize Algorithm 5 over a batch of (7.10) is to simply batch the operations of Algorithm 5 and run the entire batch in lockstep. This can be an effective strategy, particularly for structurally identical batches, and is the approach implemented by [237,238] for performing one loop of Algorithm 5.

On the other hand, even structurally identical convex programs can require substantially different numbers of iterations to converge. In the case when some programs converge substantially faster than others, the GPU is either left performing unnecessary work on the already converged programs or we must implement a masking scheme to ensure that the GPU skips work on these programs. Both result in underutilization of the GPU: the former by doing unnecessary work and the latter by affecting the way the GPU must access memory.

Another way to achieve parallelism would be to consider the entire loop of Algorithm 5 as a single workflow and batch across the entire ADMM loop. Even in this setting, it is not trivial to obtain good performance on the GPU due to the complex branching that occurs in the loops, particularly in the convergence checks. Nevertheless, in this section we show that batching solves by treating Algorithm 5 as a single workflow can be an effective strategy.

7.6.1 A Primer on GPU Architectures

We briefly review the organization of the GPU programming model as implemented in NVIDIA's `Cuda` programming language as it will be highly relevant to describing our implementation. The interested reader is referred to [273] for details.

The basic unit of work launched on a GPU is a *kernel*, which specifies a function that is executed by many threads. The threads of a kernel are organized into *blocks*, and the group of blocks is organized into a *grid*.

Threads in the same block cooperate, executing in a single-instruction, multiple-thread (SIMT) fashion; all threads in a block execute the same instruction on different data every cycle. When threads in a block follow different data-dependent branches, the block must execute those branches serially, so irregular control flow reduces effective throughput.[5] Threads in a block can cooperate by synchronizing memory accesses. This can either be done by accessing the global memory or through a local cache known as shared-memory, an explicitly managed cache that is much faster to access than the global memory when used correctly.

As threads in the same block are meant to cooperate, the fundamental unit of independent work on a GPU is a *block*. Blocks can be scheduled for execution in any order, and are scheduled for execution on streaming multiprocessors (SM), the fundamental hardware unit on a GPU. An SM is the basic compute engine of the GPU: it contains the arithmetic units that execute instructions and all threads in a block are guaranteed to execute on the same SM. A block remains on an SM until it is completed. The SM is the physical location of the shared memory used by threads in a block, and the number of blocks that can be scheduled onto the same SM is limited by the number of threads in a block, the amount of shared memory requested, and the complexity of the function executed by the block.

Finally, the kernel blocks are organized into a grid. Blocks in the grid are largely independent and the kernel is considered complete once all the blocks in the grid have been executed on the SMs.

Optimizing GPU throughput amounts to ensuring that all blocks in the kernel have sufficient work to continue saturating the GPU. An important part of ensuring blocks can perform work is optimizing memory access. In the CUDA programming model, there are three types of memory: the host (CPU) memory, the device (GPU) global memory, and the aforementioned shared memory within a block. Accessing data on the CPU from the GPU is by far the slowest type of memory access, as it requires physical transfer of data between two devices. Within the device, global memory is large and offers high aggregate bandwidth, but it is still physically separate from an SM's memory; uncached accesses incur hundreds of clock cycles of latency, so performance depends strongly on coalescing accesses across blocks and on having enough active blocks for the scheduler to hide this latency [274].

If threads in the same block repeatedly reuse the same data, e.g. when performing matrix multiplication, it can be worth staging chunks of global memory into shared memory which is physically located on the SM where the block is running. Shared-memory latency can be roughly 100 times faster than global memory, but the physical transfer of the data consumes cycles.

Besides the latency of transferring the shared memory, a further trade-off is that shared

[5] Technically this is true at the level of *warps* not blocks. Warps are groups of 32 threads which are physically executed at the same time by an SM.

memory is scarce, and requesting large amounts of it reduces the number of blocks that can reside concurrently on an SM and thus execute in parallel. Finally, global memory persists across kernel launches, while shared memory lasts only for the lifetime of a block and therefore does not persist between kernel launches.

Jointly optimizing the transfer of data across these different regions of memory as well as maximizing the throughput of a block is essential for achieving good performance.

7.6.2 `Cuda` Implementation

We consider a `Cuda` implementation of Algorithm 5 for problems in the form (7.3) without explicit $\mathcal{C}(x) = d$ constraints[6]. We denote by P_i, A_i, b_i and $\mathcal{K}_i$ the data for the ith program. The jth cone in the ith program is denoted by $\mathcal{K}_{ij}$. As GPUs are optimized for regular workflows with predictable memory access, we assume that (7.3) has been specified using *dense* data matrices. However, we do not assume that every program is structurally the same size.

In contrast to prior work which launches a separate kernel for each step of (7.3) [237,238], we fuse the entire loop Algorithm 5, including the convergence check, into a single kernel. As the fundamental unit that can be scheduled independently by the GPU is a block, we consider an implementation of Algorithm 5 where each program is assigned to a single block. That is to say, all threads in a block cooperate to execute the linear system solve Line 5.4, the cone projection Line 5.5, vector additions, and matrix-vector multiplications.

As Algorithm 5 executes many passes over the data, we consider an architecture where all data and solver state for a single program is loaded into shared memory for the entire duration of the solve. This maximizes the speed with which we can perform the computations for a single program but fundamentally limits the size of the programs we can solve. Moreover, this architecture may reduce the overall amount of parallelism that we can achieve on the GPU as it limits the number of blocks which can be scheduled concurrently on the SM.

We make heavy use of NVIDIA's `cuSolverDx` and `cuBLASDx` libraries, which allow for fusing basic linear algebra functions into larger kernels. These libraries require knowing the size of the data at compile time. Therefore, our current implementation supports programs with $n \in \{2, 4, 8, 16, 32, 64\}$ variables and $m \in \{2, 4, 8, 12, 16, 24, 32, 48, 64\}$ constraints, with further sizes being available for compilation for larger GPUs. Programs which do not fit into these sizes are automatically padded to the next supported size. When solving a heterogeneous batch, a separate kernel is launched for every supported size.

We now turn our attention to the details of the implementation of the two most computationally intense steps Line 5.4 and Line 5.5.

Factoring and Solving the Linear System To execute Line 5.4, we solve (7.24) by computing a Cholesky factorization of the matrix operator. The initial factorizations are done by a call to NVIDIA's `cuSolver potrfBatched` function.

We use NVIDIA's optimized implementation of forward-backward triangular solving via `cuSolverDx` to fuse the solve into the larger kernel. This requires the data matrices and program state to be loaded into shared memory and also requires knowing the size of the

[6]Zero constraints are included in the constraints $\mathcal{A}(x) - b$ block and get a dual variable

Workload	Num instances	CPU (ms)	GPU prep (ms)	GPU solve (ms)	D2H (ms)	Mean iters CPU/GPU	Solve Speedup	End-to-end Speedup
`cg_simple_4`	8522	1830.34	160.44	227.14	213.65	90.9 / 92.6	8.06x	3.04x
`cg_maze_b1` (25k prefix)	25000	5985.10	338.75	672.57	558.23	130.7 / 132.1	8.90x	3.81x
`Maze 30_10_48` vertices and edges	3916	78.25	163.90	21.34	120.62	22.3 / 22.3	3.67x	0.26x
`Maze 30_10_48` edges only	3016	51.34	186.59	10.54	94.43	20.0 / 20.0	4.87x	0.18x
`Maze 30_10_48` vertices only	900	34.94	210.54	13.38	30.51	30.1 / 30.1	2.61x	0.14x

Table 7.8: Performance of the `Cuda` port of `CCosmo` on heterogeneous batches of small programs. The GPU solve time is only the device factorization time plus the solve-kernel time. GPU prep is only host-to-device upload plus device allocation/setup. The end-to-end speedup divides the CPU solve time by GPU prep, GPU solve, and device-to-host (D2H) readback; host packing, host setup, constructor-time factorization, and host finalization are excluded as overhead.

data matrices at compile time. This is the primary technical reason that we support only a fixed number of program sizes.

Projection Onto Cones In the current implementation, we support only the zero set, the nonnegative orthant, bounding boxes, and Lorentz cones. We assume there is a single block of each of the zero set, the nonnegative orthant, and the bounding-box constraints, of sizes $n_{\mathbb{0}}$, $n_{\mathbb{R}_+}$, and $n_{[l,u]}$, respectively. Projection onto these sets is trivially parallelized using strided elementwise passes.

Projection onto large Lorentz cones is done using a blockwise reduction; all threads in the block cooperate to compute the projection. Projection onto small Lorentz cones is done using warpwise reductions.

Convergence Checks We support only optimality checks in the current implementation. The necessary matrix-vector products are computed using NVIDIA's `cuBlasDx` library which provides optimized BLAS implementations for fusing into larger kernels. Again, these require known, compile-time sizes of the program at hand.

Enhancements To avoid introducing more complexity into the kernel code, which may affect GPU occupancy, we do not implement any additional enhancements to the base algorithm Algorithm 5. As our assumption is that programs are small enough to fit in the shared memory of a block, we see in practice that using the unenhanced versions is sufficient.

7.6.3 Results

In Chapter 9 we consider an algorithm for solving GCS instances which involves repeatedly solving a convex program at every vertex and every edge. We also introduce a benchmark of instances in Section 9.3, some of which will have hundreds to thousands of vertices and edges associated to small programs. Here, we test the viability of accelerating the algorithm

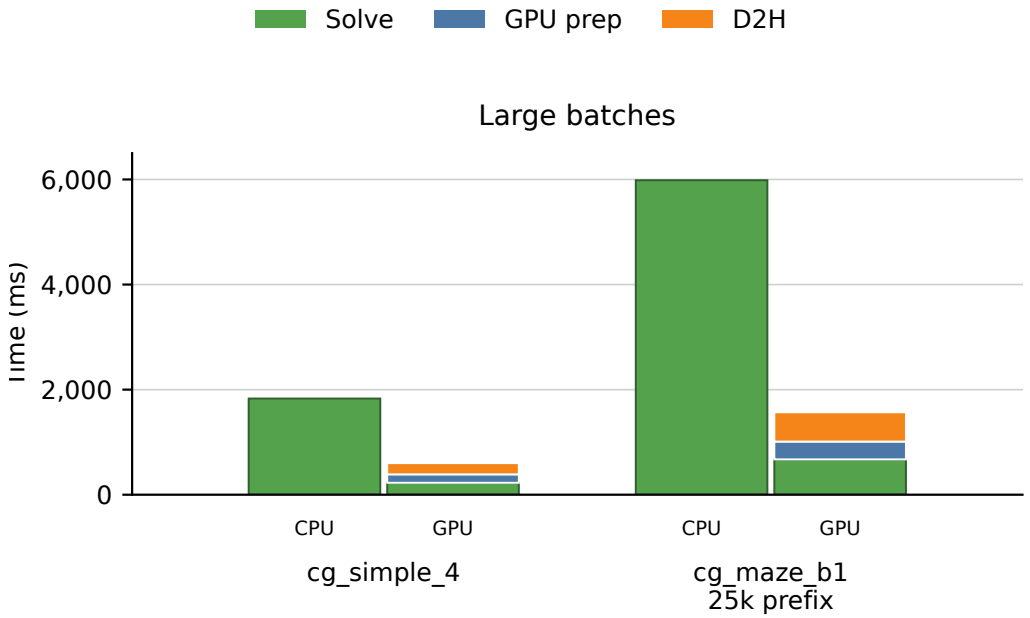


Figure 7.13: Stacked timing view of the CPU and GPU times for the large-batch workloads in Table 7.8. The GPU bar stacks GPU solve time, GPU prep, and device-to-host readback. Even with orchestrating the data between the GPU and CPU, we get substantial speedups on these instances.

in Chapter 9 by parallelizing the solution of the convex programs using the GPU. All experiments in this subsection were run on an NVIDIA GeForce RTX 3060 Laptop GPU (6 GiB VRAM, compute capability 8.6) and an 11th Gen Intel Core i9-11980HK CPU. Our results are summarized in Table 7.8. In short, on large, dense, heterogeneous batches, we achieve close to a $3-9\times$ speedup over solving the batch on a CPU. These results are comparable to those in [237, §4.2] which report similar speedups, albeit in their case they achieve a $5\times$ speedup when comparing parallelism on a GPU against serially solving on the CPU.

In the current implementation, much of the gain for very small programs can be erased by the orchestration of the data transfer between the GPU and the CPU. The work outside the GPU solving kernel erases about 50% of the speedup benefit for batches with large programs and completely erases the benefits for small programs. We provide an in-depth breakdown of the time of the GPU operations in Appendix B.3. Tables Tables B.3 and B.5 indicate we are using only $12-54\%$ of the memory bandwidth to transfer solutions back to the CPU, which is where the most time is lost.

We consider three examples. In all cases, we add a quadratic error cost to each vertex and edge program to ensure that the solution is not trivial[7].

In the first, all 8522 vertex and edge programs of the `cg_simple_4` instances were solved

[7]At the vertices, we use the original program, not the vertex program from the GCS convex relaxation which is much larger.

Family	Num instances	Num vars	Num constraints	A nonzeros	A density	Cone mix
Vertex V1	60	15	50	100	13.3%	$\mathbb{R}_+^{38} \times \mathbb{L}^{12}$
Vertex V2	96	16	58	114	12.3%	$\mathbb{R}_+^{45} \times \mathbb{L}^{13}$
Vertex V3	36	17	66	132	11.8%	$\mathbb{R}_+^{52} \times \mathbb{L}^{14}$
Start vertex	1	5	10	18	36.0%	$\mathbb{O}^4 \times \mathbb{L}^6$
Goal vertex	1	5	10	18	36.0%	$\mathbb{R}_+^4 \times \mathbb{L}^6$
Edge E1	1200	21	26	70	12.8%	$\mathbb{O}^4 \times \mathbb{L}^{22}$
Edge E2	2880	22	27	71	12.0%	$\mathbb{O}^4 \times \mathbb{L}^{23}$
Edge E3	2640	23	28	76	11.8%	$\mathbb{O}^4 \times \mathbb{L}^{24}$
Edge E4	1296	24	29	77	11.1%	$\mathbb{O}^4 \times \mathbb{L}^{25}$
Edge E5	288	25	30	82	10.9%	$\mathbb{O}^4 \times \mathbb{L}^{26}$
Edge E6	14	15	20	52	17.3%	$\mathbb{O}^4 \times \mathbb{L}^{16}$
Edge E7	10	16	21	53	15.8%	$\mathbb{O}^4 \times \mathbb{L}^{17}$

Table 7.9: Families of programs in `cg_simple_4`.

Family	Num instances	Num vars	Num constraints	A nonzeros	A density	Cone mix
Vertex V1	196	15	50	100–108	13.7% mean	$\mathbb{R}_+^{38} \times \mathbb{L}^{12}$
Vertex V2	329	16	58	110–142	13.0% mean	$\mathbb{R}_+^{45} \times \mathbb{L}^{13}$
Vertex V3	101	17	66	132	11.8%	$\mathbb{R}_+^{52} \times \mathbb{L}^{14}$
Start vertex	1	5	10	18	36.0%	$\mathbb{O}^4 \times \mathbb{L}^6$
Goal vertex	1	5	10	18	36.0%	$\mathbb{R}_+^4 \times \mathbb{L}^6$
Edge E1	8257	21	26	70	12.8%	$\mathbb{O}^4 \times \mathbb{L}^{22}$
Edge E2	10376	22	27	71	12.0%	$\mathbb{O}^4 \times \mathbb{L}^{23}$
Edge E3	4926	23	28	76	11.8%	$\mathbb{O}^4 \times \mathbb{L}^{24}$
Edge E4	767	24	29	77	11.1%	$\mathbb{O}^4 \times \mathbb{L}^{25}$
Edge E5	35	15	20	52	17.3%	$\mathbb{O}^4 \times \mathbb{L}^{16}$
Edge E6	11	16	21	53	15.8%	$\mathbb{O}^4 \times \mathbb{L}^{17}$

Table 7.10: Families of programs in the first $25,000$ regularized local programs from `cg_maze_b1`, taken in deterministic vertex-then-edge order.

as a single batch. In Table 7.9, we report the different families of programs the 8522 programs partition into. We note that these programs exhibit heterogeneity at a structural level. Nevertheless, on these instances we achieve about an $8 - 8.9\times$ speedup relative to solving the programs in parallel on the CPU. However, this performance difference drops to only $3 - 3.8\times$ if we include the overhead.

The second batch of instances is from the even larger `cg_maze_b1` instances. This instance has over $50,000$ edge and vertex programs, and so the entire batch is too large to fit on the GPU's VRAM. Instead, we solve the first $25,000$ programs. In Table 7.10, we report the precise breakdown of the different families of problems the $25,000$ problems fall into. Again, we see heterogeneity across the batch, with larger vertex programs, but more complex edge programs. We attain a similar performance increase of almost $9\times$ the CPU baseline, but about half of this performance gain is wiped out if we include the time to transfer data to and from the GPU.

The results on the `cg_simple_4` and `cg_maze_b1` are visualized in Figure 7.13.

Finally, we consider the cost of solving only the edge and vertex programs for a large member from the `Maze` examples. The description of each problem is given in Table 7.11. In this case, the programs are extremely small, less than 5 variables per vertex or edge and only

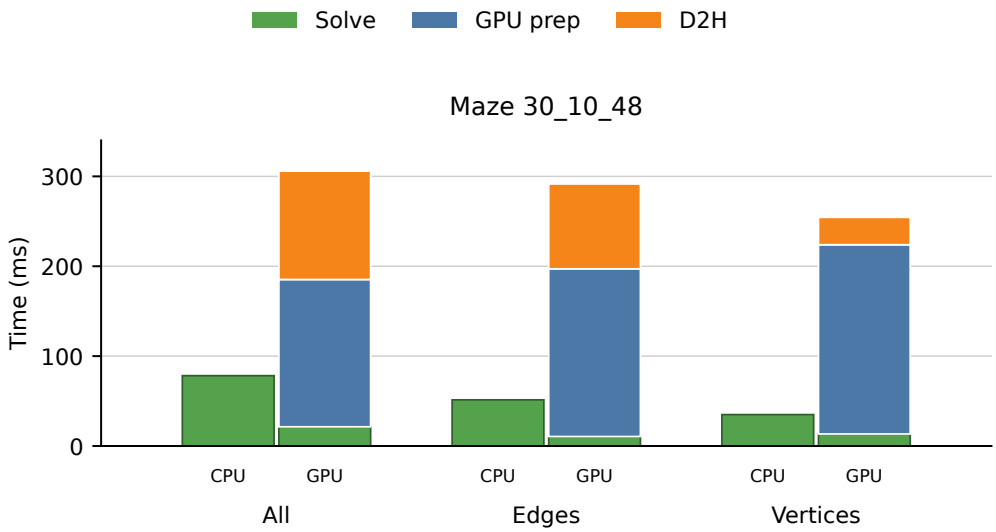


Figure 7.14: Stacked timing view of the CPU and GPU times for the `Maze 30_10_48` workloads in Table 7.8. The GPU bar stacks GPU solve time, GPU prep, and device-to-host readback. The overhead of orchestrating the data drowns out the advantage from solving on the GPU.

Family	Num instances	Num vars	Num constraints	A nonzeros	A density	Cone mix
Vertex V1	898	5	11	13	23.6%	$\mathbb{R}^8_+ \times \mathbb{L}^3$
Boundary vertex	2	5	9	11	24.4%	$\mathbb{O}^2 \times \mathbb{R}^4_+ \times \mathbb{L}^3$
Edge E1	3016	10	2	4	20.0%	$\mathbb{O}^2$

Table 7.11: Families of programs in the `Maze 30_10_48`.

a handful of constraints. In this situation, we obtain a speedup of $2.5 - 5\times$ if we consider only solve time, but are only $0.2\times$ faster (which is a slowdown) if we consider data transfer times. The reason for the discrepancy is that the 3016 edge programs are so small and solved so quickly that the data transfer times dominate.

To see this discrepancy, we also report the time taken to solve the 900 vertex programs separately from the 3016 edge programs. The vertex programs achieve a $2.61\times$ speedup in solve times, with an overall speedup of $0.14\times$ when data transfer times are included, compared to the edge programs which achieve a $0.18\times$ speedup. The results on these instances are visualized in Figure 7.14.

7.7 Conclusion

In this chapter, we introduced `CCosmo`, a family of `C++` implementations of a popular first-order method for solving convex optimization programs. Our implementation is based on

the same base algorithm as `OSQP` and `COSMO.jl`, and contains a broad family of additional heuristic enhancements. Ablation studies demonstrate that the additional enhancements do not conclusively improve the performance of `CCosmo`, and therefore our standard-form implementation achieves comparable solve times to popular open-source solvers. Nonetheless, as `CCosmo` is written in `C++`, it is substantially more portable than the `Julia` solver `COSMO.jl`.

In addition to the standard-form solver, `CCosmo` also introduces a specialized path for programs in a particular matrix-standard form. This form allows `CCosmo` to use a specialized dense matrix linear algebra backend for its most expensive step and we show that this can give benefits even when solving sparse programs.

Finally, we introduced a `Cuda` implementation of `CCosmo` for solving large batches of small programs. We demonstrate that the implementation can reduce solve times by about $3-9\times$, but that orchestrating the data between the CPU and GPU can add substantial overhead that can wipe out much of this gain.

Overall, this chapter contributes a substantial engineering effort stress testing a popular algorithm for solving convex optimization programs in the literature.

7.8 Deferred Proofs

7.8.1 Duality in Convex Quadratic Programs

We prove the duality between (7.1) and (7.2). Without loss of generality, we can assume there is no constraint $\mathcal{C}(x) - d = 0$ since we can simply append these constraints to $\mathcal{A}(x) - b \in \mathcal{K}$.

Proof 7.1. We write (7.1) as

$$\min_x \mathcal{P}(x) + \langle q, x\rangle + \delta_{\mathcal{K}}(\mathcal{A}(x) - b) \tag{7.47}$$

Since $\mathcal{K}$ is closed and convex, Fenchel conjugacy [20, Theorem 13.2] gives

$$\delta_{\mathcal{K}}(z) = \sup_{\lambda} \left(-\langle \lambda, z\rangle + \inf_{w\in\mathcal{K}} \langle \lambda, w\rangle \right). \tag{7.48}$$

Therefore (7.47) becomes

$$\min_x \sup_{\lambda} \left(\mathcal{P}(x) + \langle q, x\rangle - \langle \lambda, \mathcal{A}(x) - b\rangle + \inf_{w\in\mathcal{K}} \langle \lambda, w\rangle \right). \tag{7.49}$$

The corresponding dual function is

$$\begin{aligned} g(\lambda) &= \inf_{w\in\mathcal{K}} \langle \lambda, w\rangle + \inf_x \left(\mathcal{P}(x) + \langle q, x\rangle - \langle \lambda, \mathcal{A}(x) - b\rangle\right) \\ &= \langle \lambda, b\rangle + \inf_{w\in\mathcal{K}} \langle \lambda, w\rangle + \inf_x \left(\mathcal{P}(x) + \langle q, x\rangle - \langle \mathcal{A}^*(\lambda), x\rangle\right). \end{aligned}$$

If this infimum is finite and attained, first-order optimality requires

$$\begin{aligned} &\nabla\mathcal{P}(x) + q - \mathcal{A}^*(\lambda) = 0 \\ \implies &\nabla\mathcal{P}(x) + q = \mathcal{A}^*(\lambda) \end{aligned}$$

Plugging this into the inner objective gives

$$\begin{gathered}\mathcal{P}(x) + \langle q, x\rangle - \langle \mathcal{A}^*(\lambda), x\rangle \\ \mathcal{P}(x) + \langle q, x\rangle - \langle \nabla\mathcal{P}(x) + q, x\rangle \\ \mathcal{P}(x) - \langle \nabla\mathcal{P}(x), x\rangle \\ \mathcal{P}(x) - 2\mathcal{P}(x) \\ -\mathcal{P}(x)\end{gathered}$$

where $\langle \nabla\mathcal{P}(x), x\rangle = 2\mathcal{P}(x)$ by Euler's theorem for homogeneous functions. Therefore

$$g(\lambda) = \langle \lambda, b\rangle - \mathcal{P}(x) + \inf_{w\in\mathcal{K}} \langle \lambda, w\rangle$$

whenever $\nabla\mathcal{P}(x) = \mathcal{A}^*(\lambda) - q$. This gives (7.2). Finally,

$$\inf_{w\in\mathcal{K}} \langle \lambda, w\rangle > -\infty \iff \lambda \in (\mathcal{K}^\infty)^*,$$

so the additional term in the dual objective is finite precisely on the stated dual feasible set. □

7.8.2 Proof of Cone Projection

Proof 7.2. If $\mathcal{D} = \rho^2\mathcal{T}^*\mathcal{T}$ where $\mathcal{T}$ is an automorphism of $\mathcal{K}$, then

$$\begin{aligned}\mathcal{D}_s(\hat{s} - \tilde{s} + u) &= \rho^2\langle \mathcal{T}(\hat{s} - \tilde{s} + u), \mathcal{T}(\hat{s} - \tilde{s} + u)\rangle \\ &= \rho^2\langle \mathcal{T}(\tilde{s}) - \mathcal{T}(\hat{s} + u), \mathcal{T}(\tilde{s}) - \mathcal{T}(\hat{s} + u)\rangle\end{aligned}$$

Letting $z = \mathcal{T}(\hat{s} + u)$ and $\breve{s} = \mathcal{T}(\tilde{s})$, our problem reduces to

$$\begin{gathered}\min_{\breve{s}} \ \|\breve{s} - z\|_2^2 \\ \text{subject to } \mathcal{T}^{-1}(\breve{s}) \in \mathcal{K}\end{gathered}$$

Since $\mathcal{T}$ is an automorphism, $\mathcal{T}^{-1}(\breve{s}) \in \mathcal{K} \iff \breve{s} \in \mathcal{K}$. Therefore, $\breve{s}^* = \Pi_{\mathcal{K}}(z)$. Substituting back into the original variables, we obtain

$$\breve{s}^* = \Pi_{\mathcal{K}}(z) \implies \tilde{s} = (\mathcal{T}^{-1} \circ \Pi_{\mathcal{K}} \circ \mathcal{T})(\hat{s} + u)$$

□

7.8.3 Proof of Proposition 4

We recall our definitions

$$\begin{aligned}\hat{\mathcal{P}} &= \mathcal{P} \circ \mathcal{S}_x, & \hat{q} &= \mathcal{S}_x^*(q), \\ \hat{\mathcal{A}} &= \mathcal{S}_{\mathcal{K}} \circ \mathcal{A} \circ \mathcal{S}_x, & \hat{b} &= \mathcal{S}_{\mathcal{K}}(b), \\ \hat{\mathcal{C}} &= \mathcal{S}_0 \circ \mathcal{C} \circ \mathcal{S}_x, & \hat{d} &= \mathcal{S}_0(d),\end{aligned}$$

and prove Section 7.4.2.

Proof 7.3. Let $\hat{\mathcal{L}}_{\hat{\mathcal{D}}_x,\hat{\mathcal{D}}_s}$ denote the augmented Lagrangian of the ADMM standard-form split of the rescaled problem (7.38) with penalty functions $\hat{\mathcal{D}}_x$ and $\hat{\mathcal{D}}_s$:

$$\hat{\mathcal{L}}_{\hat{\mathcal{D}}_x,\hat{\mathcal{D}}_s}((\hat{x},\hat{s}),(\tilde{\hat{x}},\tilde{\hat{s}}),\hat{u}) = \hat{\mathcal{P}}(\hat{x}) + \langle \hat{q},\hat{x}\rangle + \mathbb{1}(\hat{\mathcal{A}}(\hat{x}) - \hat{b} = \hat{s}) + \mathbb{1}(\hat{\mathcal{C}}(\hat{x}) = \hat{d}) + \mathbb{1}(\tilde{\hat{s}} \in \mathcal{K}) \\ + \hat{\mathcal{D}}_x(\hat{x} - \tilde{\hat{x}}) + \hat{\mathcal{D}}_s(\hat{s} - \tilde{\hat{s}} + \hat{u}).$$

Substituting

$$\begin{aligned} \hat{x} &= \mathcal{S}_x^{-1}(x), & \hat{s} &= \mathcal{S}_{\mathcal{K}}(s), \\ \tilde{\hat{x}} &= \mathcal{S}_x^{-1}(\tilde{x}), & \tilde{\hat{s}} &= \mathcal{S}_{\mathcal{K}}(\tilde{s}), \\ \hat{u} &= \mathcal{S}_{\mathcal{K}}(u) \end{aligned}$$

and the definitions of $\hat{\mathcal{P}}, \hat{q}, \hat{\mathcal{A}}, \hat{b}, \hat{\mathcal{C}}, \hat{d}$ give

$$\hat{\mathcal{P}}(\hat{x}) = \mathcal{P}(x), \qquad \langle \hat{q},\hat{x}\rangle = \langle \mathcal{S}_x^*(q), \mathcal{S}_x^{-1}(x)\rangle = \langle q, x\rangle,$$
$$\mathbb{1}(\hat{\mathcal{A}}(\hat{x}) - \hat{b} = \hat{s}) = \mathbb{1}(\mathcal{S}_{\mathcal{K}}(\mathcal{A}(x) - b - s) = 0) = \mathbb{1}(\mathcal{A}(x) - b = s),$$
$$\mathbb{1}(\hat{\mathcal{C}}(\hat{x}) = \hat{d}) = \mathbb{1}(\mathcal{S}_0(\mathcal{C}(x) - d) = 0) = \mathbb{1}(\mathcal{C}(x) = d),$$
$$\mathbb{1}(\tilde{\hat{s}} \in \mathcal{K}) = \mathbb{1}(\mathcal{S}_{\mathcal{K}}(\tilde{s}) \in \mathcal{K}) = \mathbb{1}(\tilde{s} \in \mathcal{K}),$$

where the last equality uses that $\mathcal{S}_{\mathcal{K}}$ is an automorphism of $\mathcal{K}$. For the penalty terms,

$$\hat{\mathcal{D}}_x(\hat{x} - \tilde{\hat{x}}) = \hat{\mathcal{D}}_x\left(\mathcal{S}_x^{-1}(x - \tilde{x})\right) = \mathcal{D}_x(x - \tilde{x}),$$
$$\hat{\mathcal{D}}_s(\hat{s} - \tilde{\hat{s}} + \hat{u}) = \hat{\mathcal{D}}_s\left(\mathcal{S}_{\mathcal{K}}(s - \tilde{s} + u)\right) = \mathcal{D}_s(s - \tilde{s} + u).$$

Therefore

$$\hat{\mathcal{L}}_{\hat{\mathcal{D}}_x,\hat{\mathcal{D}}_s}((\hat{x},\hat{s}),(\tilde{\hat{x}},\tilde{\hat{s}}),\hat{u}) = \mathcal{L}_{\mathcal{D}_x,\mathcal{D}_s}((x,s),(\tilde{x},\tilde{s}),u),$$

where $\mathcal{L}_{\mathcal{D}_x,\mathcal{D}_s}$ is the augmented Lagrangian of (7.10) with

$$\mathcal{D}_x(x) = \hat{\mathcal{D}}_x(\mathcal{S}_x^{-1}(x)),$$
$$\mathcal{D}_s(s) = \hat{\mathcal{D}}_s(\mathcal{S}_{\mathcal{K}}(s)).$$

Since the two formulations are related by an invertible linear change of variables, minimizing one augmented Lagrangian is equivalent to minimizing the other, and the ADMM iterates correspond exactly under the same change of variables. □

It is worth noting that in exact arithmetic, static scaling can be achieved by simply changing the penalty parameter. However, in finite arithmetic, factoring the matrix

$$\begin{bmatrix} \mathcal{S}_x^{-*} \circ \nabla\mathcal{P} \circ \mathcal{S}_x^{-1} + \mathcal{S}_x^{-*} \circ \mathcal{D}_x \circ \mathcal{S}_x^{-1} & \mathcal{S}_x^{-1} \circ \mathcal{A}^* \circ \mathcal{S}_{\mathcal{K}} & \mathcal{S}_x^{-1} \circ \mathcal{C}^* \circ \mathcal{S}_0 \\ \mathcal{S}_{\mathcal{K}}^* \circ \mathcal{A} \circ \mathcal{S}_{\mathcal{K}} & -\mathcal{S}_{\mathcal{K}}^* \circ \nabla\mathcal{D}_s^{-1} \circ \mathcal{S}_{\mathcal{K}} & 0 \\ \mathcal{S}_0^* \circ \mathcal{C} \circ \mathcal{S}_x^{-1} & 0 & -\delta\mathcal{S}_0^* \circ \mathcal{S}_0 \end{bmatrix}$$

versus

$$\begin{bmatrix} \nabla\mathcal{P} + \mathcal{D}_x & \mathcal{A}^* & \mathcal{C}^* \\ \mathcal{A} & -\nabla\mathcal{D}_s^{-1} & 0 \\ \mathcal{C} & 0 & -\delta I \end{bmatrix}$$

can lead to dramatically different forward and backward error due to round off. This can lead to a difference in solver robustness.

Chapter 8

Solving Tensor-Structured Linear Systems

Numerical linear algebra is the core computational engine behind the vast majority of optimization algorithms. A central bottleneck in many algorithms, including those in the Section 7.3.2.2 is the efficient solution of linear systems. It is therefore unsurprising that a vast amount of research on solving linear systems exists. While many algorithms exist for solving general linear systems, it is often the case that specialized algorithms that target a structured subclass of linear systems can do better [275–278].

In this chapter, we take interest in systems of linear *matrix equations*. All systems we consider in this chapter have some variation of the form

$$AXB + CXD = G. \tag{8.1}$$

Such linear systems and their generalizations arise in a wide variety of contexts including differential equations [279,280] and optics [281] and have been studied extensively [282–285]. A recent survey on the topic is given in [249].

Using the properties of the Kronecker (tensor) product (see Theorem 1), we can transform (8.1) into the more typical form [47, Lemma 4.3.1]

$$\left(\left(B^T \otimes A\right) + \left(D^T \otimes C\right)\right) \mathbf{vec}(X) = \mathbf{vec}(G), \tag{8.2}$$

where $\mathbf{vec}(X)$ denotes the vector created from the matrix X by stacking the columns of X.

While this shows that solving (8.1) is no harder than solving a general linear system, in this chapter we are interested in techniques that actually exploit the fact that our system has this tensor product structure, while also leveraging the particular properties of the matrices at hand.

8.1 Positive Semidefinite Generalized Sylvester Equations

In this section, we consider a method for solving generalized positive semidefinite Sylvester matrix equations of the form

$$P_l X P_r + Q_l X Q_r = G, \tag{8.3}$$

where

$$P_l, Q_l \in \mathbb{S}^m_+, \quad P_r, Q_r \in \mathbb{S}^n_+$$
$$G \in \mathbb{R}^{m \times n}, \quad X \in \mathbb{R}^{m \times n}.$$

Linear systems of this form arise naturally when solving matrix-structure optimization programs using `CCosmo`. They also arise in optics [281], the discretization of partial differential equations [280], and other computational fields.

We describe a direct-factorization based approach to solving (8.3) which involves working with the matrices in pairs (P_l, Q_l) and (P_r, Q_r). By working with these matrices in pairs, we are able to solve dense instances in $\mathcal{O}(m^3 + n^3)$ time and can avoid forming larger Kronecker products of matrices. Additionally, our method will rely only on simple, optimized linear algebra subroutines such as the Cholesky, singular value, and QR decompositions.

8.1.1 Related Work

Applying the Kronecker identity converts (8.3) into a standard linear system with coefficient matrix

$$P_r \otimes P_l + Q_r \otimes Q_l, \tag{8.4}$$

so one immediate baseline is to solve the vectorized system directly [47, Lemma 4.3.1]. This is conceptually simple, but ignores the tensor structure and incurs the cost of factorizing an $mn \times mn$ matrix and also incurs conditioning issues due to multiplying the spectra of the P and the Q. To avoid forming this large matrix, several possible alternative avenues exist.

As (8.3) is a positive semidefinite linear system, a wide variety of iterative Krylov subspace methods can be used such as the conjugate gradient method [245], MINRES [246], or GMINRES [244]. These methods only rely on being able to apply the linear operator on the left-hand side of (8.3) efficiently and so can scale to very large instances of (8.3). Variations of these iterative methods specialized to the structure of (8.3), but not necessarily assuming the positive semidefiniteness of P and Q, have recently been proposed [254,286,287].

While iterative methods are attractive for scaling to enormous instances of (8.3), direct factorization methods are of interest when (8.3) is somewhat small or particularly poorly conditioned. The former case occurs naturally when solving matrix optimization problems in robotics and the latter occurs naturally in the context of solving instances of (8.3) during an interior point method, where the linear system becomes progressively more ill-conditioned as the algorithm iterates [288,289].

To the best of our knowledge, almost all work on direct-factorization approaches to (8.3) has focused on the more general case when P and Q are only assumed to be square. In the case when P_r and Q_l are stably invertible, (8.3) can be converted to

$$Q_l^{-1} P_l X + X Q_r P_r^{-1} = Q_l^{-1} G P_r^{-1}, \tag{8.5}$$

which is known as a Sylvester equation and can be solved using the Bartels-Stewart [283] or Hessenberg-Schur method [284].

Another, similar method converts (8.3) into a pair of Sylvester equations

$$\begin{aligned} P_l R - L Q_r &= G \\ Q_l R + L P_r &= 0, \end{aligned} \tag{8.6}$$

which is also (confusingly) called a generalized Sylvester matrix equation [290]. The solution X is recovered from either L or R by solving $R = XP_r$ or $L = -Q_l X$. Therefore, this method only requires one of Q_l or P_r to be stably invertible rather than both. Software for solving (8.6) is available [291] in LAPACK in the function DTGSYL[1].

Most similar to our method is [292]. In that work, a generalization of the Bartels-Stewart method is presented which directly targets (8.3), again without the positive semidefinite assumption on P and Q. Regularity conditions are given for when the solution exists in this setting, and the method requires computing the QZ decompositions of the pairs (P_l, Q_l) and (P_r, Q_r) [293]. A Fortran implementation is given in [282], with errors corrected by [294].

The main difference in our method compared to these prior works is the specialization to the case when P and Q are positive semidefinite. We will see in this case that we can solve (8.3) directly using only elementary factorizations such as QR, SVD, and the Cholesky decompositions rather than relying on more esoteric factorizations such as the QZ, Schur, and Hessenberg-Schur decompositions.

8.1.2 Conditions for Solvability

We begin by briefly stating the consistency equations for (8.3) to have any solution.

Theorem 20. *A solution to (8.3) exists if and only if*

$$G \perp \{Z \in \mathbb{R}^{m \times n} \mid P_l Z P_r = 0,\ Q_l Z Q_r = 0\} \tag{8.7}$$

in the standard Frobenius norm inner product. The solution is unique if and only if $\{Z \in \mathbb{R}^{m\times n} : P_l Z P_r = 0,\ Q_l Z Q_r = 0\} = \{0\}$

Proof 8.1. Denote by $\mathcal{A}(X) = P_l X P_r + Q_l X Q_r$. Since $\mathcal{A}$ can be expressed as the matrix $P_r \otimes P_l + Q_r \otimes Q_l$ it is positive semidefinite since the Kronecker product of two PSD matrices is PSD as is the sum of two PSD matrices [47].

A solution to (8.3) exists if and only if $G \perp \ker(\mathcal{A})$, where we use the standard matrix inner product $\langle X, Y \rangle = \mathbf{tr}(Y^T X)$.

We characterize $\ker(\mathcal{A})$. Because $\mathcal{A} \succeq 0$, we get that $X \in \ker(\mathcal{A})$ if and only if $\langle \mathcal{A}(X), X \rangle = 0$

$$\mathbf{tr}(X^T P_l X P_r) + \mathbf{tr}(X^T Q_l X Q_r) = 0.$$

Since $P_l \succeq 0$, then $X^T P_l X \succeq 0$ and so $\mathbf{tr}(X^T P_l X P_r) \geq 0$ since it is the inner product of positive semidefinite matrices. The same holds for the second term and therefore each term individually must be 0. Moreover,

$$\mathbf{tr}(X^T P_l X P_r) = \left\| P_l^{1/2} X P_r^{1/2} \right\|_F^2 = 0,$$

[1] https://www.netlib.org/lapack/explore-html/d4/d3b/group__tgsyl_ga96eff9d077e7600c68cd18246ca4cdc3.html#ga96eff9d077e7600c68cd18246ca4cdc3

which implies that $P_l X P_r = 0$ and similarly for $Q_l X Q_r$.

Therefore,

$$\ker(\mathcal{A}) = \{Z \mid P_l Z P_r = 0,\ Q_l Z Q_r = 0\}. \tag{8.8}$$

□

8.1.3 Simultaneous Diagonalizations of Positive Semidefinite Matrices

Our method will crucially rely on the ability to simultaneously diagonalize two positive semidefinite matrices using congruences. We will collect a handful of these results under different assumptions. These results appear in various forms in the literature from the late 1980s through the early 2000s. See [47,275,295] for a more detailed discussion as well as the relationship between these results and symmetric matrix pencils.

The first theorem concerns the simultaneous diagonalization of two positive definite matrices and is the strongest. This result dates back to at least Weierstrass [296]. The theorem states that two positive definite matrices can be simultaneously diagonalized using an invertible congruence and the resulting diagonal matrices are inverses of each other. The statement is adapted from [297] and can also be found in [298] as part of a method for computing the matrix geometric mean between two matrices.

Theorem 21. *Let $P,\ Q \succ 0$. Then there exists an invertible T and a positive diagonal matrix D such that $T^T P T = D$ and $T^T Q T = D^{-1}$.*

In particular, if

$$P = L_P L_P^T, \quad Q = L_Q L_Q^T \tag{8.9}$$

for any invertible square factors L_P and L_Q and

$$L_Q^{-1} L_P = U D V^T, \tag{8.10}$$

then, T can be given by either of the following expressions

$$T \coloneqq L_Q^{-T} U D^{-1/2} = L_P^{-T} V D^{1/2}, \tag{8.11}$$

and

$$T^{-1} \coloneqq D^{1/2} U^T L_Q^T = D^{-1/2} V^T L_P^T. \tag{8.12}$$

One realization of the matrix T and its inverse T^{-1} can be computed using two Cholesky factorizations and one singular value decomposition.

The proof of Theorem 21 is carried out by simply checking that the two expressions for T in (8.11) are equivalent and achieve the stated diagonalization.

Proof 8.2 (of Theorem 21). Since P and Q are positive definite, then the matrices L_P and L_Q in (8.9) exist and are invertible. One instance of the decomposition (8.9) can be computed using the Cholesky decomposition.

To see the equivalence of the two expressions for T in (8.11), note that from (8.10) we have

$$L_Q^{-1} = UDV^T L_P^{-1} \tag{8.13}$$

$$\implies T = \left(L_P^{-T} V D U^T\right) U D^{-1/2} = L_P^{-T} V D^{1/2}, \tag{8.14}$$

where the implication follows from plugging (8.13) into (8.11).

To see T achieves the claimed diagonalization, we expand using the most convenient form of T

$$T^T P T = (D^{1/2} V^T L_P^{-1}) L_P L_P^T (L_P^{-T} V D^{1/2}) = D \tag{8.15}$$

$$T^T Q T = (D^{-1/2} U^T L_Q^{-1}) L_Q L_Q^T (L_Q^{-T} U D^{-1/2}) = D^{-1}. \tag{8.16}$$

□

In the case where $P \succ 0$ but $Q \succeq 0$, we will not be able to diagonalize to D and its inverse, but instead can diagonalize to I and some other matrix D. This result appears in [275, Algorithm 8.7.1].

Theorem 22. *Let $P \succ 0$ and $Q \succeq 0$. Then there exists an invertible T and a nonnegative diagonal matrix D such that $T^T P T = I$ and $T^T Q T = D$.*

In particular, if

$$P = L_P L_P^T \tag{8.17}$$

and

$$L_P^{-1} Q L_P^{-T} = U D U^T, \tag{8.18}$$

then

$$T = L_P^{-T} U, \tag{8.19}$$

and

$$T^{-1} = U^T L_P^T. \tag{8.20}$$

Proof 8.3 (of Theorem 22). We again verify the statement by expanding the expression $T^T P T$ and $T^T Q T$

$$T^T P T = U^T L_P^{-1} L_P L_P^T L_P^{-T} U = I$$

$$T^T Q T = U^T L_P^{-1} Q L_P^{-T} U = U^T U D U^T U = D.$$

□

Notice that if we are in the case that $P,\ Q \succ 0$ as in Theorem 21 we can instead apply Theorem 22. The reason Theorem 21 can be preferable is that it treats P and Q symmetrically. Moreover, it avoids forming the $L_P^{-1}QL_P^{-T}$ which can have a squaring effect on the condition number. It is worth noting that when $P,\ Q \succ 0$, then a factorization of the type Theorem 21 can be recovered from Theorem 22 by post-multiplying the T in Theorem 22 with $D^{-1/4}$.

The case when both P and Q are only positive semidefinite requires a bit more care. In this case, since conjugacy preserves the signature, we will not be able to diagonalize either matrix to strictly positive diagonal. However, we will be able to diagonalize the part orthogonal to the common null space.

Theorem 23. *Let $P,\ Q \in \mathbb{S}_+^n$. Suppose that $\ker(P) \cap \ker(Q)$ has dimension k and let W be an $n \times (n-k)$ matrix with columns forming a basis of $(\ker(P) \cap \ker(Q))^\perp$.*

Then there exists an invertible matrix $\tilde{T}$ and a diagonal matrix D of size $n-k$ such that

$$\tilde{T}^T W^T P W \tilde{T} = D, \quad \tilde{T}^T W^T Q W \tilde{T} = I - D,$$

where the diagonal entries $d_i \in [0, 1]$.

Proof 8.4. Let $\tilde{P} = W^T P W$ and $\tilde{Q} = W^T Q W$. Since W is a basis $(\ker(P) \cap \ker(Q))^\perp$, then $\ker(\tilde{P}) \cap \ker(\tilde{Q}) = \{0\}$. Therefore, $\tilde{P} + \tilde{Q} \succ 0$ and so we can compute

$$\tilde{P} + \tilde{Q} = LL^T.$$

Computing the eigendecomposition

$$L^{-1}\tilde{P}L^{-T} = UDU^T$$

and selecting $\tilde{T} = L^{-T}U$ yields

$$\begin{aligned}\tilde{T}^T\tilde{P}\tilde{T} &= U^T L^{-1}\tilde{P}L^{-T}U = D \\ \tilde{T}^T\tilde{Q}\tilde{T} &= \tilde{T}^T(LL^T - \tilde{P})\tilde{T} = U^TU - D = I - D.\end{aligned}$$

The fact that $d_i \in [0, 1]$ follows from the fact that conjugacy cannot change the signature of $\tilde{Q}$ and so $I - D \in \mathbb{D}_+$.

Choosing $T = W\tilde{T}$ yields the desired diagonalization. Notice that T is *not invertible*, but $\tilde{T}$ is. □

The results of Theorems 21, 22 and 23 are summarized in Tables 8.1, 8.2 and 8.3 respectively. In the table, we also summarize the actual linear algebra subroutines needed to compute the diagonalizing factorizations in each case. We specialize the results to a few common cases such as when the PSD matrices P and Q are presented as Gram matrices $P = A^TA$ and $Q = B^TB$ as well as when $P \in \mathbb{D}_{++}$.

$P \succ 0,\ Q \succ 0$

Structure	Transform	Compute from
P and Q general	If $P = L_P L_P^T$, $Q = L_Q L_Q^T$, and $L_Q^{-1} L_P = U D V^T$, then $T = L_Q^{-T} U D^{-1/2} = L_P^{-T} V D^{1/2}$	Cholesky of P, Cholesky of Q, and 1 SVD
$P = A^T A$ and $Q = B^T B$	If $A = \hat{Q}_A R_A$, $B = \hat{Q}_B R_B$, and $R_B^{-T} R_A^T = U D V^T$, then $T = R_B^{-1} U D^{-1/2} = R_A^{-1} V D^{1/2}$	QR of A, QR of B, and 1 SVD
$P \in \mathbb{D}_{++}$ and Q general	If $P^{-1/2} Q P^{-1/2} = U \Sigma U^T$, then $T = P^{-1/2} U \Sigma^{-1/4}$ gives $T^T P T = \Sigma^{-1/2}$ and $T^T Q T = \Sigma^{1/2}$	1 SVD
$P \in \mathbb{D}_{++}$ and $Q = B^T B$	If $B P^{-1/2} = U \Sigma V^T$, then $T = P^{-1/2} V \Sigma^{-1/2}$ gives $T^T P T = \Sigma^{-1}$ and $T^T Q T = \Sigma$	1 SVD

Table 8.1: Simultaneous diagonalization summary for $P \succ 0$ and $Q \succ 0$

$P \succ 0,\ Q \succeq 0$

Structure	Transform	Compute from
P and Q general	If $P = L_P L_P^T$ and $L_P^{-1} Q L_P^{-T} = U D U^T$, then $T = L_P^{-T} U$	Cholesky of P and 1 SVD
$P = A^T A$ and $Q = B^T B$	If $A = \hat{Q}_A R_A$ and $R_A^{-T} B^T B R_A^{-1} = U D U^T$, then $T = R_A^{-1} U$	QR of A and 1 SVD
$P \in \mathbb{D}_{++}$ and Q general	If $P^{-1/2} Q P^{-1/2} = U D U^T$, then $T = P^{-1/2} U$	1 SVD
$P \in \mathbb{D}_{++}$ and $Q = B^T B$	If $B P^{-1/2} = U \Sigma V^T$, then $T = P^{-1/2} V$, giving $T^T Q T = \Sigma^2$	1 SVD

Table 8.2: Simultaneous diagonalization summary for $P \succ 0$ and $Q \succeq 0$

8.1.4 Solving the Equation

Equipped with the simultaneous diagonalization results of Section 8.1.3, we can now state a direct factorization procedure for solving (8.3). The method first compresses away the common nullspaces of the pairs (P_l, Q_l) and (P_r, Q_r), then simultaneously diagonalizes the compressed pairs, and finally solves the resulting diagonal equation entrywise. The final solution is obtained by reversing the diagonalizing transform. The procedure is formalized in Algorithm 6.

Line 6.2 may be implemented with a rank-revealing QR factorization of $P_i + Q_i$, since $\ker(P_i + Q_i) = \ker(P_i) \cap \ker(Q_i)$ for positive semidefinite matrices. If the common kernel is already known to be trivial, one may simply take $W_i = I$. Line 6.4 can be carried out using any of the constructions in Theorems 21, 22 and 23. In implementation, it is preferable to store only the triangular, unitary, and diagonal factors needed to apply T_i and T_i^T, rather than forming T_i explicitly. Accordingly, line Line 6.7 amounts to triangular solves, unitary transformations, and, when present, diagonal scalings.

The proof that Algorithm 6 solves (8.3) is given next.

$P \succeq 0,\ Q \succeq 0$

Structure	Transform	Compute from
P and Q general	Compute W, a basis for the range $P+Q$ from a QR. Let $\tilde{P} = W^T P W$, $\tilde{Q} = W^T Q W$, $\tilde{P} + \tilde{Q} = LL^T$, and $L^{-1}\tilde{P}L^{-T} = UDU^T$, then $T = W\tilde{T}$ with $\tilde{T} = L^{-T}U$	QR of $P+Q$, Cholesky of $\tilde{P}+\tilde{Q}$, and 1 SVD
$P = A^T A$ and $Q = B^T B$	Let W come from a QR of $\begin{bmatrix} A^T & B^T \end{bmatrix}$. If $\begin{bmatrix} AW \\ BW \end{bmatrix} = \hat{Q}L^T$ and $L^{-1}(AW)^T(AW)L^{-T} = UDU^T$, then $T = WL^{-T}U$	QR of $\begin{bmatrix} A^T & B^T \end{bmatrix}$, QR of $\begin{bmatrix} AW \\ BW \end{bmatrix}$, and 1 SVD
$P \in \mathbb{D}_+$ and Q general	Let W span $(\ker(P) \cap \ker(Q))^\perp$. If $\tilde{P} = W^T P W$, $\tilde{Q} = W^T Q W$, $\tilde{P} + \tilde{Q} = LL^T$, and $L^{-1}\tilde{P}L^{-T} = UDU^T$, then $T = WL^{-T}U$	QR of $P+Q$, Cholesky of $\tilde{P}+\tilde{Q}$, and 1 SVD
$P \in \mathbb{D}_+$ and $Q = B^T B$	Let W come from a QR of $\begin{bmatrix} P^{1/2} & B^T \end{bmatrix}$. If $\begin{bmatrix} P^{1/2}W \\ BW \end{bmatrix} = \hat{Q}L^T$ and $L^{-1}W^T P W L^{-T} = UDU^T$, then $T = WL^{-T}U$	QR of $\begin{bmatrix} P^{1/2} & B^T \end{bmatrix}$, QR of $\begin{bmatrix} P^{1/2}W \\ BW \end{bmatrix}$, and 1 SVD

Table 8.3: Simultaneous diagonalization summary for $P \succeq 0$ and $Q \succeq 0$

Proposition 5. *Let X denote the matrix returned by Algorithm 6, assuming it succeeds. If (8.3) is consistent, then X is a solution to (8.3).*

Proof 8.5 (of Proposition 5). Since W_l and W_r are bases for $(\ker(P_i) \cap \ker(Q_i))^\perp$, then we have that

$$P_l X P_r + Q_l X Q_r = P_l W_l W_l^T X W_r W_r^T P_r + Q_l W_l W_l^T X W_r W_r^T Q_r.$$

Letting

$$\tilde{X} \coloneqq W_l^T X W_r, \tag{8.21}$$

substituting (8.21) into (8.3), and multiplying on the left by W_l^T and on the right by W_r yields

$$W_l^T P_l W_l \tilde{X} W_r^T P_r W_r + W_l^T Q_l W_l \tilde{X} W_r^T Q_r W_r = W_l^T G W_r \tag{8.22}$$

$$\tilde{P}_l \tilde{X} \tilde{P}_r + \tilde{Q}_l \tilde{X} \tilde{Q}_r = \tilde{G}, \tag{8.23}$$

where $\tilde{G} \coloneqq W_l^T G W_r$.

Since T_l and T_r are invertible, multiplying the reduced equation on the left by T_l^T and on the right by T_r, and substituting $\tilde{X} = T_l T_l^{-1} \tilde{X} T_r^{-T} T_r^T$ gives

$$T_l^T \tilde{P}_l T_l T_l^{-1} \tilde{X} T_r^{-T} T_r^T \tilde{P}_r T_r + T_l^T \tilde{Q}_l T_l T_l^{-1} \tilde{X} T_r^{-T} T_r^T \tilde{Q}_r T_r = T_l^T \tilde{G} T_r \tag{8.24}$$

$$D_{P_l} \hat{X} D_{P_r} + D_{Q_l} \hat{X} D_{Q_r} = \hat{G}, \tag{8.25}$$

Algorithm 6: Direct factorization method for (8.3)

Input: $P_l, Q_l \in \mathbb{S}^m_+$, $P_r, Q_r \in \mathbb{S}^n_+$, and $G \in \mathbb{R}^{m\times n}$.
Output: A matrix X satisfying (8.3) if it exists.

6.1 **for** $i \in \{l, r\}$ **do**
6.2 Compute an orthonormal matrix W_i whose columns form a basis for $(\ker(P_i) \cap \ker(Q_i))^\perp$
6.3 Set $\tilde{P}_i \leftarrow W_i^T P_i W_i$ and $\tilde{Q}_i \leftarrow W_i^T Q_i W_i$
6.4 Compute an invertible matrix T_i and diagonal matrices D_{P_i} and D_{Q_i} such that $T_i^T \tilde{P}_i T_i = D_{P_i}$ and $T_i^T \tilde{Q}_i T_i = D_{Q_i}$
6.5 **if** $G \neq W_l W_l^T G W_r W_r^T$ **then**
6.6 report that (8.3) is inconsistent and stop
6.7 Set $\hat{G} \leftarrow T_l^T W_l^T G W_r T_r$
6.8 **for** *each entry* (i, j) *of* $\hat{G}$ **do**
6.9 $d_{ij} \leftarrow d_{P_l,i} d_{P_r,j} + d_{Q_l,i} d_{Q_r,j}$
6.10 **if** $d_{ij} > 0$ **then**
6.11 $\hat{x}_{ij} \leftarrow \dfrac{\hat{g}_{ij}}{d_{ij}}$
6.12 **else**
6.13 **if** $\hat{g}_{ij} \neq 0$ **then**
6.14 report that (8.3) is inconsistent and stop
6.15 Set $\hat{x}_{ij} \leftarrow 0$
6.16 Set $X \leftarrow W_l T_l \hat{X} T_r^T W_r^T$ and return X

where

$$\hat{X} := T_l^{-1} \tilde{X} T_r^{-T}, \ \hat{G} := T_l^T \tilde{G} T_r. \tag{8.26}$$

We have used the fact that T_i simultaneously diagonalizes $\tilde{P}_i$ and $\tilde{Q}_i$.

Because D_{P_l}, D_{Q_l}, D_{P_r}, and D_{Q_r} are diagonal, this equation decouples entrywise as

$$d_{P_l,i} \hat{x}_{ij} d_{P_r,j} + d_{Q_l,i} \hat{x}_{ij} d_{Q_r,j} = \hat{g}_{ij} \tag{8.27}$$

$$(d_{P_l,i} d_{P_r,j} + d_{Q_l,i} d_{Q_r,j}) \hat{x}_{ij} = \hat{g}_{ij}. \tag{8.28}$$

In the case that $(d_{P_l,i} d_{P_r,j} + d_{Q_l,i} d_{Q_r,j}) = 0$, (8.27) is inconsistent if $\hat{g}_{ij} \neq 0$, which is Line 6.14. Otherwise, we can set $\hat{x}_{ij}$ arbitrarily and so we can choose it to be 0. If $(d_{P_l,i} d_{P_r,j} + d_{Q_l,i} d_{Q_r,j}) \neq 0$, then

$$\hat{x}_{ij} = \frac{\hat{g}_{ij}}{d_{P_l,i} d_{P_r,j} + d_{Q_l,i} d_{Q_r,j}}. \tag{8.29}$$

This is the entrywise formula used in Algorithm 6 to construct $\hat{X}$.

Recovering $\tilde{X}$ and X from (8.26) and (8.21) gives

$$\tilde{X} = T_l \hat{X} T_r^T \tag{8.30}$$

$$X = W_l \tilde{X} W_r^T, \tag{8.31}$$

which is the matrix returned by the algorithm. By construction this satisfies

$$W_l^T(P_l X P_r + Q_l X Q_r)W_r = W_l^T G W_r$$

Finally, since the equations must be consistent, otherwise the algorithm would not have returned an answer. Therefore, we have that $W_l W_l^T G W_r W_r^T = G$ since $G \perp \{Z \in \mathbb{R}^{m\times n} \mid P_l Z P_r = 0,\ Q_l Z Q_r = 0\}$ by Theorem 20. This concludes the proof. □

8.1.5 Results

We compare the direct factorization methods of Algorithm 6 against two baselines

1. We lift (8.3) to the Kronecker system (8.4) and compute a triangular factorization of (8.4). When (8.4) is positive definite a Cholesky factorization is used, with an approximate minimum degree (AMD), fill-reducing order used if the P and Q matrices are sparse. If (8.4) is positive semidefinite a pivoted LU decomposition is used.

2. We use the algorithm described in [282] which makes no assumptions on the structure of P and Q in (8.3) besides that the system has a unique solution. In the positive semidefinite case, we do not compare to this method.

Frequently, linear systems are factored once and solved for many right-hand sides. In these cases, the initial cost of the factorization can be amortized over many solves. Therefore, in our comparisons we measure both the setup time and the solve time. The setup time is the amount of time needed to assemble all matrices in a given method and compute any factorizations. The solve time is the amount of time needed to actually perform all triangular solves and backsubstitutions. All experiments are performed on an 11th Gen Intel Core i9-11980HK CPU with 64 GiB RAM and times are reported as the median over 10 runs.

8.1.5.1 Dense Square Matrices

We begin by considering instances of (8.3) where both the left and right factors are the same size $P_l, P_r, Q_l, Q_r \in \mathbb{S}^n$. We consider

$$n \in \{8, 12, 16, 24, 32, 40, 48, 56, 64\}.$$

We compare four random families for each of the left pairs and the right pairs. This results in 16 experiments per size. For each experiment, we choose a target condition number κ and select eigenvalues $\lambda_i = \kappa^{-\frac{n-i}{n-1}}$ so that $\lambda_1 = \lambda^{-1}$ and $\lambda_n = 1$. We also generate the vector $\nu = \alpha\lambda$ where α is drawn uniformly at random from $[0.5, 2]$. We then generate the following matrices

1. Dense/Dense: P and Q are generated by drawing random unitary matrices U and V and setting $P = U^T \operatorname{diag}(\lambda) U$ and $Q = V^T \operatorname{diag}(\nu) V$.

2. Gram/Gram: We choose $d < n$ and draw a random unitary matrix U. We keep its first $r := n - d$ columns as the matrix U_r. We draw two more random $r \times r$ unitaries V_A and V_B and form $A = \operatorname{diag}(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_r}) V_A U_r^T$ and $B = \operatorname{diag}(\sqrt{\nu_1}, \ldots, \sqrt{\nu_r}) V_B U_r^T$. Set $P = A^T A$ and $Q = B^T B$. In the experiments, we work directly with A and B. We do not use the algorithm from [282] for this case, since the solutions are non-unique.

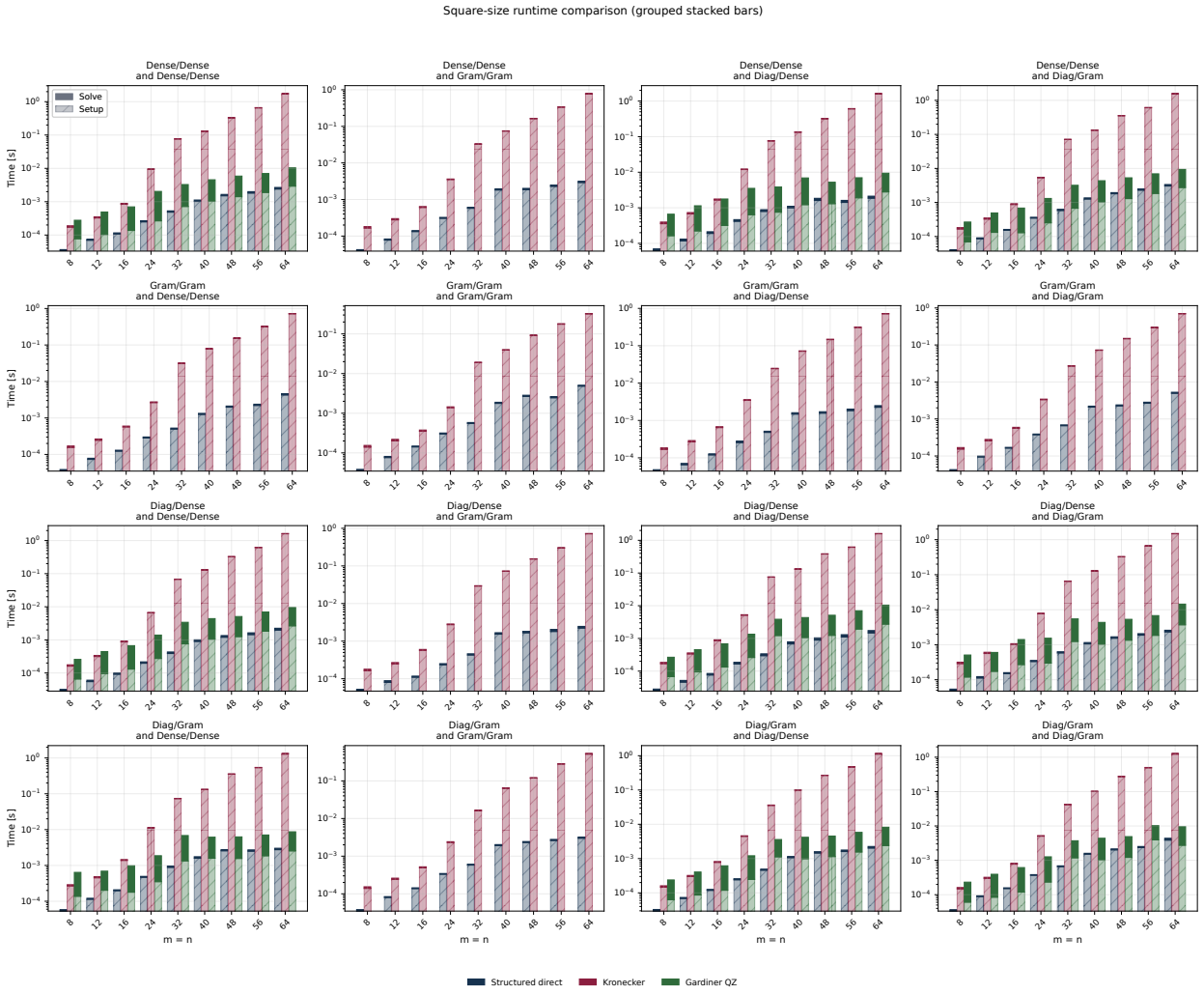


Figure 8.1: Comparison of the proposed method (in blue) against the method of [282] (in green) and Kronecker vectorization (in red). The hatched portion denotes setup time and the solid portion denotes solve time.

3. Diag/Dense: P is chosen as $\text{diag}(\lambda)$ and Q is generated as in the Dense/Dense case.

4. Diag/Gram: P is chosen as $\text{diag}(\lambda)$, $B = \text{diag}(\nu)U$ where U is a random unitary and $Q = B^T B$.

We generate our matrices G by choosing $X_\star$ with each of its entries drawn i.i.d. from a standard normal distribution and computing $G = P_l X_\star P_r + Q_l X_\star Q_r$.

In the Dense/Dense case, we will always use the diagonalizing factorization from Theorem 23 in every case besides when $P \in \mathbb{D}_{++}$ where we will use the special case outlined in Table 8.1. This choice is justified in Section 8.1.5.3.

Our results are summarized in Figure 8.1. Across all tested families, the proposed direct method is consistently faster than the Kronecker baseline, with two or three orders of magnitude separation in total wall-clock time once $n \geq 64$. This is unsurprising as the proposed method scales as $\mathcal{O}(n^3)$ while the Kronecker baseline scales as $\mathcal{O}(n^6)$. We also consistently

outperform the algorithm from [282] due to the fact that we are leveraging the positive semidefinite nature of our matrices. For example, when all four matrices are dense and $n = 64$, our method requires a median of 2.67ms to solve the problem, while the algorithm from [282] requires a median of 10.37ms, and the Kronecker method requires 1.76s to solve.

Further speedups are attained by exploiting the Gram and diagonal structures. For example, when the left pair is Diag/Dense and the right pair is Diag/Gram, our method requires only 2.59ms, compared to 14.43ms for [282], and 1.50s for the Kronecker method.

In all cases, the total solve time is dominated by the initial factorization. However, it is worth noting that the algorithm from [282] actually takes *less* time to perform the initial factorization than both the Kronecker baseline and our method when both matrices in a pair are dense. This is because it is only reducing the left and right factors to a quasi-triangular form rather than diagonal as in our method. This cost saving during factorization is not justified since the time to solve after the factorization is substantially increased. Indeed, the Kronecker baseline takes less time during the solve phase than the method in [282] and therefore could become competitive should a very large number of right-hand sides need to be solved. By contrast, our method is the fastest method for a single solve, where both factorization and actual solving need to occur, and for subsequent solves where the cost of the initial factorization is amortized.

8.1.5.2 Sparse Comparison

The main reason that both our method and [282] beat the Kronecker baseline is due to the different orders of growth $\mathcal{O}(n^3)$ versus $\mathcal{O}(n^6)$. While this leads to separation in the dense case, when the matrices P and Q are sparse, the Kronecker formulation can take advantage of sparse symmetric factorization methods that our proposed method cannot. Therefore, in this section we interrogate the cross-over point; at what scales and densities is our dense method comparable to a sparse method?

We consider sparse, square positive definite matrices with

$$n \in \{16, 24, 32, 48, 96, 128\}$$

and with an expected density of

$$\{0.5\%, 1\%, 2\%, 5\%, 10\%, 20\%\},$$

for the off-diagonal entries.

We consider both a banded sparsity pattern and a completely random sparsity pattern. These represent different extremes for sparse linear algebra. The banded matrix is chordal, and therefore a Cholesky factorization experiences no fill-in, which is optimal for a sparse triangular factorization [299]. The latter patterns can exhibit significant fill-in, and so the advantages of a sparse factorization are less pronounced.

To generate a completely random sparsity pattern each entry in the upper triangle of the matrix was included with probability equal to the expected density. For the banded entries, the bandwidth b was chosen so that the total number of entries of the matrix was approximately the target density. Each nonzero off-diagonal entry was given a value drawn uniformly at random from $[-0.5, -0.05]$ and the diagonal entry P_{ii} for a given row was set

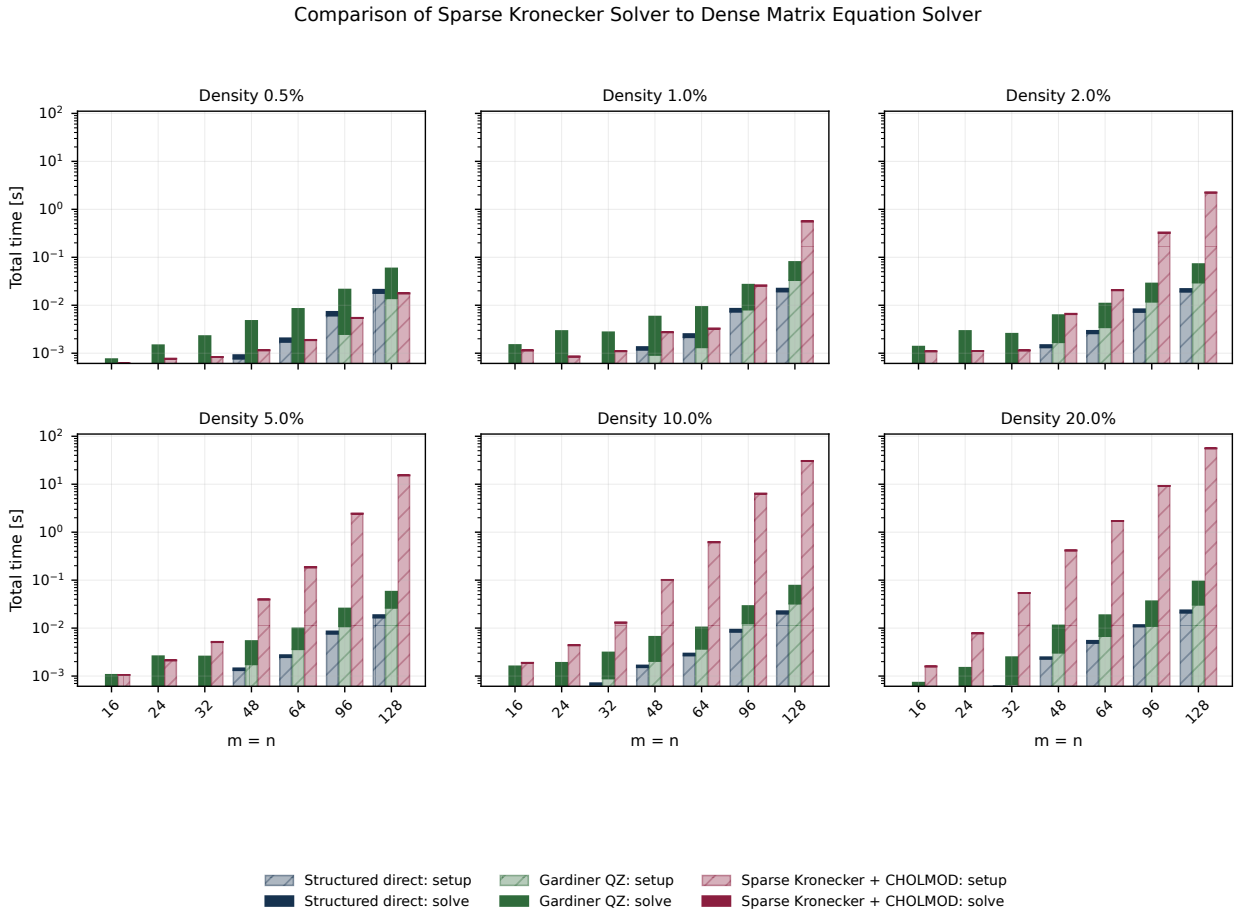


Figure 8.2: Comparison on random sparsity patterns of solving (8.3) using a sparse factorization of the Kronecker system (8.4) versus the dense matrix equation solvers. For the sparse Kronecker baseline, setup includes sparse Kronecker assembly and sparse Cholesky factorization.

to be $P_{ii} = 1 + \sum_{j \neq i} |P_{ij}|$. The Q matrices were generated in the same manner. We use a fill-reducing, permuted Cholesky factorization to factorize the sparse Kronecker operator.

We summarize our results in Figures 8.2 and 8.3. For small matrices, we expect to see limited benefits to using sparse linear algebra. Indeed, this is the case. With $n < 48$ our dense method outperforms all instances of the sparse Kronecker formulation. This is in contrast to the algorithm from [282] which never beats the Kronecker formulation on extremely sparse instances due to long solve times and only begins to be competitive with the baseline when the density is more than 2%. Once the matrices get even modestly dense, both our method and [282] begin outperforming the Kronecker baseline.

8.1.5.3 Choice of Simultaneous Diagonalization

When both P and Q are positive definite, the chapter describes three viable diagonalization strategies: diagonalization to D and its inverse in Theorem 21 which we call the Williamson-style decomposition, the transformation in Theorem 22 which we call the whitening trans-

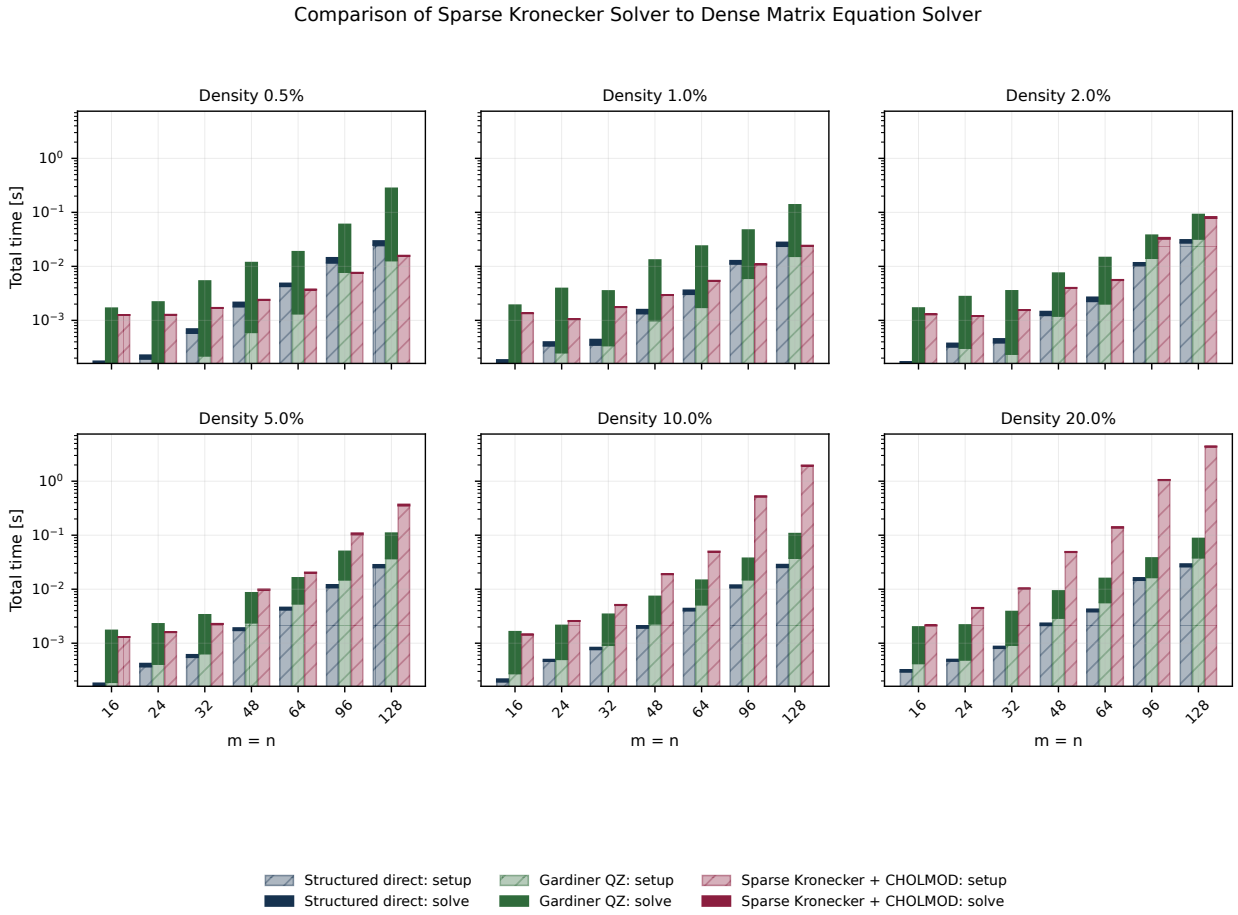


Figure 8.3: Comparison on banded sparsity patterns of solving (8.3) using a sparse factorization of the Kronecker system (8.4) versus the dense matrix equation solvers. For the sparse Kronecker baseline, setup includes sparse Kronecker assembly and sparse Cholesky factorization.

formation, and the transformation in Theorem 23 without compression which we call the average whitening transformation.

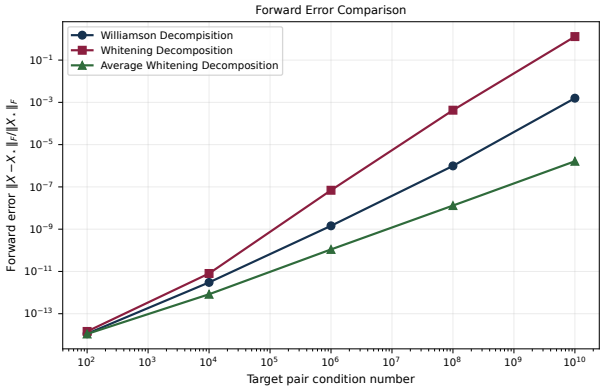
All three require different numbers of factorizations and have different numerical properties. We are interested in both the runtime and accuracy of each method. To compare the accuracy, we compute a solution X_s, compute the relative forward error

$$\frac{\|X_s - X_\star\|}{\|X_\star\|}, \tag{8.32}$$

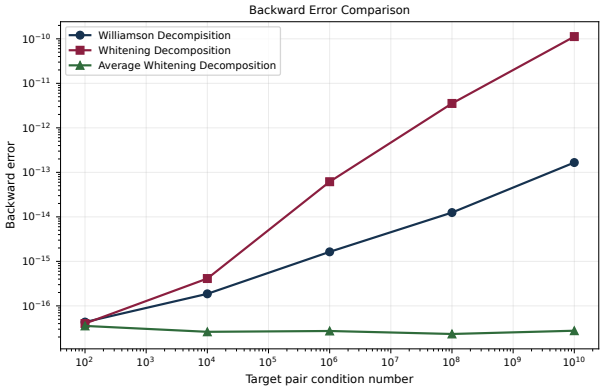
and relative backward error

$$\frac{\|P_l X_s P_r + Q_l X_s Q_r - G\|}{\max\left(1, \|P_l\|\,\|X_s\|\,\|P_r\| + \|Q_l\|\,\|X_s\|\,\|Q_r\| + \|G\|\right)}, \tag{8.33}$$

which measure how well we recover the original solution and how accurately our solution solves the equation.



(a) Forward error for the three diagonalization strategies on the dense SPD case $P_l, Q_l, P_r, Q_r \succ 0$. The factorization from Theorem 23 is the most accurate on the ill-conditioned instances considered here.



(b) Backward error for the same dense SPD comparison. All three methods are backward stable on this experiment, but the factorization from Theorem 23 remains the best numerically across the full conditioning sweep.

We compared all three on dense positive definite instances with $m = n = 64$ and target pair condition numbers

$$\{10^2, 10^4, 10^6, 10^8, 10^{10}\}.$$

Matrices are generated in the same manner as the Dense/Dense setup in Section 8.1.5.1. The forward-error, backward-error, and timing results are shown in Figures 8.4a, 8.4b and 8.5.

In Figure 8.5, we see that the decomposition from Theorem 21 is the slowest to compute its factorization, which is unsurprising given that it must compute an additional Cholesky compared to the methods from Theorem 22 and Theorem 23 (there is no need to compute the QR of $P + Q$ in Theorem 23 since both P and Q are full rank). Since all three methods structurally reduce the problem to the same diagonal form, all have essentially the same solve time.

The largest difference comes in the accuracy of each method. Though all three methods are quite stable in backward error, staying below 10^{-10} precision even when the condition number is 10^{10}, the decomposition from Theorem 23 shows remarkably better precision, managing to stay at essentially machine precision the entire time. This is due to the fact that even if P and Q are poorly conditioned, $P + Q$ is generically not and so the Cholesky factorization is substantially more stable. Conversely, the worst performing factorization is from Theorem 22, which suffers from the effect of computing the Cholesky of P and then an SVD of the poorly scaled matrix $L_P^{-T} Q L_P^{-1}$.

Meanwhile, the forward error is markedly worse as the condition number increases for all three methods. Nevertheless, we see the same rankings of the methods. The factorization from Theorem 23 is still the most stable, followed by the factorization of Theorem 21, and finally Theorem 22.

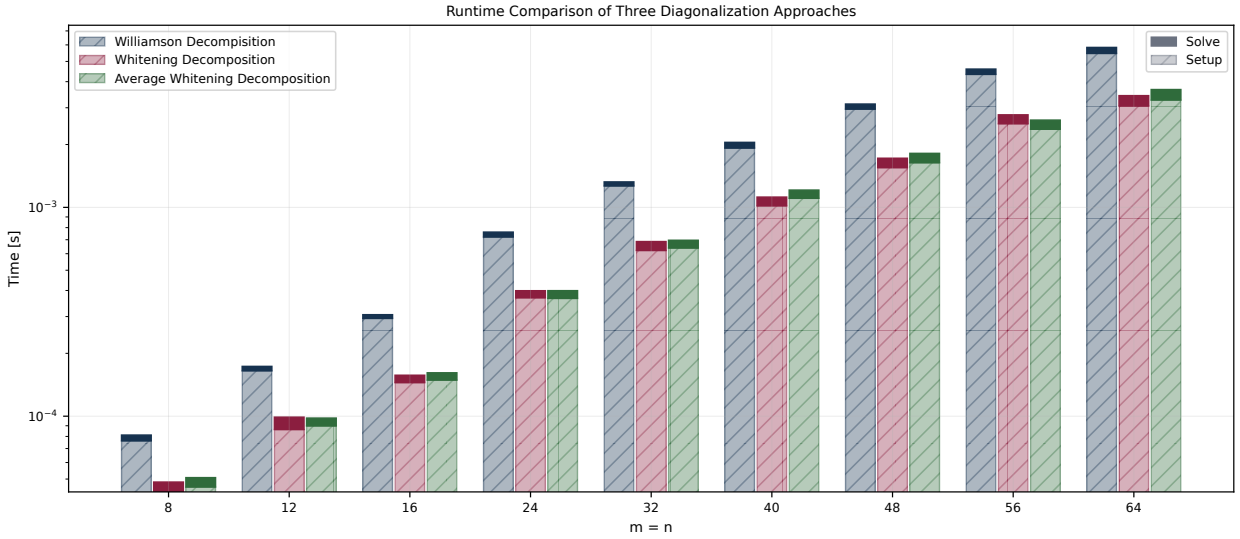


Figure 8.5: Stacked setup and solve times for the same dense SPD case, at fixed target condition number $\kappa = 10^4$. The difference is dominated by setup, while solve times are nearly identical.

8.2 Solving Poset Structure Least Squares Problems

In this section, we consider a different matrix equation arising in the context of a structured optimization problem. This matrix exhibits a rich combinatorial structure that interacts in a non-trivial manner with the tensor product nature of the linear system. We begin by formally introducing the problem.

A pervasive constraint in optimization problems is a structural requirement on the solution. Perhaps the most basic optimization problem with sparsity requirements one can consider is the sparse least-squares problem

$$\min \ \|\mathcal{A}(x) - b\|_2^2 \tag{8.34}$$

$$\text{subject to } x \in \mathcal{S}, \tag{8.35}$$

where $\mathcal{A}$ is a linear map and $\mathcal{S}$ is some sparsity pattern on the vector x.

For example, problems of the form (8.34) arise in controller synthesis [300,301]. Such constraints often appear as lower-triangular structure on a matrix variable X encoding causality; a controller can depend only on past measurements, not future states. In decentralized settings, the constraint must also encode information flow through the network; the control action at a node a cannot depend on information from a node b unless b can communicate with a.

In this section, we are interested in sparsity patterns that arise from modelling information flow over a network. We model this structure by a directed acyclic graph (DAG), or equivalently a partially ordered set (poset), whose edges represent forward flow of information in time and whose topology records which nodes can communicate. In the present setting, that information structure is imposed on data matrices A, B, C and on the unknown matrix X.

Concretely, we are interested in the problem

$$\min \ \|A - CXB\|_F^2 \tag{8.36}$$
$$\text{subject to } X \in \text{Inc}(\mathcal{P}), \tag{8.37}$$

where $A,\ B,\ C,\ X$ are all constrained to be block lower triangular, and the sparsity pattern on the lower triangular part is the transitive closure of a DAG. Formally, we require that $A,\ B,\ C$, and X be blockwise in the incidence algebra of a given poset $\mathcal{P}$, whose definition will be reviewed in Section 8.2.1.

Problems of the form (8.34) arise naturally when designing linear controllers for networked linear dynamical systems, i.e. systems of the form

$$x_{k+1} = Fx_k + Gu_k + Ww_k \tag{8.38}$$
$$y_k = Hx_k + Vw_k, \tag{8.39}$$

where F, G, W, H, V are constrained to have the aforementioned sparse lower triangular structure. Controllers with the same sparsity pattern are compatible with the network structure; they can be implemented in a decentralized manner without additional communication between nodes. These controllers are known as poset-causal controllers [302] and prior work has shown that in the case of linear dynamics with quadratic costs, the optimal controller for (8.38) with the sparsity constraint is linear [303]. Moreover, Youla domain techniques relying on quadratic invariance [300] or Systems Level Synthesis [301] enable posing the search for linear, poset-causal controllers as a structurally constrained least squares program of the form (8.36), albeit in an infinite dimensional Hilbert space rather than the finite dimensional setting considered here.

In principle, solving (8.34) and by extension (8.38) is not hard. The sparsity constraint can be written as $x_i = 0$ for i in some index set and so (8.34) is simply an equality constrained quadratic program. The solution of such programs is standard and amounts to solving a linear system [240, Chpt. 16.1].

We will show that the optimality conditions of (8.36) are a matrix linear system. As was the case for the matrix systems considered in Section 8.1, one can in principle apply the Kronecker vectorization trick or iterative methods to solve this linear system. However, those approaches tend to obscure the block structure of the solution. Understanding the structure of the solution is especially important in control applications, which motivates direct solution techniques in the vein of [302,304,305] that exploit the order-theoretic structure to produce efficient and interpretable solutions to (8.36).

In this section, we will show that solving (8.36) amounts to solving a highly structured, sparse matrix linear system. In particular, we will give

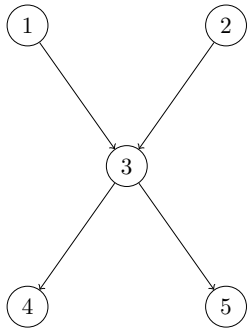
- A characterization of the optimality conditions as a sparse matrix linear system, with sparsity pattern related to the *poset of intervals* of $\mathcal{P}$ (Theorem 24).
- An identification of a key separability property among the block variables of the linear system of Theorem 24 which increases the sparsity after one step of elimination is performed (Theorem 25).

- An explicit, sequential elimination strategy for solving the linear system of Theorem 24. When $\mathcal{P}$ is a multitree (defined in Subsubsection 8.2.3.3) this elimination strategy leads to progressive sparsification of the linear system (Theorem 26), reducing the number of operations required to solve the system.

8.2.1 Preliminaries: Posets and Graphs of Linear Systems

In this section, we formally introduce the order-theoretic language that we will use throughout the remainder of this section. We also recall some basic facts about solving linear systems via elimination.

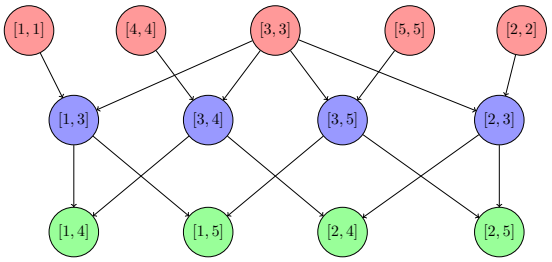
8.2.1.1 Posets



(a) The Hasse diagram of the poset in Example 17.

$$\begin{bmatrix} X_{11} & 0 & 0 & 0 & 0 \\ 0 & X_{22} & 0 & 0 & 0 \\ X_{31} & X_{32} & X_{33} & 0 & 0 \\ X_{41} & X_{42} & X_{43} & X_{44} & 0 \\ X_{51} & X_{52} & X_{53} & 0 & X_{55} \end{bmatrix}$$

(b) A block matrix $X \in \mathrm{Inc}(\mathcal{P})$.



(c) The Hasse diagram of $\mathrm{Int}(\mathcal{P})$ for the poset in Figure 8.6a.

Figure 8.6: An example of a poset, a block matrix X in $\mathrm{Inc}(\mathcal{P})$, and the associated poset of intervals. Notice that the block X_{ij} is in one-to-one correspondence with the interval $[j, i] \in \mathrm{Int}(\mathcal{P})$. The order on $\mathrm{Int}(\mathcal{P})$ will define a natural elimination order for the matrix linear system associated with (8.36).

Definition 46. *A* partially ordered set *(poset)* $\mathcal{P} = (\mathcal{S}, \sqsubseteq)$ *is a set* $\mathcal{S}$ *and a binary relation* $\sqsubseteq$ *satisfying:*

1. $i \sqsubseteq i$ *(reflexivity)*
2. $i \sqsubseteq j$ *and* $j \sqsubseteq i$ *implies that* $i = j$ *(antisymmetry)*
3. $i \sqsubseteq j$ *and* $j \sqsubseteq k$ *implies that* $i \sqsubseteq k$ *(transitivity)*

We say that i and j are comparable if either $i \sqsubseteq j$ or $j \sqsubseteq i$ which is denoted by $i \sqsubseteq\!\sqsupseteq j$. The symbols $\sqsubset, \sqsupseteq, \sqsupset, \not\sqsubseteq, \not\sqsupseteq, \not\sqsubseteq\!\not\sqsupseteq$ are defined as expected. We assume that $\mathcal{S}$ is finite and has size $|\mathcal{S}| = n$. When the underlying set is clear, we abuse notation and conflate $\mathcal{P}$ with $\mathcal{S}$, writing $i \in \mathcal{P}$ and $\mathcal{T} \subseteq \mathcal{P}$ to denote membership in or subset relation to the set underlying $\mathcal{P}$ with inheritance of the order relation $\sqsubseteq$. We also assume that $\mathcal{P}$ is connected: for every $i, j \in \mathcal{P}$ there exists a finite sequence k_α such that $j \sqsubseteq\!\sqsupseteq k_1 \sqsubseteq\!\sqsupseteq \ldots \sqsubseteq\!\sqsupseteq k_l \sqsubseteq\!\sqsupseteq i$.

For every partial order, we can assign a linear order consistent with the partial order. Such an ordering is called a linear extension.

Definition 47. *An* ordering σ *of the poset* $\mathcal{P} = (\mathcal{S}, \sqsubseteq)$ *is a bijection* $\sigma : \mathcal{S} \to \{1, \ldots, n\}$. *We call* σ *a* linear extension *of* $\mathcal{P}$ *if* $j \sqsubseteq i \implies \sigma(j) \leq \sigma(i)$.

Throughout this section, we assume that a fixed linear extension has been chosen for every poset.

Example 17. *Posets are frequently drawn as Hasse diagrams where a directed edge from* j *to* i *is drawn if* $j \sqsubset i$ *and there does not exist* k *such that* $j \sqsubset k \sqsubset i$. *The* $\sqsubseteq$ *relationships are the directed transitive closure of this graph. In Figure 8.6a we draw the Hasse diagram of a poset with* 5 *nodes* $\{1, 2, 3, 4, 5\}$ *satisfying the relationships:*

- $1 \sqsubseteq 3 \sqsubseteq 4$
- $1 \sqsubseteq 3 \sqsubseteq 5$
- $1 \not\sqsubseteq\not\sqsupseteq 2$
- $2 \sqsubseteq 3 \sqsubseteq 4$
- $2 \sqsubseteq 3 \sqsubseteq 5$
- $4 \not\sqsubseteq\not\sqsupseteq 5$

We now introduce several subsets associated with a poset.

Definition 48.

- ***Maximal (resp. minimal) node:*** *i is maximal (resp. minimal) if there does not exist $j \in \mathcal{P}$ such that $i \sqsubset j$ ($j \sqsubset i$)*
- ***Interval:*** $[j, i] \coloneqq \{k \in \mathcal{S} \mid j \sqsubseteq k \sqsubseteq i\}$
- ***Downstream (resp. upstream) set:*** $\downarrow i \coloneqq \{k \in \mathcal{S} \mid i \sqsubseteq k\}$, *($\uparrow i \coloneqq \{k \in \mathcal{S} \mid k \sqsubseteq i\}$)*
- ***Set of nodes with common descendants:*** $\wr i \wr \coloneqq \{k \in \mathcal{P} \mid \downarrow i \cap \downarrow k \neq \emptyset\}$

Given a poset $\mathcal{P}$, its intervals inherit a natural ordering by set inclusion. The resulting poset is known as the poset of intervals $\mathrm{Int}(\mathcal{P})$.

Definition 49. *Given a poset* $\mathcal{P} = (\mathcal{S}, \sqsubseteq)$, *consider the set of all intervals* $\mathcal{T} = \{[j, i] \mid i, j \in \mathcal{S},\ [j, i] \neq \emptyset\}$. *We order* $\mathcal{T}$ *by* $\preceq$ *with* $[j, i] \preceq [l, k]$ *if* $[j, i] \subseteq [l, k]$ *as a set. The tuple* $\mathrm{Int}(\mathcal{P}) = (\mathcal{T}, \preceq)$ *is the* poset of intervals *of* $\mathcal{P}$.

Since $\mathcal{P}$ is finite, $\mathrm{Int}(\mathcal{P})$ is also finite; the total number of elements in $\mathrm{Int}(\mathcal{P})$ is the number of comparable pairs in $\mathcal{P}$ and so $|\mathrm{Int}(\mathcal{P})| \leq \binom{n+1}{2}$. We denote $|\mathrm{Int}(\mathcal{P})| = N$.

An example of an ordered relationship in $\mathrm{Int}(\mathcal{P})$ for the poset $\mathcal{P}$ in Figure 8.6a is given by

$$\{3, 4\} \subseteq \{2, 3, 4\} \implies [3, 4] \preceq [2, 4].$$

However, since $5 \notin \{2, 3, 4\}$, we have $[3, 5] \not\preceq\not\succeq [2, 4]$. The Hasse diagram for $\mathrm{Int}(\mathcal{P})$ is shown in Figure 8.6c.

Finally, we recall the standard notion of the incidence algebra of a poset $\mathcal{P}$.

Definition 50. *Let $\mathcal{P}$ be a poset and $\mathbb{Q}$ a ring. The set of all functions $f : \mathcal{P} \times \mathcal{P} \to \mathbb{Q}$ such that $f(i,j) = 0$ if $j \not\sqsubseteq i$ is called the* incidence algebra of $\mathcal{P}$ over $\mathbb{Q}$ *and is denoted by* $\mathrm{Inc}(\mathcal{P})$.

Since $\mathcal{P}$ is finite, we can arrange the values of f into a lower triangular $\mathbb{Q}$-valued matrix with the rows and columns indexed by $\mathcal{P}$ in the fixed linear extension. This lower triangular matrix has a $\mathbb{Q}$-sparsity pattern determined by the order relation of the poset. A generic example of an element of $\mathrm{Inc}(\mathcal{P})$ for the poset in Figure 8.6a is shown in Figure 8.6b. Addition, scalar multiplication, and multiplication of $f, g \in \mathrm{Inc}(\mathcal{P})$ are defined to be compatible with the usual matrix operations and the operations of the ring. Under these definitions, $\mathrm{Inc}(\mathcal{P})$ forms an associative algebra [306].

Definition 51. *We say a matrix $Y \in \mathrm{Inc}(\mathcal{P})$ if $y_{ij} = 0$ when $j \not\sqsubseteq i$. We say a matrix Y is in* $\mathrm{Inc}(\mathcal{P})$ blockwise *if there is a partitioning of Y into n^2 blocks such that $Y_{ij} = 0$ if $j \not\sqsubseteq i$, or equivalently if $[j,i] \notin \mathrm{Int}(\mathcal{P})$.*

Remark 12. *If $Y \in \mathrm{Inc}(\mathcal{P})$ blockwise, there is a natural pairing between the block Y_{ij} and the interval $[j,i]$. We choose the lower-triangular convention to remain consistent with the poset-causal controller literature [302].*

Finally, we define a special class of posets known as multitrees. We first define a diamond.

Definition 52 (Diamond in a Poset)**.** *Let $\mathcal{P}$ be a poset. Nodes $i,j,k,l \in \mathcal{P}$ form a diamond if $i \sqsubseteq j \sqsubseteq l$ and $i \sqsubseteq k \sqsubseteq l$ but $j \not\sqsubseteq\not\sqsupseteq k$.*

Examples of posets containing diamonds are given in Figures 8.7a, 8.7b and 8.7c.

A multitree is simply a diamond-free poset.

Definition 53 (Multitree [307, Definition 1])**.** *The poset $\mathcal{P}$ is a* multitree *if it contains no diamonds. Equivalently, $\mathcal{P}$ is a multitree if every interval is totally ordered.*

This class of posets was studied in [307], where the optimal poset-causal controllers were shown to enjoy attractive recursive properties.

8.2.1.2 Linear Systems and Graphs

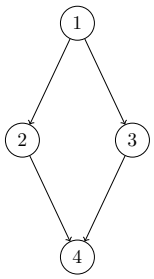
Direct methods for solving linear systems do so via elimination, sequentially solving for one variable in terms of all the others to generate a sequence of smaller and smaller systems. In addition to shrinking the size of the linear operator, elimination also typically densifies the linear operator. This is known as fill-in. A classic result in sparse linear algebra is that this fill-in can be predicted combinatorially, by inspecting only the location of the non-zero entries and not their values.

In this section, we review these classic notions as we will need to use them later.

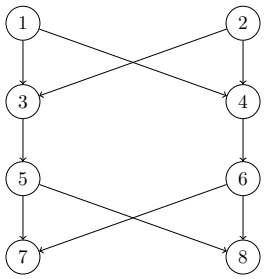
Definition 54. *Let $\mathcal{L} : \mathbb{R}^{\sum_{i=1}^{M} m_i} \to \mathbb{R}^{\sum_{i=1}^{M} m'_i}$ be a block-structured linear operator. Consider the linear system $\mathcal{L}(y) = b$, or equivalently the equations*

$$\mathcal{L}_i(y) = \sum_{j=1}^{M} \mathcal{L}_{ij}(y_j) = b_i, \tag{8.40}$$

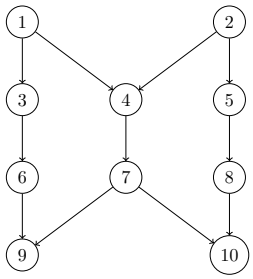
where y_j are block variables of size m_j. The system is symmetric *if the matrix representing $\mathcal{L}$ is symmetric.*



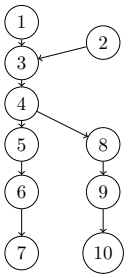
(a) A basic diamond.



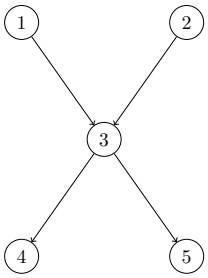
(b) $\{1, 3, 4, 5, 6, 7\}$ forms one dimaond and $\{2, 3, 4, 5, 6, 8\}$ forms another.



(c) Both $\{1, 3, 4, 6, 7, 9\}$ and $\{2, 4, 5, 7, 8, 10\}$ form diamonds.



(d) A multitree with 10 nodes



(e) A basic multitree

Figure 8.7: The top row shows several examples of posets containing diamonds, in particular the canonical diamond. The bottom row shows multitree posets

We now make precise the notion of elimination used throughout this section. It is the usual Schur complement written at the level of block equations.

Definition 55. *Let $\mathcal{S} \subseteq \{1, \ldots, M\}$ and consider the partitioned linear system*

$$\begin{aligned} \mathcal{L}_{\mathcal{S},\mathcal{S}}(y_{\mathcal{S}}) + \mathcal{L}_{\mathcal{S},\mathcal{S}^c}(y_{\mathcal{S}^c}) &= b_{\mathcal{S}}, \\ \mathcal{L}_{\mathcal{S}^c,\mathcal{S}}(y_{\mathcal{S}}) + \mathcal{L}_{\mathcal{S}^c,\mathcal{S}^c}(y_{\mathcal{S}^c}) &= b_{\mathcal{S}^c}. \end{aligned}$$

After eliminating the variables $y_{\mathcal{S}^c}$, the resulting eliminated linear system $\mathcal{L}^{\mathcal{S}}(y_{\mathcal{S}}) = b^{\mathcal{S}}$ *is*

$$\left(\mathcal{L}_{\mathcal{S},\mathcal{S}} - \mathcal{L}_{\mathcal{S},\mathcal{S}^c}\mathcal{L}^{-1}_{\mathcal{S}^c,\mathcal{S}^c}\mathcal{L}_{\mathcal{S}^c,\mathcal{S}}\right)(y_{\mathcal{S}}) = b_{\mathcal{S}} - \mathcal{L}_{\mathcal{S},\mathcal{S}^c}\mathcal{L}^{-1}_{\mathcal{S}^c,\mathcal{S}^c}b_{\mathcal{S}^c}. \tag{8.41}$$

Remark 13. *When solving a linear system via elimination, elements of the eliminated set are typically removed one at a time. While the order of elimination changes the number of operations required, the final eliminated system is unique. That is, if $\mathcal{S} = \mathcal{U} \sqcup \mathcal{R} \sqcup \mathcal{W}$ is the disjoint union of three sets, then*

$$\begin{aligned} (\mathcal{L}^{\mathcal{U}\sqcup\mathcal{R}})^{\mathcal{U}}(y_{\mathcal{U}}) &= \left(b^{\mathcal{U}\sqcup\mathcal{R}}\right)^{\mathcal{U}}, \\ (\mathcal{L}^{\mathcal{U}\sqcup\mathcal{W}})^{\mathcal{U}}(y_{\mathcal{U}}) &= \left(b^{\mathcal{U}\sqcup\mathcal{W}}\right)^{\mathcal{U}}, \\ \mathcal{L}^{\mathcal{U}}(y_{\mathcal{U}}) &= b^{\mathcal{U}} \end{aligned}$$

are all the same equations [308]. Moreover, if $\mathcal{L}$ is symmetric, then $\mathcal{L}^{\mathcal{U}}$ is also symmetric.

The complexity of solving a linear system $\mathcal{L}(y) = b$ via elimination is closely related to the interdependence between its variables. This interdependence can be described in terms of an undirected graph.

Definition 56. *Consider a block symmetric linear system of M equations in M block variables $\mathcal{L}(y) = b$. The* graph of the linear system $\mathcal{G}(\mathcal{L}) = (\mathcal{V}(\mathcal{L}), \mathcal{E}(\mathcal{L}))$ *is the undirected graph with vertex set $\mathcal{V}(\mathcal{L}) = \{1, \dots, M\}$. The pair (i, j) forms an edge in $\mathcal{G}(\mathcal{L})$ if and only if $\mathcal{L}_{ij}(\cdot) \neq 0$.*

Classical work in sparse linear algebra characterizes the complexity of elimination by studying the edge set of the sequence of graphs $\mathcal{G}(\mathcal{L}^{\mathcal{S}})$ as $\mathcal{S}$ shrinks. This is because the absence of an edge in $\mathcal{G}(\mathcal{L}^{\mathcal{S}})$ corresponds to a floating point operation which can be skipped due to the structural zero. The evolution of $\mathcal{G}(\mathcal{L}^{\mathcal{S}})$ can be described completely combinatorially and is related to cliques in a graph (see Definition 37).

The basic combinatorial fact is that, absent coefficient cancellations, eliminating a variable makes its current neighbors pairwise adjacent.

Proposition 6. *Let $\mathcal{S} \subseteq \{1, \dots, M\}$ and let $i \in \mathcal{S}$. Define the neighborhood of i in $\mathcal{G}(\mathcal{L}^{\mathcal{S}})$ by*

$$\mathcal{N}_i \coloneqq \{j \in \mathcal{S} \setminus \{i\} \mid (i, j) \in \mathcal{E}(\mathcal{L}^{\mathcal{S}})\}.$$

Then

$$\mathcal{V}(\mathcal{L}^{\mathcal{S}\setminus\{i\}}) = \mathcal{S} \setminus \{i\}.$$

Generically, the edge set of $\mathcal{G}(\mathcal{L}^{\mathcal{S}\setminus\{i\}})$ is obtained from $\mathcal{G}(\mathcal{L}^{\mathcal{S}})$ by deleting the vertex i and all edges incident to i, and then adding the edges

$$(j, k) \quad \text{for all distinct } j, k \in \mathcal{N}_i.$$

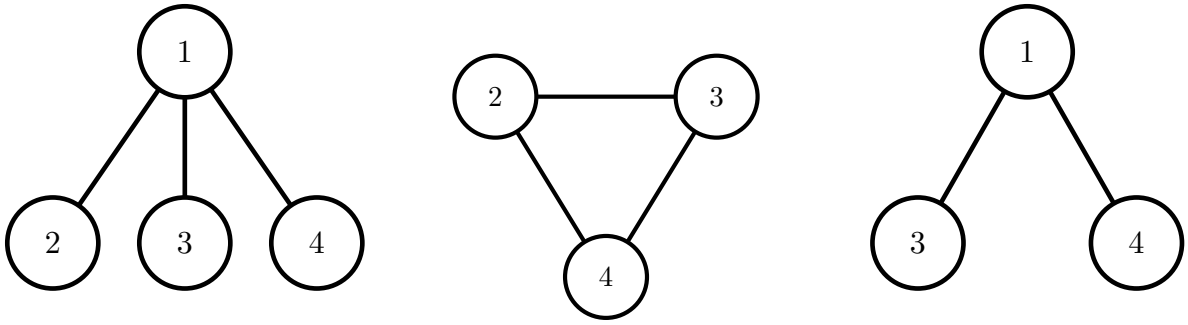
Equivalently, eliminating i turns its neighborhood into a clique.

By carefully eliminating vertices in an order that avoids creating large cliques, sparse linear algebra can save dramatically on computation by reducing the overall number of arithmetic operations [309–311]. The following example illustrates why the elimination order matters.

Example 18. *Consider the symmetric linear system*

$$Ax = b, \quad A = \begin{bmatrix} 6 & -2 & -2 & -2 \\ -2 & 3 & 0 & 0 \\ -2 & 0 & 3 & 0 \\ -2 & 0 & 0 & 3 \end{bmatrix},$$

where $x = (x_1, x_2, x_3, x_4) \in \mathbb{R}^4$ and $b \in \mathbb{R}^4$ is arbitrary. The graph of this linear system is the claw graph shown in Figure 8.8a, with all nodes adjacent to 1 and no other edges.



(a) Initial claw graph $\mathcal{G}(A)$. (b) After eliminating x_1. (c) After eliminating x_2.

Figure 8.8: A sparse linear system whose initial graph is a claw graph and whose elimination graph depends on the elimination order. Eliminating the center node 1 creates the clique K_3 on the remaining vertices, whereas eliminating the leaf node 2 creates no new edge.

If we eliminate x_1 first, then Definition 55 gives the Schur complement

$$A^{\{2,3,4\}} = A_{\{2,3,4\},\{2,3,4\}} - A_{\{2,3,4\},\{1\}} A^{-1}_{\{1\},\{1\}} A_{\{1\},\{2,3,4\}}$$
$$= \begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 3 \end{bmatrix} - \frac{1}{6} \begin{bmatrix} 4 & 4 & 4 \\ 4 & 4 & 4 \\ 4 & 4 & 4 \end{bmatrix} = \begin{bmatrix} 7/3 & -2/3 & -2/3 \\ -2/3 & 7/3 & -2/3 \\ -2/3 & -2/3 & 7/3 \end{bmatrix}.$$

This graph is fully dense. This is predicted by Proposition 6 as the neighbors of 1 are 2, 3, and 4 and so the resulting graph should introduce the clique $(2,3)$, $(2,4)$, and $(3,4)$. This graph is shown in Figure 8.8b.

If instead we eliminate x_2 first, then the remaining system on (x_1, x_3, x_4) is

$$A^{\{1,3,4\}} = A_{\{1,3,4\},\{1,3,4\}} - A_{\{1,3,4\},\{2\}} A^{-1}_{\{2\},\{2\}} A_{\{2\},\{1,3,4\}}$$
$$= \begin{bmatrix} 6 & -2 & -2 \\ -2 & 3 & 0 \\ -2 & 0 & 3 \end{bmatrix} - \frac{1}{3} \begin{bmatrix} 4 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 14/3 & -2 & -2 \\ -2 & 3 & 0 \\ -2 & 0 & 3 \end{bmatrix}.$$

Since the neighborhood of 2 is just 1, no new edges appear in the graph $\mathcal{G}(A^{\{1,3,4\}})$. This graph is visualized in Figure 8.8c.

The family of chordal graphs is exactly those where no fill-in occurs and for these graphs the optimal orders are known as perfect-elimination orders [299].

Definition 57 (Chordal Graphs)**.** *A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is* chordal *if for every cycle of length $K \geq 4$ vertices $\{i_1, \ldots, i_K\}$, then there exists a chord i.e. indices $j, k \in [K]$ such that $(i_j, i_k) \in \mathcal{E}$.*

These good orderings are completely characterized by $\mathcal{G}(\mathcal{L})$, which makes it possible to abstract away the underlying linear operators.

8.2.2 Problem Formulation

Equipped with our formal mathematical preliminaries, we are ready to formally introduce our problem. Let $\mathcal{P}$ be a poset and let $A, B, C \in \mathrm{Inc}(\mathcal{P})$ blockwise. We assume that B has full row rank and C has full column rank and consider the problem

$$\begin{aligned} &\min \ \|A - CXB\|_F^2 \\ &\text{subject to } X \in \mathrm{Inc}(\mathcal{P}). \end{aligned}$$

The equivalent vectorized problem is

$$\min_x \left\| \mathbf{vec}(A) - (B^T \otimes C)Ex \right\|_2^2, \tag{8.42}$$

where x stacks the non-zero entries of the nonzero blocks of X and E is the associated embedding matrix satisfying $Ex = \mathbf{vec}(X)$. We use this representation to derive the optimality conditions; the remainder of the section keeps the block variables explicit.

In this section, the norm of matrices will always be the Frobenius norm and the norm of vectors will always be the ℓ^2 norm and so we will drop the subscripts moving forward.

8.2.3 Main Results

We now derive the block linear system associated with (8.36) and study how its sparsity changes under elimination.

8.2.3.1 Optimality Conditions

Our first theorem characterizes the optimality conditions of (8.36) as a sparse matrix linear system indexed by the intervals of $\mathcal{P}$.

Theorem 24. *Let $\mathcal{P}$ be a poset of size n and let $\mathrm{Int}(\mathcal{P})$ be its associated poset of intervals of size N. The solution to (8.36) is given by the symmetric matrix linear system*

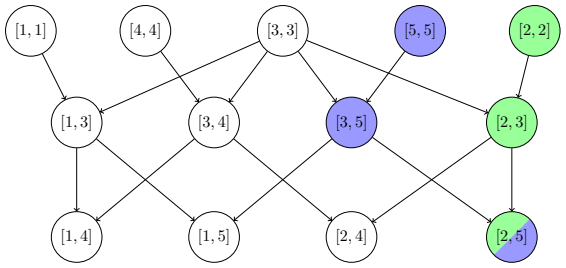
$$\mathcal{L}(X) = b, \tag{8.43}$$

whose block equation associated with the interval $[j, i]$ is

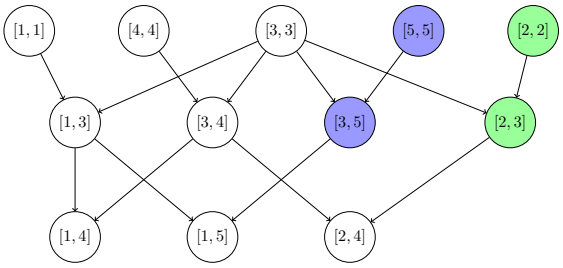
$$\left(\sum_{s \sqsupseteq i} C_{si}^T C_{si}\right) X_{ij} \left(\sum_{t \sqsubseteq j} B_{jt} B_{jt}^T\right) + \sum_{\substack{[l,k] \in \supsetneq [j,i] \subsetneq \\ [l,k] \neq [j,i]}} \left(\sum_{\substack{s \sqsupseteq i \\ s \sqsupseteq k}} C_{si}^T C_{sk}\right) X_{kl} \left(\sum_{\substack{t \sqsubseteq j \\ t \sqsubseteq l}} B_{lt} B_{jt}^T\right) = \sum_{[t,s]:\ t \sqsubseteq j,\ i \sqsubseteq s} C_{si}^T A_{st} B_{jt}^T. \tag{8.44}$$

This system has a unique solution since B and C have full row and column rank by assumption.

Immediate from (8.44) is the following corollary.



(a) The node $[2,5]$ is a common descendant of the nodes $\{[2,2],[2,3]\}$ in green and $\{[5,5],[3,5]\}$ in blue, so these nodes share edges in $\mathcal{G}(\mathcal{L})$.



(b) After elimination of $[2,5]$, Theorem 25 shows that the blue and green groups no longer share an edge in the eliminated system.

Figure 8.9: A common-descendant relation in $\mathrm{Int}(\mathcal{P})$ and the corresponding sparsification after one elimination step.

Corollary 2. *Two nodes form an edge in $\mathcal{G}(\mathcal{L})$ only if they have a common descendant in* $\mathrm{Int}(\mathcal{P})$.

Using Corollary 2, we can read the adjacency matrix of $\mathcal{G}(\mathcal{L})$ directly from the Hasse diagram of $\mathrm{Int}(\mathcal{P})$. For example, using the poset from Figure 8.6a, the node $[2,5]$ is a common descendant of the groups of nodes $\{[2,2],[2,3]\}$ and $\{[5,5],[3,5]\}$, and therefore these nodes all share edges in $\mathcal{G}(\mathcal{L})$. This relationship is illustrated on the Hasse diagram in Figure 8.9a and the adjacency matrix of $\mathcal{G}(\mathcal{L})$ is shown in Figure 8.10 with the induced $[2,5]$ relationship highlighted in blue and green.

Proof 8.6. Since (8.36) is equivalent to (8.42), the first-order optimality conditions can be written in two equivalent ways:

$$E^T(BB^T \otimes C^TC)Ex = E^T \mathbf{vec}(C^TAB^T) \tag{8.45}$$

$$C^TAB^T - C^TCXBB^T \in \mathrm{Inc}(\mathcal{P})^c \text{ blockwise.} \tag{8.46}$$

Here $Y \in \mathrm{Inc}(\mathcal{P})^c$ blockwise means that $Y_{ij} = 0$ whenever $j \sqsubseteq i$.

Symmetry of the optimality conditions is immediate from the symmetry of $E^T(BB^T \otimes C^TC)E$. The expression (8.44) is simply the (i,j) block of (8.46) for $j \sqsubseteq i$. To expand that expression, note that since $B, C \in \mathrm{Inc}(\mathcal{P})$ we have

$$(C^TC)_{ik} = \sum_{\substack{s \sqsupseteq i \\ s \sqsupseteq k}} C^T_{si} C_{sk},$$

$$(BB^T)_{lj} = \sum_{\substack{t \sqsubseteq j \\ t \sqsubseteq l}} B_{lt} B^T_{jt}.$$

Expanding the second term of (8.46) yields

$$(C^TCXBB^T)_{ij} = \sum_k \sum_l (C^TC)_{ik} X_{kl} (BB^T)_{lj}.$$

A term in this sum is zero unless the following hold

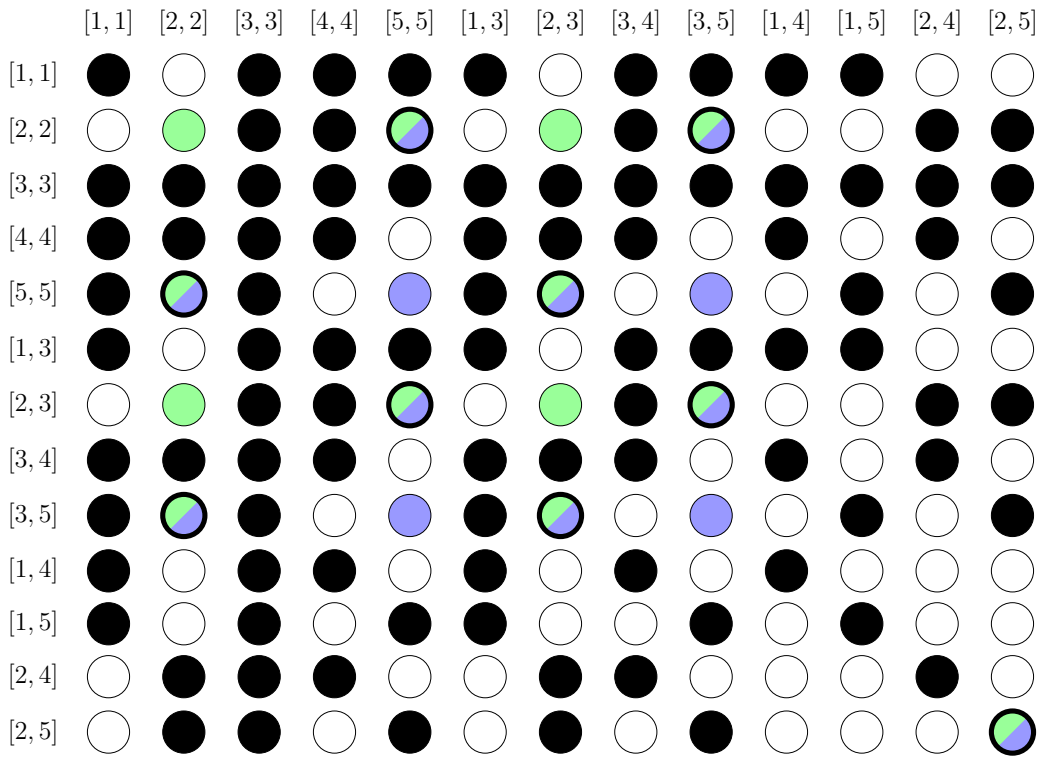


Figure 8.10: The adjacency matrix of $\mathcal{G}(\mathcal{L})$ for the poset from Example 17. A filled circle represents an edge in $\mathcal{G}(\mathcal{L})$ and an empty circle represents the lack of an edge. A solid green or blue circle represents an edge inherited from $\mathrm{Int}(\mathcal{P})$. A half blue and green circle represents an edge induced by the common descendant relationship to $[2, 5]$ that is not present in $\mathrm{Int}(\mathcal{P})$. The blue-green couplings are those induced by the common descendant $[2, 5]$.

1. $[l, k]$ is a nonempty interval
2. There exists s such that $s \sqsupseteq i$ and $s \sqsupseteq k$
3. There exists t such that $t \sqsubseteq l$ and $t \sqsubseteq j$.

This is exactly the common-descendant condition in $\mathrm{Int}(\mathcal{P})$. The right-hand side of (8.44) is obtained similarly by keeping only the terms allowed by the incidence-algebra sparsity.

Uniqueness of the solution to (8.43) follows from uniqueness of the minimizer of (8.36), which is guaranteed by the rank assumptions on B and C. □

Remark 14. *We note that the sparsity pattern predicted by Theorem 24 is at least as dense as* $\mathrm{Int}(\mathcal{P})$*. Moreover, this sparsity pattern is not chordal. For example, the common descendant relationships of* $\mathrm{Int}(\mathcal{P})$ *for the posets shown in Figures 8.7c and 8.7d are not chordal. For* $\mathrm{Int}(\mathcal{P})$ *of Figure 8.7d* $[1, 1] - [5, 5] - [2, 2] - [8, 8]$ *is a four cycle with no chord and for* $\mathrm{Int}(\mathcal{P})$ *of Figure 8.7c* $[1, 1] - [9, 9] - [2, 2] - [10, 10] - [1, 1]$ *is a four cycle with no chord. Therefore, we should expect an elimination for* (8.45) *to exhibit significant fill-in despite already being relatively dense.*

In the next two subsections, we study how (8.43) changes under elimination. The key result is that the particular structure of (8.46) allows us to generate substantially less fill-in than is predicted by the general theory of Proposition 6.

8.2.3.2 Separation in the Optimality Conditions

Although (8.43) is highly structured, the graph $\mathcal{G}(\mathcal{L})$ can still be comparatively dense. For example, if $\mathcal{P}$ is a chain, then $\mathcal{G}(\mathcal{L})$ is complete. The next theorem shows that elimination nevertheless creates cancellations that are not predicted by generic sparse-system theory.

Theorem 25. *Consider the interval $[j,i]$ and two upstream intervals $[j,q] \prec [j,i]$ and $[p,i] \prec [j,i]$. After eliminating X_{ij} from (8.43), the variable X_{qj} vanishes from equation $[p,i]$ and, symmetrically, X_{ip} vanishes from equation $[j,q]$. Equivalently,*

$$\mathcal{L}_{qj}^{\{[j,i]\}^c}(\cdot) = \mathcal{L}_{ip}^{\{[j,i]\}^c}(\cdot) = 0,$$

so $[p,i]$ and $[j,q]$ are not adjacent in $\mathcal{G}(\mathcal{L}^{\{[j,i]\}^c})$.

Theorem 25 demonstrates a separation property specific to the block structure of (8.43): eliminating a downstream block can reduce the interdependence of upstream blocks. For instance, in Figure 8.9a, the green nodes $\{[2,2],[2,3]\}$ and the blue nodes $\{[5,5],[3,5]\}$ share an edge in $\mathcal{G}(\mathcal{L})$ because both sets have the common descendant $[2,5]$. Eliminating X_{52} removes the node $[2,5]$ itself and also removes the blue-green couplings.

Proof 8.7. Using (8.44), we can solve for X_{ij} as a function of the remaining variables. From that expression, the only term that depends on X_{ip} is

$$-\left(\sum_{s\sqsupseteq i} C_{si}^T C_{si}\right)^{-1}\left(\sum_{s\sqsupseteq i} C_{si}^T C_{si}\right) X_{ip} \left(\sum_{\substack{t\sqsubseteq j\\ t\sqsubseteq p}} B_{pt}B_{jt}^T\right)\left(\sum_{t\sqsubseteq j} B_{jt}B_{jt}^T\right)^{-1}. \tag{8.47}$$

If we now inspect the dependence of the equation $\mathcal{L}_{qj}(X) = b_{qj}$ on X_{ij} and X_{ip}, we obtain

$$\left(\sum_{s\sqsupseteq i} C_{sq}^T C_{si}\right) X_{ij} \left(\sum_{t\sqsubseteq j} B_{jt}B_{jt}^T\right) + \left(\sum_{s\sqsupseteq i} C_{sq}^T C_{si}\right) X_{ip} \left(\sum_{\substack{t\sqsubseteq j\\ t\sqsubseteq p}} B_{pt}B_{jt}^T\right). \tag{8.48}$$

Substituting (8.47) into (8.48) gives

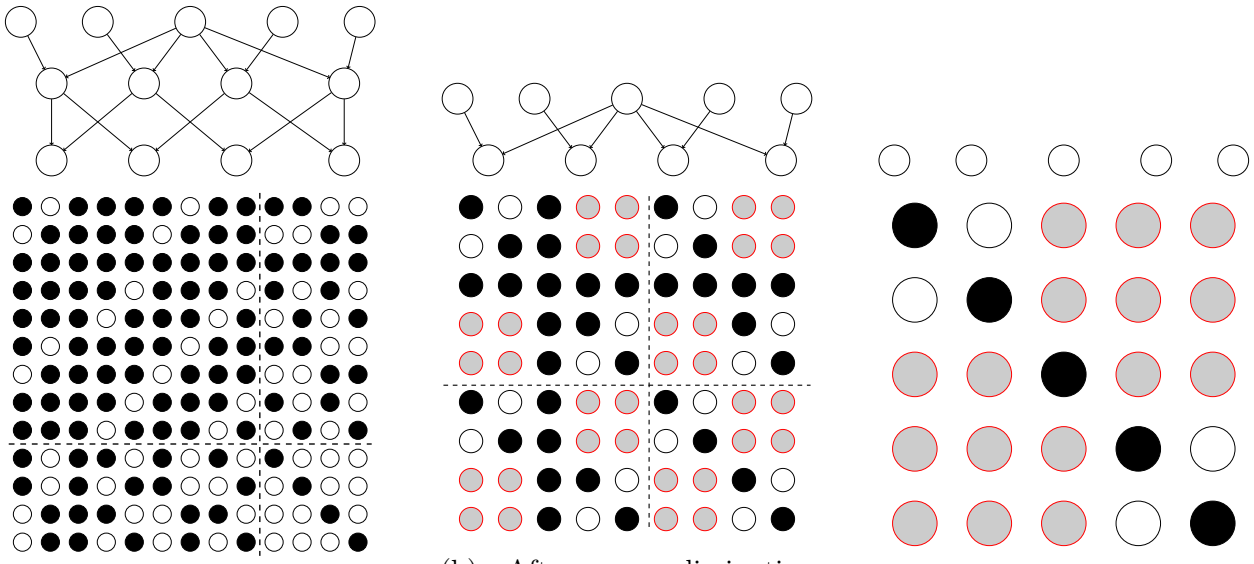
$$-\left(\sum_{s\sqsupseteq i} C_{sq}^T C_{si}\right) X_{ip} \left(\sum_{\substack{t\sqsubseteq j\\ t\sqsubseteq p}} B_{pt}B_{jt}^T\right) + \left(\sum_{s\sqsupseteq i} C_{sq}^T C_{si}\right) X_{ip} \left(\sum_{\substack{t\sqsubseteq j\\ t\sqsubseteq p}} B_{pt}B_{jt}^T\right) = 0. \tag{8.49}$$

Therefore $\mathcal{L}_{qj}^{\{[j,i]\}^c}(\cdot) = 0$. The proof that $\mathcal{L}_{ip}^{\{[j,i]\}^c}(\cdot) = 0$ is symmetric. □

We emphasize that Theorem 25 is true because of the interplay between the poset structure and the tensor product structure of the optimality equations. Both are important.

While Theorem 25 characterizes the disappearance of some edges after a single elimination step, it does not by itself describe the full graph $\mathcal{G}(\mathcal{L}^{\mathcal{S}})$ after many steps. The next subsection shows that, for multitree posets, the same sparsification continues throughout elimination.

8.2.3.3 Sparsification During Elimination in Multitree Posets



(a) Initial system.

(b) After one elimination step.

(c) After two elimination steps.

Figure 8.11: Evolution of the graph $\mathcal{G}(\mathcal{L}^{(r)})$. Black nodes represent nonzero entries, white nodes are structural zeros inherited from $\mathrm{Int}(\mathcal{P})$, and gray nodes vanish because of the additional cancellations captured by Theorem 25. At every, the variables associated to the bottom right block after the hashed line are eliminated.

For multitree posets, the cancellations from Theorem 25 persist throughout elimination and can be tracked combinatorially. In short, if we eliminate variables from largest to smallest in $\mathrm{Int}(\mathcal{P})$, the cancellations from Theorem 25 persist through the whole elimination process.

We begin by characterizing when two intervals can have a unique common descendant: after ordering them so that $t \not\sqsubseteq l$, one must share the descendant's lower endpoint and the other its upper endpoint.

Proposition 7. *Let $\mathcal{P}$ be a multitree and let $[l,k]$ and $[t,s]$ be two nodes in $\mathrm{Int}(\mathcal{P})$ with $t \not\sqsubseteq l$ whose only strict common descendant is $[j,i]$. Then $l=j$ and $s=i$.*

Proof 8.8. Since $\mathcal{P}$ is a multitree, the interval $[j,i]$ is a chain, so l,k,s,t are mutually comparable. Without loss of generality, assume $j \sqsubseteq l \sqsubseteq s \sqsubseteq i$. If $s \sqsubset i$, then $[l,i]$ contains both intervals. If $l \sqsupset j$, then $[j,s]$ contains both intervals. In either case, the assumption that $[l,k]$ and $[t,s]$ have only one common descendant is violated. □

Next, we fix a linear extension σ of $\mathrm{Int}(\mathcal{P})$ and perform elimination in the reverse of this order. After r steps, the remaining variables are associated with the subposet

$$\mathrm{Int}(\mathcal{P})^{(r)} := \{[l,k] \in \mathrm{Int}(\mathcal{P}) \mid \sigma([l,k]) \leq N - r\}, \tag{8.50}$$

and the eliminated system is

$$\mathcal{L}^{\mathrm{Int}(\mathcal{P})^{(r)}}(X_{\mathrm{Int}(\mathcal{P})^{(r)}}) = b^{\mathrm{Int}(\mathcal{P})^{(r)}}. \tag{8.51}$$

We will use the shorthand $\mathcal{L}^{(r)}(X_{(r)}) = b^{(r)} = \mathcal{L}^{\mathrm{Int}(\mathcal{P})^{(r)}}(X_{\mathrm{Int}(\mathcal{P})^{(r)}}) = b^{\mathrm{Int}(\mathcal{P})^{(r)}}$.

We can now state the main elimination theorem.

Theorem 26. *Let $\mathcal{P}$ be a multitree and fix a linear extension σ for $\mathrm{Int}(\mathcal{P})$. Then two nodes form an edge in $\mathcal{G}(\mathcal{L}^{(r)})$ only if they share a common descendant in $\mathrm{Int}(\mathcal{P})^{(r)}$.*

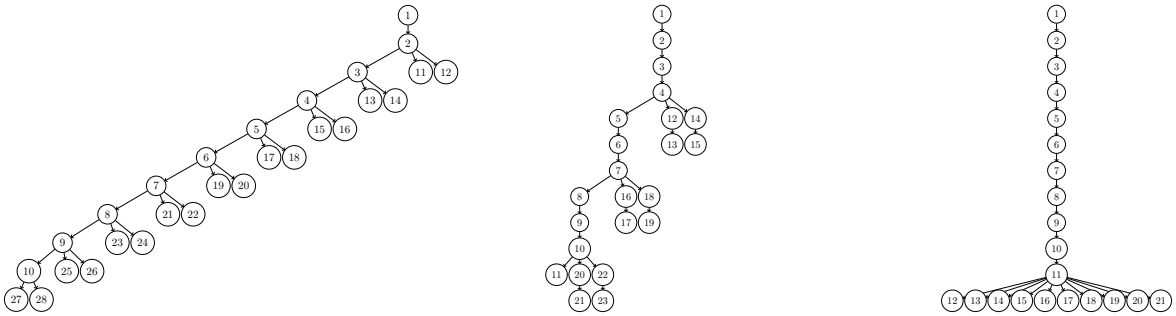
Theorem 26 completely characterizes the edge set of $\mathcal{G}(\mathcal{L}^{(r)})$ throughout elimination according to linear extension orderings of multitrees. It improves on Theorem 25, which only describes a single elimination step for a subset of the edges. It is also stronger than the standard graph-only theory of [309–311], because it predicts that the eliminated systems are sparser than the generic fill pattern would suggest. This progressive sparsification is visualized in Figure 8.11.

Proof 8.9. Let $[j,i] = \sigma^{-1}(N)$. The interval $[j,i]$ is maximal in $\mathrm{Int}(\mathcal{P})$, so $\wr[j,i]\wr = \uparrow [j,i]$. Therefore, the neighbors of $[j,i]$ form a clique in $\mathcal{G}(\mathcal{L})$. If $\mathcal{L}$ were a generic sparse system, [309] would predict that every edge in the induced subgraph on the remaining variables persists in $\mathcal{G}(\mathcal{L}^{(1)})$.

Now consider nodes $[j,p],[q,i] \in \mathcal{V}(\mathcal{L}^{(1)})$ whose only common descendant is $[j,i]$. By Theorem 25, these nodes do not share an edge in $\mathcal{G}(\mathcal{L}^{(1)})$. By Proposition 7, these are the only pairs whose only common descendant is $[j,i]$. Therefore two nodes share an edge in $\mathcal{G}(\mathcal{L}^{(1)})$ only if they share a common descendant in $\mathrm{Int}(\mathcal{P})^{(1)}$.

The same argument continues inductively. Suppose inductively that adjacency in $\mathcal{G}(\mathcal{L}^{(r-1)})$ is given by the common-descendant relation of the poset $\mathrm{Int}(\mathcal{P})^{(r-1)}$. Since $[j,i] = \sigma^{-1}(N - r + 1)$ is maximal in $\mathrm{Int}(\mathcal{P})^{(r-1)}$, its neighbors again form a clique in $\mathcal{G}(\mathcal{L}^{(r-1)})$. Therefore, after eliminating $[j,i]$, no new adjacencies can appear beyond those already present in $\mathcal{G}(\mathcal{L}^{(r-1)})$. Using Theorem 25 together with the order-independence statement in Remark 13, it follows that adjacency in $\mathcal{G}(\mathcal{L}^{(r)})$ is precisely the common-descendant relation of $\mathrm{Int}(\mathcal{P})^{(r)}$. □

The result of Theorem 26 is similar to the results that linear systems with chordal sparsity exhibit no-fill if variables are eliminated according to a perfect elimination ordering [299]. However, we note that our result is distinct. First, the common descendant relationship of $\mathrm{Int}(\mathcal{P})$, which governs the sparsity of the linear system, is not chordal (see Remark 14). Second, our theorem goes beyond zero fill-in. In fact, it states that in multitrees the complexity of elimination is only governed by the edges of $\mathrm{Int}(\mathcal{P})$ rather than the edges of the common descendant relationship.



(a) Caterpillar (b) Palm (c) Broom

Figure 8.12: Representative medium-sized multitrees used to define the three benchmark families. Each family is diamond-free but contains substantial chain overlap, so the benchmark stresses the cancellation mechanism of Theorem 25.

8.2.4 Results

We illustrate the consequence of Theorem 26 on three multitree posets shown in Figure 8.12.

- The caterpillar poset consists of one long backbone chain. Each node on the backbone, except the initial root, sprouts K incomparable leaf nodes attached directly to the backbone. This poset is shown in Figure 8.12a.
- The palm poset consists of one long backbone chain. At every Kth node along the backbone, the poset sprouts M additional branches, and adds a chain of length N to each branch.
- The broom poset consists of one long backbone chain whose terminal node sprouts K incomparable leaves. Equivalently, it has K maximal chains of length N that share their first $N-1$ nodes and branch only at the final step. This poset is shown in Figure 8.12c.

For each of these posets, we construct a poset of three different sizes. For each poset, we construct data matrices B and C in $\mathrm{Inc}(\mathcal{P})$, by sampling the diagonal entries independently and uniformly at random from $[1.1, 1.9]$ and off-diagonal entries uniformly in $[-0.4, 0.4]$. A solution $X_\star \in \mathrm{Inc}(\mathcal{P})$ is drawn with entries drawn i.i.d. from a standard Gaussian. The matrix A is constructed as $A = CX_\star B$. We set the block size to 1×1 for simplicity. Finally, we assemble the normal equations from Theorem 24 and compare three elimination strategies:

1. The *poset-order structural cancellation* path eliminates nodes using the reverse of a linear extension of $\mathrm{Int}(\mathcal{P})$. It keeps track of the additional structural zeros predicted by Theorem 26 to avoid doing additional work.
2. The *poset-order no structural cancellation* path eliminates nodes using the reverse of a linear extension of $\mathrm{Int}(\mathcal{P})$. However, it does not keep track of the additional structural zeros predicted by Theorem 26, forcing it to keep track of the additional edges generically introduced by performing elimination.

3. An *AMD* sparse baseline which ignores the poset altogether and simply applies the approximate minimum degree heuristic to compute a fill-reducing ordering of the linear system. We then solve the linear system according to this order.

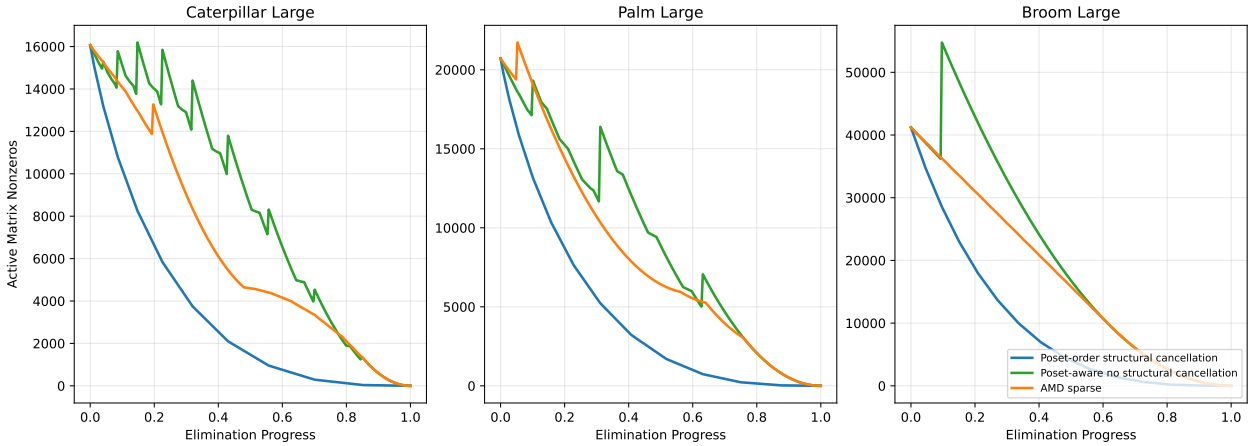


Figure 8.13: Number of nonzeros in the remaining eliminated system compared to the fraction of eliminated variables.

In Figure 8.13, we plot the number of structural non-zeros in the linear operators on the largest instance from each family as we perform elimination. The two poset-order methods use the same reverse interval-poset elimination order, so the gap between them isolates the benefit of structural cancellation rather than the benefit of ordering alone. We see that our reverse interval-poset elimination order, when aware of the structural cancellations, maintains the most structurally sparse matrices at every point in the elimination, beating out the AMD heuristic by a factor of 2.08 to 2.57. By contrast, if the algorithm is not aware of the structural cancellation that occurs because of our special linear system, then the elimination algorithm must continue to keep track of these zero elements and therefore believes that the systems are 2.74 to 4.18 times denser than they are. This results in a system that is consistently structurally denser than the one tracked by the AMD ordering.

Similarly, in Figure 8.14, we consider the number of neighbors that the variable to be eliminated has in the remaining graph. This is the number of off-diagonal non-zero elements in the row selected for elimination. This number is called the front size and quantifies how much arithmetic work it takes to eliminate a given variable. The total arithmetic work at a step is proportional to the square of the front size at that step. Again, we see that keeping track of the structural cancellations explained by Theorem 26 results in substantially less work than the AMD baseline, but failing to keep track of these additional zeros leads to performing substantially more work.

Finally, in Figure 8.15 we illustrate the effect of these structural results on the runtime of solving the linear system.

Overall, we see that using the elimination order together with the additional structural information from Theorem 26 results in a 1.73 to 4.29 times speedup across the range of instances compared to the AMD baseline. By contrast, not leveraging the additional structural zeros results in an algorithm that is 1.15 to 2.32 times slower than the AMD baseline and 3.16 to 8.70 times slower than leveraging the structural zeros.

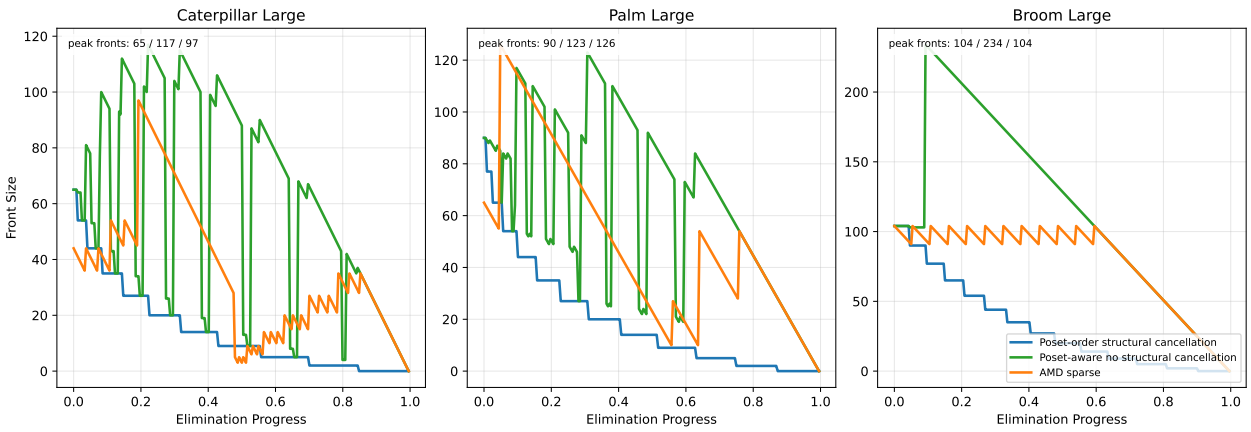


Figure 8.14: Number of neighbors of the eliminated variable compared to the fraction of eliminated variables. The front-size is the number of neighbors and corresponds to the amount of arithmetic work needed to eliminate a given variable.

8.2.5 Conclusion

In this section, we showed that solving (8.36) can be reduced to a sparse linear system naturally indexed by the intervals of the underlying poset. Theorem 24 shows that the sparsity of this linear system is governed by the common descendant relationship of the original poset, which can be quite dense. Fortunately, the tensor and order-theoretic structure of the linear system exhibits a certain separability property: when a unique common descendant of an interval is eliminated, the two upstream intervals become decoupled in the linear system. This is Theorem 25. Finally, we show in Theorem 26 that for multitree posets, this separability property holds inductively; by eliminating the variables from largest to smallest, these cancellations persist throughout the entire elimination process. The result is a solution process that is much more efficient than a general sparse solver baseline.

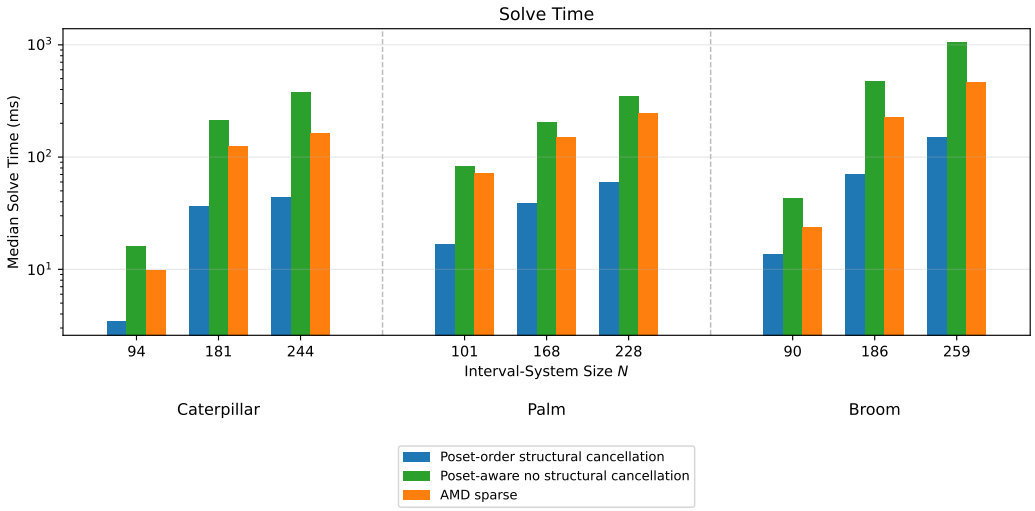


Figure 8.15: Solve times for each of the posets on various sizes. The *poset-aware no structural cancellation* baseline isolates the benefit of the structural cancellation theorem when the elimination order is held fixed.

Chapter 9

Solving Graph of Convex Sets Instances

In Chapter 3, we briefly introduced the GCS framework of [37] and discussed its connection to SOS programming. The GCS relaxation uses the SOS machinery to construct a relaxation of a hard problem. Rather than add the quadratically many variables and constraints of the first-order SOS relaxation, the GCS relaxation parsimoniously adds only a minimal subset of the SOS constraints needed to obtain sufficiently tight relaxations in practice. In particular, GCS relaxation adds only those constraints which do not disrupt the graphical structure of the original GCS instance. Despite these frugal choices, GCS relaxations can still be quite large, even without lifting to higher-order SOS relaxations [94–96].

In practice, the GCS relaxation need only be solved to low-accuracy to discover a relaxed value of the binaries. Once the discrete part of the problem is understood, a much smaller convex program can instead be solved. Additionally, GCS is currently being used in motion planning pipelines at Sony Robotics [312] and Amazon [313] and so fast, repeated solve times are of the utmost importance.

In this chapter, we explore the application of ADMM to the GCS convex relaxation. We design a natural split of the variables that will enable us to partition the problem into its three native components: an optimization program at each vertex, an optimization program at each edge, and an optimization program for only the discrete part of the problem. We provide an implementation we call the `VEGA` (Vertex-Edge-Graph-ADMM) solver. The solver performs best on very large GCS programs, while demonstrating only modest performance on small to moderate sized instances when compared to using generic solvers.

9.1 An ADMM Split for GCS

The GCS MICP is given in (3.17) (after dropping the redundant quadratic equality (3.17g)). The GCS relaxation is the exact same program, with (3.17f) relaxed to $y \in [0, 1]$. This constraint can be appended to the polytope in (3.17e).

In order to design an ADMM splitting scheme, it is helpful to abstract away the particular structure of the linear operators and the cones in the GCS relaxation. Doing so, the

relaxation can be written in the form

$$\min \sum_{v\in\mathcal{V}} f_v(z_v, y_v) + \sum_{e\in\mathcal{E}} f_e(z_{e_u}, z_{e_v}, z_e, y_e) \tag{9.1a}$$

$$\text{subject to } \bar{\mathcal{A}}_v(z_v, y_v, z_{\mathcal{E}_v}, y_{\mathcal{E}_v}) \in \mathcal{K}_v,\ \forall v \in \mathcal{V} \tag{9.1b}$$

$$\bar{\mathcal{A}}_e(z_{e_u}, z_{e_v}, z_e, y_e) \in \mathcal{K}_e,\ \forall e \in \mathcal{E} \tag{9.1c}$$

$$\bar{\mathcal{A}}_{\mathcal{G}}(y) - b_{\mathcal{G}} \in \mathcal{K}_{\mathcal{G}} \tag{9.1d}$$

where $\bar{\mathcal{A}}_v$, $\bar{\mathcal{A}}_e$, and $\bar{\mathcal{A}}_{\mathcal{G}}$ are all linear operators and $\mathcal{K}_v, \mathcal{K}_e, \mathcal{K}_{\mathcal{G}}$ are all products of proper cones and the zero cone. Concretely, (9.1b) corresponds to aggregation of (3.17b) and (3.17c), (9.1c) corresponds to (3.17d), and (9.1d) corresponds to (3.17e). We remind the reader of the notation that $z_{\mathcal{E}_v} = \begin{bmatrix} z_{e1_v} & \dots & z_{e|\mathcal{E}_v|_v} \end{bmatrix}$, i.e. a matrix where every column corresponds to an edge spatial variable incident to the vertex v.

In (9.1) we color variables by type. The spatial vertex variables z_v are colored blue, the edge-spatial variables z_{e_v} and z_e are colored red, and the flow variables y are colored orange.

The coloring of the variables in (9.1) highlights the way that the variables interact. We visualize the overlap in the variables in the program (9.1) for a chain of three vertices u, v, w jointly connected through v in Figure 9.1.

The blue circles contain all the variables paired through a constraint (9.1b). The red circles contain all the variables paired through a constraint (9.1c). The orange variables are coupled globally through (9.1d).

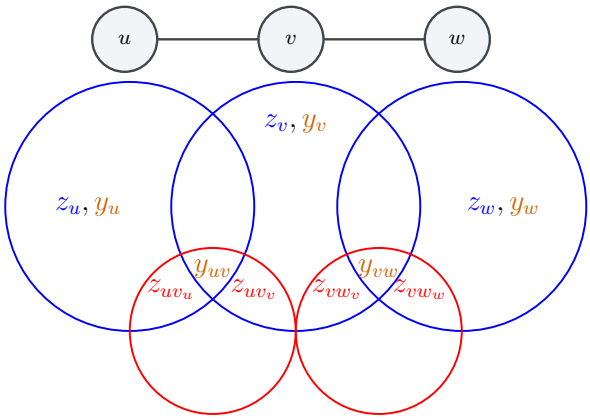
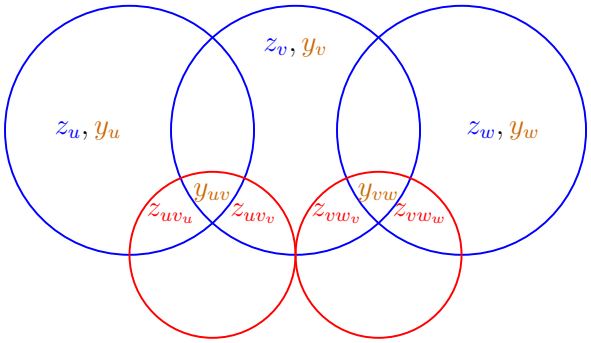
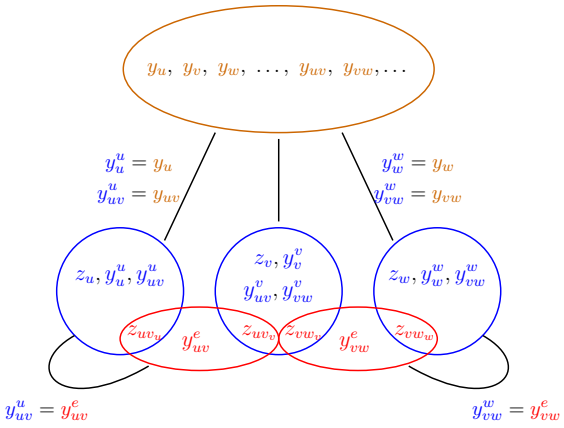


Figure 9.1: We visualize the overlap in the variables in the program (9.1) for a chain of three vertices. The blue circles contain all the variables paired through a constraint (9.1b). The red circles contain all the variables paired through a constraint (9.1c). The orange variables are coupled globally through (9.1d).

In order to disentangle the interdependencies between the constraints in (9.1), we will introduce copies of variables and pair them through equalities. We will do this in stages. First, we will disentangle the flow-variables which are coupled through every constraint. At every vertex v, we will introduce a local copy of the vertex flow y_v^v and a local copy of the edge flow y_e^v for every $e \in \mathcal{E}_v$. Similarly, at every edge we will introduce a local copy of the edge flow y_e^e. We will couple all the copies through the vertex copies. We visualize this process in Figure 9.2b.



(a) Before duplication, the binaries and endpoint variables couple the vertex, edge, and graph blocks.



(b) Duplicating only the binary variables separates the graph problem from the spatial conic programs.

Figure 9.2: We can separate the constraints encoding the subgraph selection from the conic programs by introducing copies of the flow variable.

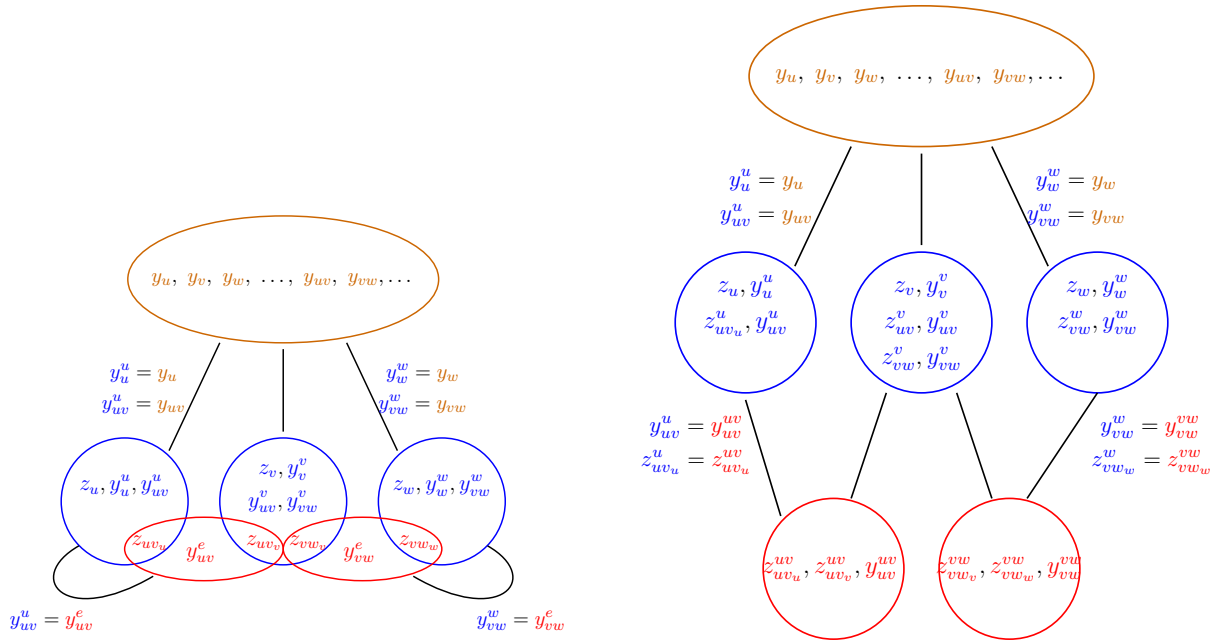
The resulting program is

$$\min \sum_{v\in\mathcal{V}} f_v(z_v, y_v^v) + \sum_{e\in\mathcal{E}} f_e(z^e_{e_u}, z^e_{e_v}, z_e, y^e_e) \tag{9.2a}$$
$$\text{subject to } \bar{\mathcal{A}}_v(z_v, y_v^v, z_{\mathcal{E}_v}, y^v_{\mathcal{E}_v}) \in \mathcal{K}_v,\ \forall v \in \mathcal{V} \tag{9.2b}$$
$$\bar{\mathcal{A}}_e(z_{e_u}, z_{e_v}, z_e, y^e_e) \in \mathcal{K}_e,\ \forall e \in \mathcal{E} \tag{9.2c}$$
$$\bar{\mathcal{A}}_{\mathcal{G}}(y) - b_{\mathcal{G}} \in \mathcal{K}_{\mathcal{G}} \tag{9.2d}$$
$$y_v^v = y_v,\ y_e^v = y_e,\ \forall v \in \mathcal{V},\ \forall e \in \mathcal{E}_v \tag{9.2e}$$
$$y_e^v = y_e^e,\ \forall v \in \mathcal{V},\ \forall e \in \mathcal{E}_v. \tag{9.2f}$$

Notice that (9.2e), (9.2c), and each term of (9.2a) are now fully decoupled in the sense that they only contain variables of one color. The only overlap, besides the equalities (9.2e) and (9.2f), is the shared spatial edge variables z_{e_v} between a vertex v and an incident edge $e \in \mathcal{E}_v$. To finish decoupling (9.2b) from the remaining constraints, we introduce a local copy of the edge spatial variable $z^v_{e_v}$ at every vertex. This procedure is again visualized in Figure 9.3. The final program is written as

$$\min \sum_{v\in\mathcal{V}} f_v(z_v, y_v^v) + \sum_{e\in\mathcal{E}} f_e(z^e_{e_u}, z^e_{e_v}, z_e, y_e) \tag{9.3a}$$
$$\text{subject to } \bar{\mathcal{A}}_v(z_v, y_v^v, z^v_{\mathcal{E}_v}, y^v_{\mathcal{E}_v}) \in \mathcal{K}_v,\ \forall v \in \mathcal{V} \tag{9.3b}$$
$$\bar{\mathcal{A}}_e(z_{e_u}, z_{e_v}, z_e, y^e_e) \in \mathcal{K}_e,\ \forall e \in \mathcal{E} \tag{9.3c}$$
$$\bar{\mathcal{A}}_{\mathcal{G}}(y) - b_{\mathcal{G}} \in \mathcal{K}_{\mathcal{G}} \tag{9.3d}$$
$$y_v^v = y_v,\ y_e^v = y_e,\ \forall v \in \mathcal{V},\ \forall e \in \mathcal{E}_v \tag{9.3e}$$
$$y_e^v = y_e^e,\ z^v_{e_v} = z^e_{e_v},\ \forall v \in \mathcal{V},\ \forall e \in \mathcal{E}_v. \tag{9.3f}$$

The final step to writing (9.3) is to move the constraints (9.3b), (9.3c), and (9.3d) into



(a) Duplicating only the binary variables separates the graph problem from the spatial conic programs.

(b) By duplicating the spatial edge variables, we separate the edge conic programs from the vertex conic programs.

Figure 9.3: We can separate the vertex and edge conic programs by introducing copies of the spatial edge variables. This results in a program partitioned into three blocks.

the cost. We let

$$\begin{aligned} F_v(z_v, y_v^v, z_{\mathcal{E}_v}^v, y_{\mathcal{E}_v}^v) &= f_v(z_v, y_v^v) + \mathbb{1}(\bar{\mathcal{A}}_v(z_v, y_v^v, z_{\mathcal{E}_v}^v, y_{\mathcal{E}_v}^v) \in \mathcal{K}_v) \\ F_e(z_{e_u}^e, z_{e_v}^e, z_e^e, y_e^e) &= f_e(z_{e_u}^e, z_{e_v}^e, z_e, y_e^e) + \mathbb{1}\left(\bar{\mathcal{A}}_e(z_{e_u}^e, z_{e_v}^e, z_e, y_e^e) \in \mathcal{K}_e\right) \\ F_{\mathcal{G}}(y) &= \mathbb{1}(\bar{\mathcal{A}}_{\mathcal{G}}(y) - b_{\mathcal{G}} \in \mathcal{K}_{\mathcal{G}}). \end{aligned}$$

We note that in (9.3), the variable z_v and the variable z_e do not appear in any consensus constraint. Many works have shown that ADMM exhibits superior numerical convergence if an artificial variable $\tilde{\lambda}$ is introduced for every uncoupled variable λ [23,103,112,224]. One reason for this is ensuring that the subproblems in Lines 1.3 and 1.4 have unique solutions, as is done in Chapter 7. It can be shown that if the penalty between λ and $\tilde{\lambda}$ is decoupled from other consensus constraints then no dual variable is needed to drive consensus.

Therefore, we write (9.3) in the ADMM standard form

$$\min \sum_{v\in\mathcal{V}} F_v(z_v, y_v^v, z_{\mathcal{E}_v}^v, y_{\mathcal{E}_v}^v) + \sum_{e\in\mathcal{E}} F_e(z_{e_u}^e, z_{e_v}^e, z_e, y_e^e) + F_{\mathcal{G}}(y) \tag{9.4a}$$

$$\text{subject to } y_v^v = y_v,\ y_e^v = y_e,\ \forall v \in \mathcal{V},\ \forall e \in \mathcal{E}_v \tag{9.4b}$$

$$y_e^v = y_e^e,\ z_{e_v}^v = z_{e_v}^e,\ \forall v \in \mathcal{V},\ \forall e \in \mathcal{E}_v. \tag{9.4c}$$

$$z_v = \tilde{z}_v\ \forall v \in \mathcal{V},\ z_e = \tilde{z}_e\ \forall e \in \mathcal{E}. \tag{9.4d}$$

9.2 Implementation of the ADMM Steps

We apply Algorithms 1 and 2 to (9.4) by placing the blue variables $(z_v, y_v^v, z_{\mathcal{E}_v}^v, y_{\mathcal{E}_v}^v)$ and the variables $\tilde{z}_e$ on one side of the split. The other side of the split has variables red variables $(z_{e_u}^e, z_{e_v}^e, z_e, y_e^e)$, orange variables y, and $\tilde{z}_v$.

To describe an efficient implementation, we specify how we select our penalty parameter and how we implement Line 1.3 and Line 1.4. For the constraints (9.4d), the constraint update is trivial and the same as in Section 7.3.3. We implement our algorithm in `C++` and distribute it as the package `VEGA`. `VEGA` solves graph of convex sets instances and uses `CCosmo` as its internal convex solver. It is possible to extend `VEGA` to use other solvers besides `CCosmo`.

9.2.1 Selection of the Penalty Functions

The constraints (9.4b) are scalar. For every pair $\lambda = \mu$ in (9.4b) and (9.4c) with dual variable $u_{\lambda\mu}$ we define a penalty parameter $\rho_{\lambda\mu} > 0$ and set $\mathcal{D}_{\rho_{\lambda\mu}}(\lambda, \mu; u_{\lambda\mu}) = \frac{\rho_{\lambda\mu}}{2} \|\lambda - \mu + u_{\lambda\mu}\|_2^2$.

We select the penalty parameters ρ in the following manner. For every vertex v and edge e, we choose $\eta_v > 0$ and $\eta_e > 0$. Additionally, we choose a spatial weight $\kappa_v > 0$ and a flow weight $\omega_v > 0$. We define:

$$\rho_{z_v \tilde{z}_v} = \kappa_v \eta_v \tag{9.5}$$
$$\rho_{z_{e_v}^v z_{e_v}^e} = \kappa_v \eta_e \tag{9.6}$$
$$\rho_{y_v^v y_v} = \omega_v \eta_v \tag{9.7}$$
$$\rho_{y_e^v y_e} = \omega_v \eta_e / 2 \tag{9.8}$$
$$\rho_{y_e^v y_e^e} = \omega_v \eta_e / 2 \tag{9.9}$$
$$\rho_{z_e \tilde{z}_e} = (\rho_{z_{e_u}^u z_{e_u}} + \rho_{z_{e_v}^v z_{e_v}^e})/2 \tag{9.10}$$

This strategy ensures that the scale of the penalty between the binaries and the spatial flow at every graph element remains constant at every vertex.

$$\frac{\rho_{y_v^v y_v}}{\rho_{z_v \tilde{z}_v}} = \frac{\rho_{y_e^v y_e} + \rho_{y_e^v y_e^e}}{\rho_{z_{e_v}^v z_{e_v}^e}} = \frac{\omega_v}{\kappa_v}, \qquad \forall e \in \mathcal{E}_v$$

We choose $\eta_v = 4$, $\eta_e = 4$, $\kappa_v = 0.5$, and $\omega_v = 4$ for all vertices and edges.

9.2.2 Solving the Vertex Programs

Line 1.3 requires solving an instance of

$$\begin{aligned} \min\ & f_v(z_v, y_v^v) + \mathcal{D}_{\rho_{z_v \tilde{z}_v}}(z_v, \tilde{z}_v) + \mathcal{D}_{\rho_{y_v^v y_v}}(y_v^v, y_v^k, u_{y_v^v y_v}^k) + \\ & \sum_{e \in \mathcal{E}_v} \mathcal{D}_{\rho_{y_e^v y_e}}(y_e^v, y_e^k, u_{y_e^v y_e}^k) + \mathcal{D}_{\rho_{y_e^v y_e^e}}(y_e^v, y_e^{ek}, u_{y_e^e y_e}^k) + \\ & \mathcal{D}_{\rho_{z_{e_v}^v z_{e_v}^e}}(z_{e_v}^v, z_e^k, u_{z_{e_v}^v z_{e_v}^e}) \end{aligned} \tag{9.11a}$$

$$\text{subject to } A_v \begin{bmatrix} z_v & z_{\mathcal{E}_v} \\ y_v & y_{\mathcal{E}_v} \end{bmatrix} (A_{\mathfrak{H}}^{vT} - b_{\mathfrak{H}}^{vT}) \in \bigoplus_{i=1}^{m_{\mathfrak{H},\geq}^v} \mathcal{K}_v \tag{9.11b}$$

at every vertex. This program can be solved by vectorizing it into standard form.

Alternatively, under the particular choice of weight (9.5) we can write (9.11) in the form (7.4).

To see this, note that by completing the square, we can write:

$$\mathcal{D}_{\rho_{y_e^v y_e}}(y_e^v, y_e^k, u^k_{y_e^v y_e}) + \mathcal{D}_{\rho_{y_e^v y_e^e}}(y_e^v, y_e^{ek}, u^k_{y_e^v y_e^e}) = \frac{\rho_{y_e^v y_e} + \rho_{y_e^v y_e^e}}{2} \left\| y_e^v - \bar{y}_e^k \right\|_2^2 + c^k_{e_v},$$

where $c^k_{e_v}$ is the left-over constant and

$$\bar{y}_e^k \coloneqq \frac{\rho_{y_e^v y_e}(y_e^k - u^k_{y_e^v y_e}) + \rho_{y_e^v y_e^e}(y_e^{ek} - u^k_{y_e^v y_e^e})}{\rho_{y_e^v y_e} + \rho_{y_e^v y_e^e}}.$$

We define

$$X_v \coloneqq \begin{bmatrix} z_v & z^v_{\mathcal{E}_v} \\ y_v^v & y^v_{\mathcal{E}_v} \end{bmatrix}, \quad S_v \coloneqq \begin{bmatrix} \tilde{z}_v^k & z^k_{\mathcal{E}_v} - u^k_{z^v_{\mathcal{E}_v} z_{\mathcal{E}_v}} \\ y_v^k - u^k_{y_v^v y_v} & \bar{y}^k_{\mathcal{E}_v} \end{bmatrix}$$

and penalty matrices

$$\begin{aligned} D_{l,v} &\coloneqq \operatorname{diag}(\kappa_v I_{n_v}, \omega_v), \\ D_{r,v} &\coloneqq \operatorname{diag}(\eta_v, (\eta_e)_{e \in \mathcal{E}_v}). \end{aligned}$$

The quadratic penalties, after absorbing constants into c_v^k in (9.11a), can be written as

$$\frac{1}{2} \mathbf{tr}\left((X_v - S_v^k)^T D_{l,v} (X_v - S_v^k) D_{r,v} \right) + \tilde{c}_v^k. \tag{9.12}$$

The linear cost can be written as:

$$\mathbf{tr}\left(f_v^T X_v \begin{bmatrix} 1 \\ 0_{\mathcal{E}_v} \end{bmatrix} \right) = \mathbf{tr}\left(\begin{bmatrix} 1 \\ 0_{\mathcal{E}_v} \end{bmatrix} f_v^T X_v \right). \tag{9.13}$$

We emphasize that the transcription of (9.11) into matrix standard form is only possible due to the particular form of $\mathcal{D}$ we have described in Section 9.2.1.

We solve (9.11) with `CCosmo`, by either vectorizing the program if it is sufficiently sparse or as a matrix program if it is sufficiently dense. The solve over the batch of vertices can be parallelized.

9.2.3 Solving the Edge and Graph Programs

Line 1.4 applied to (9.4) requires solving

$$\min\ f_e(z_{e_u}, z_{e_v}, z_e, y_e) + \sum_{v \in e} \mathcal{D}_{\rho_{\hat{y}_e^{v(k+1)} y_e^e}}(\hat{y}_e^{v(k+1)}, y_e^e, u^k_{y_e^e y_e}) + \mathcal{D}_{\rho_{z_{e_v}^v z_{e_v}^e}}(\hat{z}_{e_v}^{v(k+1)}, z_e, u_{z_{e_v}^v z_{e_v}^e}) \tag{9.14a}$$

$$\text{subject to } \mathcal{A}_e(z_{e_u}, z_{e_v}, z_e, y_e) \in \mathcal{K}_e \tag{9.14b}$$

where $\hat{y}_e^{v(k+1)} = \alpha y_e^{v(k+1)} + (1-\alpha) y_e^{ek}$ and similarly for $\hat{z}_{e_v}^{v(k+1)}$, at every edge.

Additionally, Line 1.4 requires solving:

$$\min \sum_{v \in \mathcal{V}} \left(\mathcal{D}_{\rho_{y_v^v y_v}}(y_v^v, y_v^k, u_{y_v^v y_v}^k) + \sum_{e \in \mathcal{E}_v} \mathcal{D}_{\rho_{y_e^v y_e}}(y_e^v, y_e^k, u_{y_e^v y_e}^k) \right) \tag{9.15a}$$

$$\text{subject to } \mathcal{A}_{\mathfrak{H}}(y) - b_{\mathfrak{H}} \geq 0 \tag{9.15b}$$

$$\mathcal{C}_{\mathfrak{H}}(y) - d_{\mathfrak{H}} = 0. \tag{9.15c}$$

This corresponds to Euclidean projection onto the feasible set of the graph problem underlying our GCS instance.

Both (9.14) and (9.15) are conic programs with quadratic costs. We once again use `CCosmo`, solving each edge program and the graph problem in parallel.

9.2.4 Inexact Substeps

ADMM is known to converge even if the substeps Line 1.3 and Line 1.4 are only solved approximately. This was first shown in [21], with a recent modern resurgence in interest [102,223,224].

Rather than solving (9.11), (9.14), and (9.15) to optimality, we adapt the summable forcing-sequence rule of [21,102]. At each iteration k, we require each vertex program to solve to an absolute tolerance of $\epsilon_{v,abs}^k$ and relative tolerance $\epsilon_{v,rel}^k$, each edge program to tolerances $\epsilon_{e,abs}^k$ and $\epsilon_{e,rel}^k$, and each graph program to tolerance $\epsilon_{\mathcal{G},abs}^k$ and $\epsilon_{\mathcal{G},rel}^k$. By [102, Assumption 3], we require that

$$\sum_{k=1}^{\infty} \epsilon_v^k < \infty, \quad \sum_{k=1}^{\infty} \sqrt{\epsilon_{\mathcal{G}}^k + \sum_{e \in \mathcal{E}} \epsilon_e^k} < \infty$$

for both the absolute and relative tolerances.

In practice, we choose $\epsilon_v^k = c_v (k+1)^{-(1+\delta)}$ and $\epsilon_e^k = \epsilon_{\mathcal{G}}^k = c_z (k+1)^{-(2+\delta)}$ for any $\delta > 0$ and some initial c_v and c_z.

9.2.4.1 Closed-Form Updates

It is possible to attain a variant of our algorithm by applying the same split as `CCosmo` to (9.3) before standardizing into ADMM form. We introduce variables s_v, $\tilde{s}_v$ and s_e, $\tilde{s}_e$ and split the conic constraints (9.3b) and (9.3c) resulting in the program

$$\min \sum_{v\in\mathcal{V}} f_v(z_v, y_v^v) + \sum_{e\in\mathcal{E}} f_e(z_{e_u}^e, z_{e_v}^e, z_e, y_e^e) \tag{9.16a}$$

$$\text{subject to } \bar{\mathcal{A}}_v(z_v, y_v^v, z_{\mathcal{E}_v}^v, y_{\mathcal{E}_v}^v) = s_v,\ \forall v \in \mathcal{V} \tag{9.16b}$$

$$\tilde{s}_v \in \mathcal{K}_v,\ \forall v \in \mathcal{V} \tag{9.16c}$$

$$\bar{\mathcal{A}}_e(z_{e_u}, z_{e_v}, z_e, y_e^e) = s_e,\ \forall e \in \mathcal{E} \tag{9.16d}$$

$$\tilde{s}_e \in \mathcal{K}_e\ \forall e \in \mathcal{E} \tag{9.16e}$$

$$\bar{\mathcal{A}}_{\mathcal{G}}(y) - b_{\mathcal{G}} \in \mathcal{K}_{\mathcal{G}} \tag{9.16f}$$

$$y_v^v = y_v,\ y_e^v = y_e,\ \forall v \in \mathcal{V},\ \forall e \in \mathcal{E}_v \tag{9.16g}$$

$$y_e^v = y_e^e,\ z_{e_v}^v = z_{e_v}^e,\ \forall v \in \mathcal{V},\ \forall e \in \mathcal{E}_v. \tag{9.16h}$$

$$s_v = \tilde{s}_v,\ z_v = \tilde{z}_v\ \forall v \in \mathcal{V} \tag{9.16i}$$

$$\tilde{s}_e = s_e,\ \tilde{z}_e = z_e\ \forall e \in \mathcal{E} \tag{9.16j}$$

In this split the blue and purple variables are placed on one side of the split and the red, orange, and green variables are placed on the other. Rather than the steps of the Algorithm 1 being fully `CCosmo` solves, the steps which update blue and red correspond to the linear system solve from Line 5.3 while the updates for the purple and green correspond to the cone projection from Line 5.5. We keep the graph solution as a full projection onto the feasible set of the relaxed underlying graph problem as we observe that this step takes very few internal iterations in practice. We again introduce the regularizers $z_v = \tilde{z}_v$ and $z_e = \tilde{z}_e$ from (9.4).

In summary, with this split Algorithm 1 reduces to

1. Line 1.3:
   (a) Solving a linear system at every vertex in parallel.
   (b) Projecting onto the cone $\mathcal{K}_e$ at every edge in parallel and updating $\tilde{z}_e$.
2. Line 1.4:
   (a) Projecting onto the cone $\mathcal{K}_v$ at every vertex in parallel and updating $\tilde{z}_v$.
   (b) Solving a linear system at every edge in parallel.
   (c) Projecting onto the feasible set of the relaxed graph problem.

All these steps, besides the projection onto the graph problem, can be carried out in closed form. We can also interpret this as an extreme case of solving (9.11) and (9.14) where the solvers for these programs are not even allowed to complete a full iteration before returning.

We note that since (9.16) is a program in the ADMM standard form (4.1), Algorithm 1 is guaranteed to converge asymptotically. The proposed split trades off simplicity of the ADMM steps with an increase in the number of iterations of Algorithm 1. However, as the steps of Algorithm 1 applied to (9.3) are iterative themselves, and internally apply the same steps as (9.16) in a different order, it is worth comparing how much total work both approaches perform.

9.2.5 Termination Criterion

We terminate Algorithm 1 using the criterion from [101, §3.3.1]. We compute primal and dual residuals for (9.4b), (9.4c), and (9.4d).

At iteration k, we define the primal residual for a constraint of the form $\lambda = \mu$ in (9.4b), (9.4c), and (9.4d) as

$$r^k_{p\lambda\mu} = \lambda - \mu \tag{9.17}$$

The dual residual for the pair $\lambda = \mu$ is given by:

$$r^k_{d\lambda\mu} = \rho_{\lambda\mu}(\mu^k - \mu^{k-1}), \tag{9.18}$$

except for the pair of constraints $y^v_e = y_e$ and $y^v_e = y^e_e$ whose dual check must be coupled as

$$r_{dy_e} = \rho_{y^v_e y_e}(y^k_e - y^{k-1}_e) + \rho_{y^v_e y^e_e}(y^{e,k}_e - y^{e,k-1}_e) \tag{9.19}$$

To terminate, we choose thresholds ϵ_{abs} and ϵ_{rel}. Before checking convergence, we check that every tolerance from Section 9.2.4 is below ϵ_{abs} and ϵ_{rel} as needed. We stop updating the tolerances from Section 9.2.4 once they are less than $\epsilon_{abs}/10$ and $\epsilon_{rel}/10$.

Finally, we terminate if for every pair $\lambda = \mu$

$$\|r_{p\lambda\mu}\|_\infty \le \epsilon_{abs} + \epsilon_{rel} \max\{\|\lambda^k\|_\infty, \|\mu^k\|_\infty\} \tag{9.20a}$$
$$\|r_{d\lambda\mu}\|_\infty \le \epsilon_{abs} + \epsilon_{rel} \max\{\|\mu^k\|_\infty, \|\mu^{k-1}\|_\infty\}. \tag{9.20b}$$

For the pair of constraints $y^v_e = y_e$ and $y^v_e = y^e_e$, we replace (9.20b) with

$$\|r_{dy_e}\|_\infty \le \epsilon_{abs} + \epsilon_{rel} \max\{\|y^k_e\|_\infty, \|y^{k-1}_e\|_\infty, \|y^{e,k}_e\|_\infty, \|y^{e,k-1}_e\|_\infty\} \tag{9.21}$$

If we are solving using the updates from Section 9.2.4.1, we must also check the primal and dual convergence of the constraints (9.16i) and (9.16j).

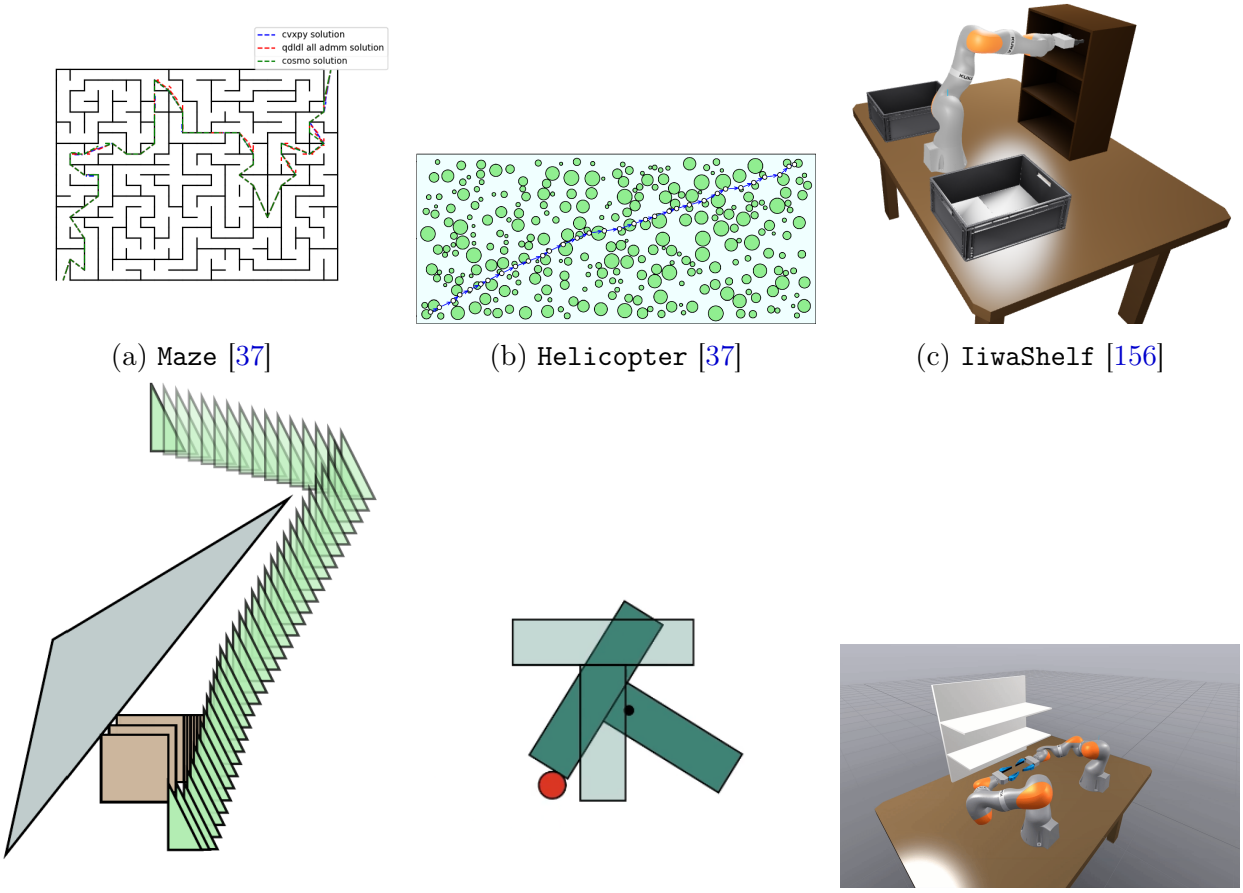
9.3 Test Instances: the `GcsCc` Library and `GcsBench` Instances

Domain-specific languages (DSL) for modelling convex optimization are quite common, with modern efforts emphasizing the integration of the DSL into general-purpose programming languages. Examples include `CVXPY` [216] for `Python`, `JuMP` [314,315] for `Julia`, and `Drake`'s `MathematicalProgram` [69] for `C++` and `Python`. Using these DSLs substantially simplifies the process of modelling a convex optimization program, before standardizing into a given conic form such as (CSF).

To facilitate modelling, GCS libraries combine an appropriate DSL with a graph data structure. Examples include `GCSOPT`[1] [37] which uses `CVXPY`, `Drake`'s native `GraphOfConvexSets`[2], and the in-development `GraphsOfConvexSets.jl` which uses `JuMP`.

[1] https://github.com/TobiaMarcucci/gcsopt

[2] https://drake.mit.edu/doxygen_cxx/classdrake_1_1geometry_1_1optimization_1_1_graph_of_convex_sets.html



(a) `Maze` [37] (b) `Helicopter` [37] (c) `IiwaShelf` [156]

(d) `ContactGraph` [96] (e) `SdpPushing` [95] (f) `Bimanual` [93]

Figure 9.4: Representative `GcsBench` instance families. The `Maze` and `Helicopter` families are trajectory-planning problems from `gcsopt`. The `IiwaShelf` and `Bimanual` examples are collision-free motion planning problems. The `ContactGraph` and `SdpPushing` both consider contact-rich manipulation problems.

While DSLs are useful for modelling optimization programs, they are not as appropriate for distributing programs. Aside from the language-specific nature of a DSL, serializing an optimization program described in a DSL is quite challenging, with `JuMP` only supporting it in a limited format; `CVXPY` and `Drake` do not support it at all. To facilitate sharing instances, researchers have proposed and distributed a number of benchmark formats such as the CBLIB format [316], the SDPA format [317], and MPS format [318]. These formats express optimization programs in a form much closer to the standard form (CSF), a process that inevitably confounds the original structure of the problem based on the specific standardization used.

In order to standardize the transcription and distribution of GCS programs, we introduce `GcsCc`, a `C++` library with `Python` bindings for modelling GCS instances. `GcsCc` models GCS programs by building a graph data structure and storing a program in the conic standard form (CSF) at every vertex and every edge. Given a graph program, `GcsCc` can output the relaxation (3.17) in the form (CSF). To facilitate transcription into the `GcsCc` format, we

provide bridges for converting GCS programs modelled in `GCSOPT` in `Python` and `Drake` in both `Python` and `C++`.

Due to the simple data structures used, `GcsCc` models are relatively easy to serialize and deserialize. This allows us to collect a variety of GCS shortest path instances from the literature and distribute them in a library we call `GcsBench`. `GcsBench` currently contains six families.

We extract the two families `Maze` and `Helicopter` from the examples in `GCSOPT` which are described in [37,38]. To facilitate increasing the complexity of the sets in the `Maze` examples, we extend the example from `GCSOPT` to add velocity continuity constraints on the path as well as penalties on the acceleration. These instances are defined by small programs at each vertex and edge which are homogeneously sized; the programs have almost exactly the same constraints and number of variables. Moreover, the graphs for these programs are extremely sparse.

The remaining four families come from examples using GCS for motion planning in robotics. The `IiwaShelf` examples are adapted from [154, §6.4.4][3] and model a seven-degree-of-freedom robot arm interacting with a shelf. The environment is visualized in Figure 9.4c. The `Bimanual` examples are similar; they model a pair of arms coordinating to manipulate a common manipuland. These examples are visualized in Figure 9.4f. The graphs underlying both of these examples are small and relatively dense. The graphs have only a few dozen vertices and 50–100 edges. The complexity in these instances comes from the heterogeneity of the programs. While the edge programs are relatively uniform, the vertex programs contain dramatically different numbers of constraints.

The final two examples come from the contact-rich manipulation literature. The `SdpPushing` examples are from [95] and visualized in Figure 9.4e. The graphs in these instances are of moderate size. Most of the complexity in these programs comes from the vertices. First, the vertex programs are heterogeneous; some are small SOCPs, others are relatively larger SDPs. These are the only instances in `GcsBench` which have SDPs at the vertices. Finally, the `ContactGraph` examples are taken from [96] and visualized in Figure 9.4d. These programs use a much simpler model of planar pushing, but model manipulating many objects at once. This leads to very simple and uniform vertex and edge programs, but a massive and extremely dense graph. These instances are frequently too large to be solved practically by a centralized solution method without very powerful compute resources.

9.4 Results

In this section, we describe the results of using the `VEGA` in both modes proposed in Section 9.2 and Section 9.2.4.1. We compare to two first-order solvers `SCS` [210,220] and `CCosmo` from Chapter 7, as well as the interior-point solver `Clarabel` [22]. All experiments in this subsection were run on an 11th Gen Intel Core i9-11980HK CPU with 64 GB of RAM. For `SCS` and `CCosmo`, we solve to 10^{-3} absolute tolerance and 10^{-4} relative tolerance. For

[3] see the notebook at https://deepnote.com/workspace/Manipulation-ac8201a1-470a-4c77-afd0-2cc45bc229ff/project/0762b167-402a-4362-9702-7d559f0e73bb/notebook/iris_builder-3c25c10bc29d4c9493e48eaced475d03?secondary-sidebar-autoopen=true&secondary-sidebar=agent

`Clarabel`[4] in high accuracy mode, we solve to the default absolute and relative tolerances of 10^{-8}. For `Clarabel` in low-accuracy mode, we solve to the same tolerance as for `SCS` and `CCosmo`.

We test our `VEGA` and the other solvers on instances from `GcsBench`. For each graph in the benchmark, we select two vertices and form the convex relaxation of the shortest path in a GCS [1]. At each vertex, we include the vertex local constraints $y_v \geq y_e$, $y_e \geq 0$, the flow conservation constraints $y_v + \sum_{e \in \mathcal{E}_v^{in}} y_e = 0$, $y_v + \sum_{e \in \mathcal{E}_v^{out}} y_e = 0$ and the two cycle elimination constraints $y_v \geq y_{uv} + y_{vu}$ for every pair of vertices such that $(u, v), (v, u) \in \mathcal{E}$.

We begin by noting that these programs are incredibly challenging.

9.4.1 Centralized Solver Results

We begin by comparing the runtimes and solution quality of various solvers on the `GcsBench` instances. We exclude all instances from the `ContactGraph` family as the resulting programs are too large, with a single step taking over 2 hours on the smallest instance.

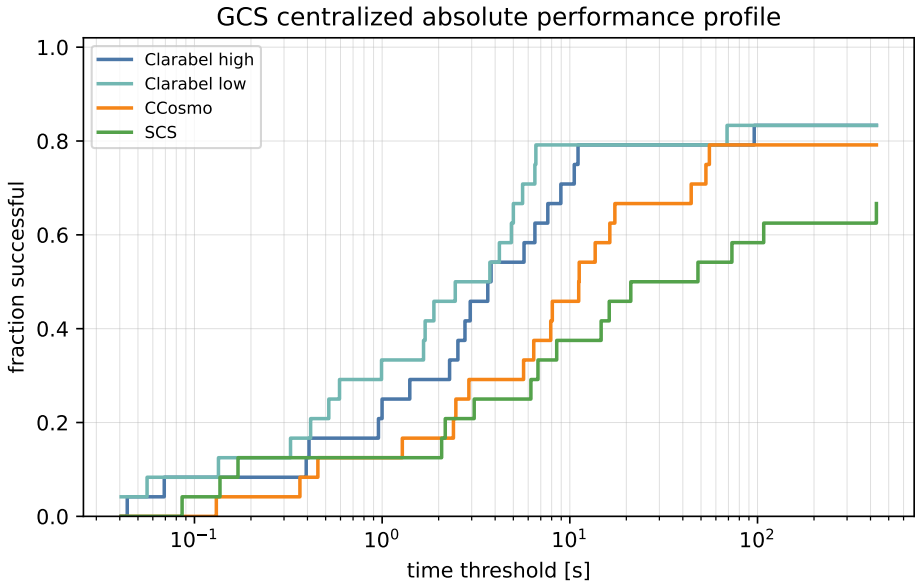


Figure 9.5: Absolute performance profile for centralized solvers on the non-`ContactGraph` `GcsBench` instances.

An absolute performance profile and the shifted geometric mean with one-second shift are shown in Figure 9.5 and Table 9.1. As in Section 7.5, `Clarabel` demonstrates the best performance and robustness on these challenging test instances, with the first-order solvers requiring a surprisingly high number of iterations to converge. `CCosmo` tends to perform better than `SCS`. This may be explained by the fact that the GCS relaxation is already homogeneous, and so the additional homogenization of `SCS` may be hurting in this case.

[4]For the `SdpPushing` family, `Clarabel` encountered numerical difficulties unless `static_regularization_enabled = false`.

Solver	Success	Opt.	Inacc.	Err./Max	Shifted GM [s]
`Clarabel` high	22/24	10	12	2	12.6
`Clarabel` low	20/24	22	0	4	9.8
`CCosmo`	19/24	15	4	5	28.07
`SCS`	16/24	13	11	8	78.16

Table 9.1: Success counts and one-second shifted geometric mean solve times for centralized solvers on the non-`ContactGraph` `GcsBench` instances. We report a success when the returned objective is within 2% of the high-accuracy `Clarabel` reference whenever that reference is available. Opt. and Inacc. report the returned solver statuses, while Err./Max reports unsuccessful rows.

In Tables 9.1 and C.6, we also note the returned statuses of every solver. The solvers mostly return the same answers. However, the very high number of optimal-inaccurate statuses generated by all the solvers (except for `Clarabel` in its low-accuracy mode) point to these problems being quite difficult to solve accurately. Indeed, `Clarabel` in low-accuracy mode is a useful cautionary case on the `SdpPushing` instances: despite reporting successful termination on `SdpPushing`'s box example, it returns a dramatically different objective than the high-accuracy reference.

A detailed breakdown of each solver's performance on these problems is available in Table C.5. In addition, we include a table on each solver's accuracy relative to the high-accuracy baseline in Table C.6.

9.4.2 Regularized Problems

For many programs in `GcsBench`, it is possible to find points that have zero cost in the vertices and edges. Even worse, it is possible to construct cycles in these graphs which achieve zero cost. This makes the optimal solution of the relaxation non-unique and the optimization landscape quite poor. This is one potential source of degeneracy which may be contributing to the reported low confidence by the solvers in the prior section. We demonstrate that eliminating zero cost cycles can contribute to better accuracy of many solvers.

To address the zero-cost cycle issue, we solve exactly the same set of instances, but add a constant $\epsilon = 10$ penalty to each edge. This encourages the shortest path relaxation to visit as few edges as possible and forces the y flow variables to be unique. We again solve the 24 non-`ContactGraph` instances from `GcsBench`.

In Table 9.2 we compare both the shifted geometric mean and the optimality counts after adding the regularization. The effect is not uniform, but the first-order solvers benefit in aggregate: the shifted geometric mean for `CCosmo` improves from 28.07 seconds to 20.31 seconds and the shifted geometric mean for `SCS` improves from 78.16 seconds to 21.59 seconds. Low-accuracy `Clarabel` is slightly slower on this matched set, while high-accuracy `Clarabel` improves.

Moreover, this improved robustness is reflected in the objective values returned by the low-accuracy solvers. In Table 9.3, the mean and maximum relative errors to the `Clarabel` high accuracy solution for `CCosmo` and `SCS` decrease substantially. The low-accuracy `Clarabel`

	Original					Regularized, $\epsilon = 10$				
Solver	Success	Opt.	Inacc.	Err./Max	Shifted GM [s]	Success	Opt.	Inacc.	Err./Max	Shifted GM [s]
`Clarabel` high	22/24	10	1	2	12.6	22/24	20	4	1	10.63
`Clarabel` low	20/24	22	0	4	9.8	23/24	23	0	1	9.376
`CCosmo`	19/24	15	4	5	28.07	20/24	16	1	4	20.31
`SCS`	16/24	13	11	8	78.16	20/24	17	7	4	21.59

Table 9.2: Comparison of solve times, in seconds, on the original and edge-regularized $\epsilon = 10$ problems over the 24 non-`ContactGraph` problems.

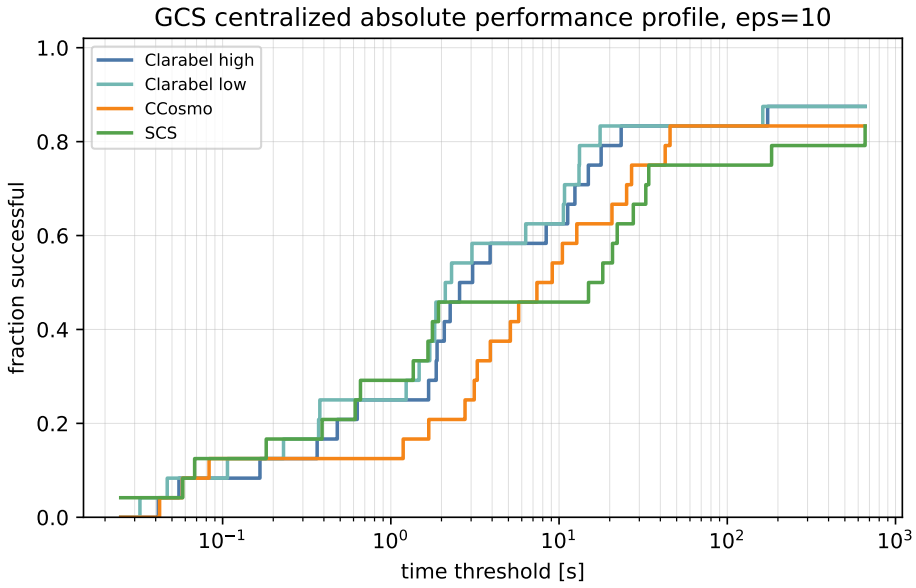


Figure 9.6: Absolute performance profile for centralized solvers on the edge-regularized $\epsilon = 10$ non-`ContactGraph` `GcsBench` instances.

errors remain small in both campaigns, so the main gain is in suppressing the worst degeneracy-sensitive outliers of the first-order solvers.

We again report a detailed breakdown of the solve times in Table C.7 and solution quality in Table C.9. A per-instance ratio of the time taken by each solver on the regularized versus unregularized instances is provided in Table C.8.

Solver	Original			Regularized, $\epsilon = 10$		
	Median err.	Mean err.	Max err.	Median err.	Mean err.	Max err.
`Clarabel` low	0.00906%	0.103%	0.871%	0.0232%	0.139%	1.28%
`CCosmo`	0.0151%	0.211%	1.89%	0.00231%	0.063%	0.811%
`SCS`	0.0299%	1.32%	10.4%	0.00281%	0.194%	2.31%

Table 9.3: Comparison of relative errors to the high-accuracy `Clarabel` solution between the original and edge-regularized $\epsilon = 10$ problems. The table uses the 22 problems where the high-accuracy `Clarabel` reference returned a successful status in both campaigns.

9.4.3 Performance of `VEGA`

In this section, we turn our attention to the performance of `VEGA`. We consider two key metrics. The first is the actual wall-clock time taken by `VEGA` before returning a solution which is the most practically relevant metric. We also report the critical path of the solve time. At every step which has parallelism, we increment the time only by the longest running unit of work in step. This measures the longest serial path in our code, removing the penalty of thread overhead and limited parallelism. This best-case metric gives a sense of whether more parallelism would rescue some `VEGA` solve times.

Overall, we see that `VEGA` is competitive or only modestly slower than `CCosmo` on most of the `GcsBench` instances. The major advantage of `VEGA` becomes clear when considering the very large `ContactGraph` instances where `VEGA` is the only solver capable of solving any of these programs, generating the first known lower bound.

9.4.3.1 Original `VEGA` Instances

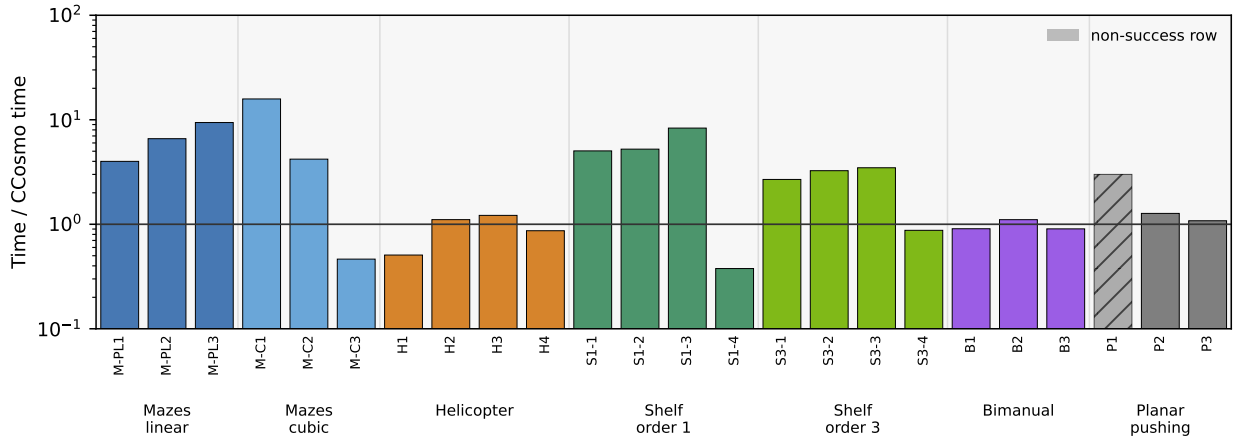
We solve (9.3) with the subproblems solved to the accuracy specified by the relative error criterion in Section 9.2.4. All subproblems are solved in parallel. Each instance in `VEGA` is solved to an absolute tolerance 10^{-3} and relative tolerance 10^{-4}.

We use `VEGA` in two modes. The first allows the subproblems to converge to the prescribed accuracies from Section 9.2.4 or a maximum iteration cap at every iteration. We call this mode the subproblem mode. We also compare to the variant described in Section 9.2.4.1, which interleaves the cone and subspace updates of the edge and vertex programs. We call this the half-step mode. We do not report the critical path for this mode.

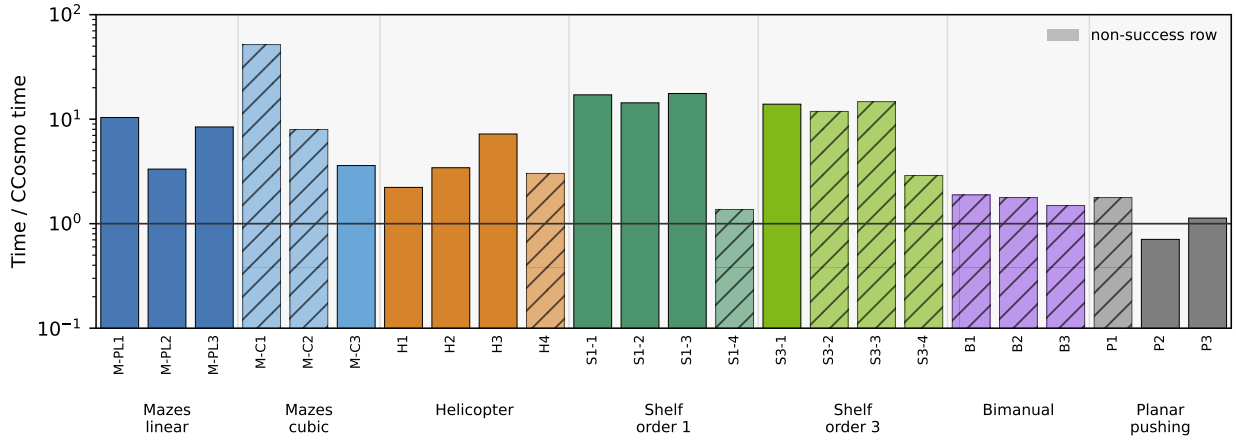
Our results indicate that the subproblem mode is substantially more performant than the half-step mode, achieving a median slowdown relative to `CCosmo` of 1.98 on wall-clock time and is faster than `CCosmo` on 7 instances. These results are summarized in Figure 9.7 with a detailed, per-instance table given in Tables C.10 and C.11. The results along the critical path are also promising. In this case, `VEGA` is faster on 12 of the instances. This is shown in Figure 9.8.

We additionally report the number of outer iterations of `VEGA` compared to the number of iterations required by `CCosmo` in Figure 9.9. This result is mixed, with `CCosmo` requiring more iterations on about half of the instances.

In contrast, though the half-step mode has far lower cost per iteration, it only solves



(a) Subproblem mode.



(b) Half-step mode.

Figure 9.7: Ratio of `VEGA` times to `CCosmo` times on `GcsBench`. Ratios below 1 favor `VEGA`. Hashed bars indicate hitting the iteration limit of `VEGA`.

13/24 problems before reaching its iteration limit and is slower on every problem as shown in Figure 9.7b. This is due to poor tail convergence.

9.4.3.2 Regularized `GcsBench` Instances

We also evaluate `VEGA` on the edge-regularized $\epsilon = 10$ instances from Section 9.4.2. In subproblem mode, `VEGA` solves 19 of the 24 non-`ContactGraph` rows and reaches the iteration limit on 5 rows. The effect is mixed. The shifted geometric mean increases from 20.49s on the original instances to 108.9s on the edge-regularized instances. This is summarized in Table 9.4 and Figure 9.10, with detailed per-instance results in Tables C.14 and C.15.

The critical-path comparison is more favorable than the measured wall-clock comparison, as in the original instances. This suggests that some of the slowdown is due to thread overhead and limited parallelism rather than the serial structure of the split alone. This is shown in Figure 9.11.

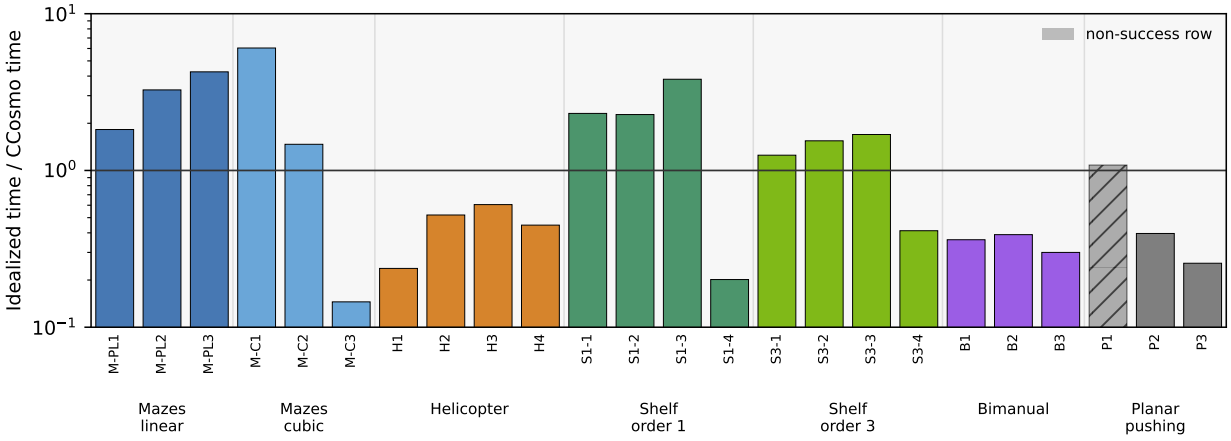


Figure 9.8: Ratio of idealized critical-path times for `VEGA` to `CCosmo` solve times on `GcsBench`. Ratios below 1 favor `VEGA`.

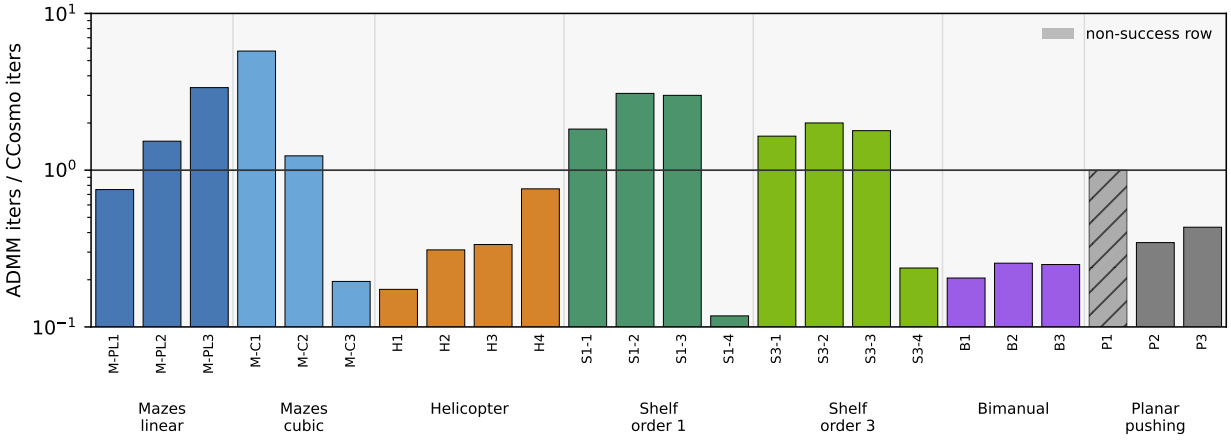


Figure 9.9: Ratio of outer iterations required by `VEGA` to iterations required by `CCosmo` on `GcsBench`.

Family	Num Instances	Orig. solved	Reg. solved	Orig. GM [s]	Reg. GM [s]	Ratio
Mazes linear	3	3	3	4.04	140.0	34.62
Mazes cubic	3	3	3	12.01	60.09	5
`Helicopter`	4	4	2	26.04	303.3	11.65
`IiwaShelf` order 1	4	4	4	15.86	45.15	2.85
`IiwaShelf` order 3	4	4	3	39.84	106.1	2.66
`Bimanual`	3	3	3	8.39	66.27	7.9
`SdpPushing`	3	2	1	148.6	212.8	1.43
Overall	24	23	19	20.49	108.9	5.32

Table 9.4: Effect of edge regularization on `VEGA`.

The half-step mode behaves differently under edge regularization. It solves 15 of the 24 problems, compared with 13 problems on the original instances, and has a slightly smaller shifted geometric mean. However, it is still slower than `CCosmo` on most problems, indicating that the tail-convergence issue remains. The family-level comparison is shown in Table 9.5, and the detailed half-step solves are reported in Tables C.16 and C.17.

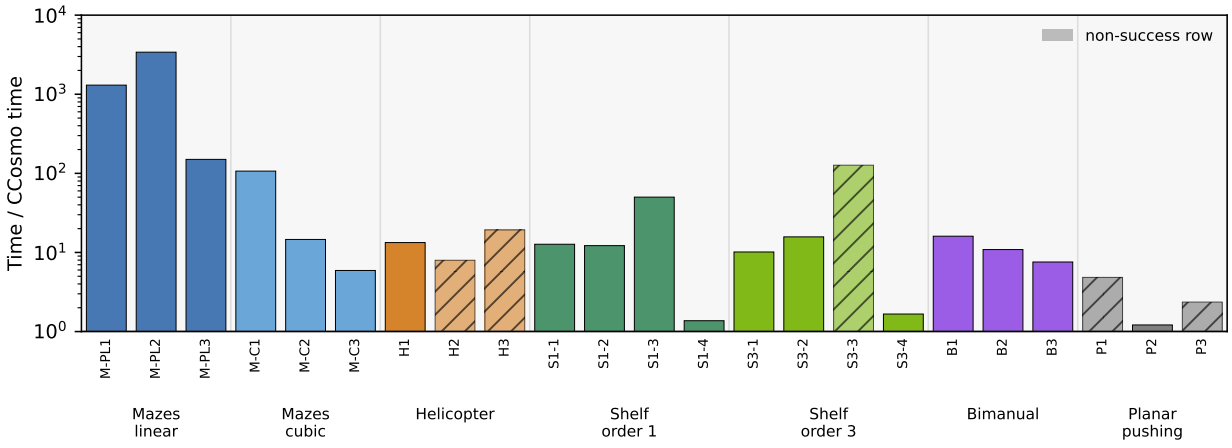
Family	Num Instances	Orig. solved	Reg. solved	Orig. GM [s]	Reg. GM [s]	Ratio
Mazes linear	3	3	3	4.21	4.4	1.04
Mazes cubic	3	1	2	41.70	38.97	0.934
`Helicopter`	4	3	4	103.2	73.77	0.715
`IiwaShelf` order 1	4	3	3	46.05	46.83	1.02
`IiwaShelf` order 3	4	1	1	160.5	157.0	0.978
`Bimanual`	3	0	0	14.75	14.79	1
`SdpPushing`	3	2	2	104.6	122.7	1.17
Overall	24	13	15	46.77	44.89	0.96

Table 9.5: Effect of edge regularization on half-step `VEGA`.

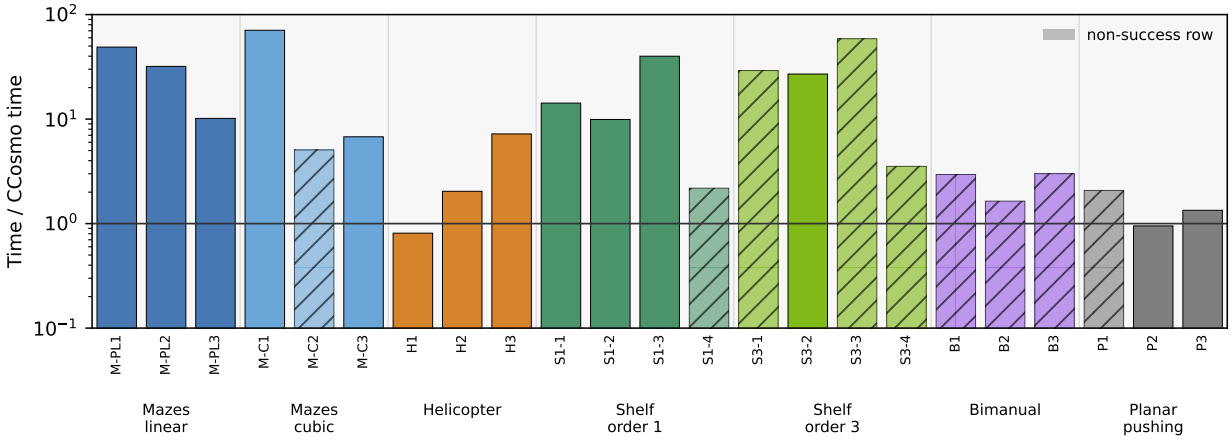
9.4.3.3 `ContactGraph` Instances

`VEGA` shows a clear performance benefit over all the centralized solvers on the smallest of the `ContactGraph` instances. For these instances, no centralized solver was able to take a single iteration on `cg_simple_4` within a 2-hour window.

However, `VEGA` was able to successfully solve the problem in 680s providing the first lower bound for this problem of 11.04. A known upper bound of 27.5 is given in [96].



(a) Subproblem mode.



(b) Half-step mode.

Figure 9.10: Ratio of `VEGA` times to `CCosmo` times on edge-regularized `GcsBench` instances with $\epsilon = 10$. Ratios below 1 favor `VEGA`. Hashed bars indicate hitting the iteration limit of `VEGA`.

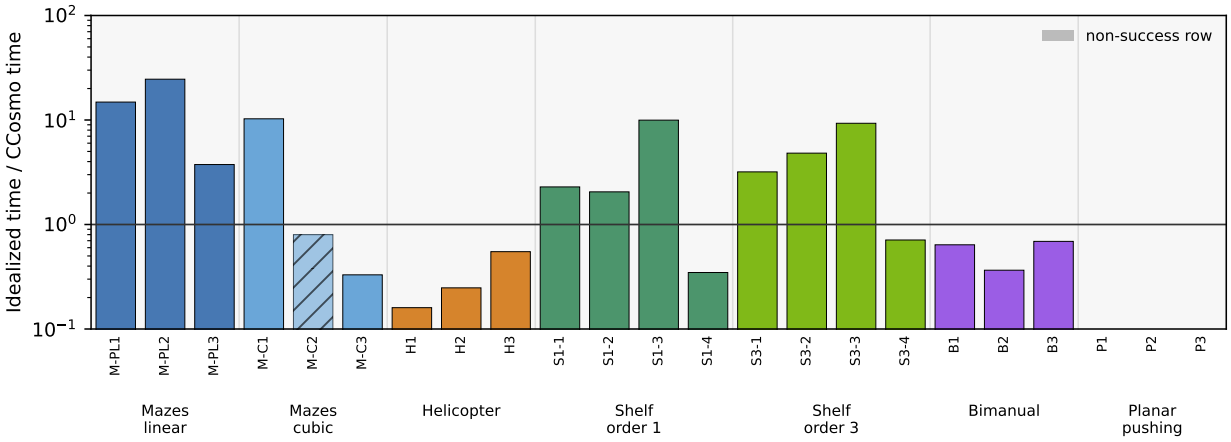
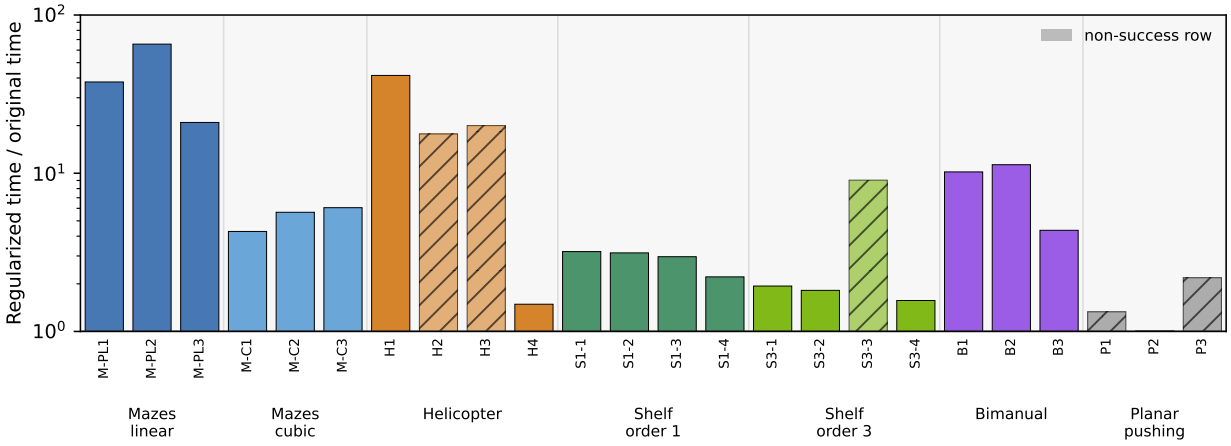
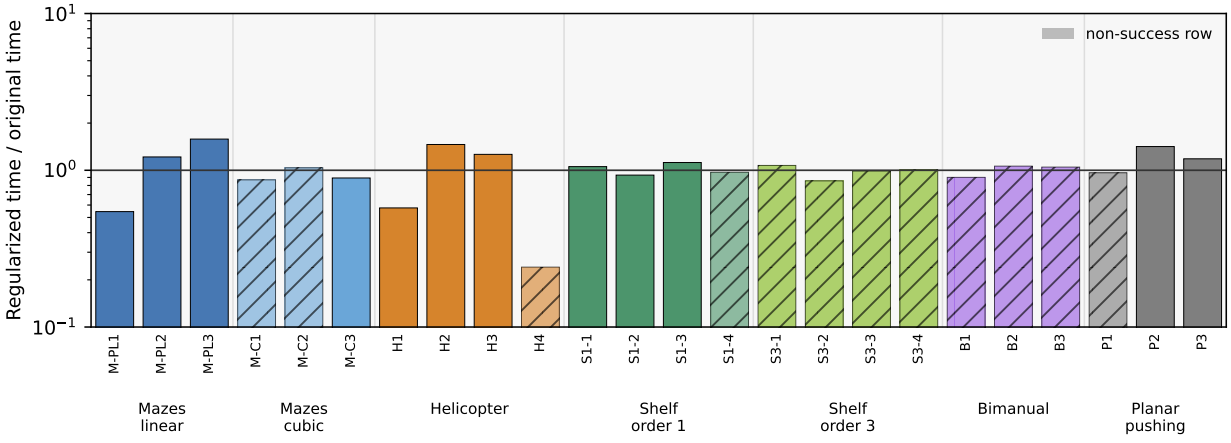


Figure 9.11: Ratio of idealized critical-path times for `VEGA` to `CCosmo` solve times on edge-regularized `GcsBench` instances with $\epsilon = 10$. Ratios below 1 favor `VEGA`.



(a) Subproblem mode.



(b) Half-step mode.

Figure 9.12: Effect of edge regularization on `VEGA` solve time. Values below 1 indicate that the edge-regularized instance is faster.

9.5 Conclusion

In this chapter, we proposed a reformulation of the GCS convex relaxation which naturally partitions the problem into its constituent parts: a convex program with a quadratic cost at every vertex, every edge, and encoding the graph program. We proposed using ADMM to solve the resulting formulation and provided an implementation of our method in the `VEGA` solver. As every iteration of `VEGA` requires solving a convex program per vertex, per edge, and for the flows, `VEGA` uses `CCosmo` internally to implement its steps.

We additionally contributed `GcsCc`, a `C++` and `Python` library for modelling GCS instances in a format that is amenable to serialization and distribution. We used `GcsCc` to collect a benchmark of GCS problem instances from the robotics literature that we distribute as `GcsBench`.

On most of the `GcsBench` instances, `VEGA` shows mixed performance relative to using `CCosmo` to solve the centralized formulation of the GCS convex relaxation. Occasionally, `VEGA` solves problems faster, but it can take much longer. However, timing of the critical path indicates that additional parallelism could substantially improve solve times for many problems, suggesting that the GPU developments of Section 7.6 could be used to close this gap in the future.

Additionally, we demonstrated that though more splitting results in simpler subproblems, this did not lead to any improvements in solve times and indeed led to substantial regressions, especially in the tail convergence. The effect of different choices of splitting on the runtime of ADMM is not well understood in the literature, and this corroborates the folk-wisdom that more splitting can hurt tail convergence.

The primary positive result is that `VEGA` is able to solve certain problems which are completely inaccessible to the general-purpose, centralized convex optimization solvers. Namely, we demonstrated the first-known lower bound on instances in the `ContactGraph` family. Future work will consider solving the larger instances to further corroborate this finding.

Conclusion

In this thesis, we considered the practical use of the Sums-of-Squares/Tensor Product relaxation of polynomial optimization problems in robotics. Though a powerful and general framework, we have argued that this machinery requires thoughtful design choices throughout the entire computational pipeline to be effective in practice; the strongest generic mathematical guarantee is rarely the only relevant objective. The original program must be modeled in a way that exposes exploitable structure. The relaxation itself must be strong enough to provide meaningful information while still remaining tractable to solve. Finally, the choice of numerical method can have a large impact on what relaxations are actually practical to solve in practice. For robotics applications, the decisive question is whether the relaxation can be made accurate, scalable, and dependable enough to become part of a larger planning, control, or verification workflow.

In our first set of contributions in Chapter 5, we demonstrated how polynomial optimization can be used to tractably certify that subsets of a robot's configuration space are collision free. Our formulation made it possible to certify non-collision for polynomial trajectories, certify full-dimensional convex regions in an algebraic reparametrization of configuration space, and grow certified regions. Moreover, we demonstrated an automatic algorithm for placing regions in the configuration in order to build a convex cover of the free space. These covers can be used as building blocks for motion planning. Our SOS-based formulation is the first efficient method for providing full dimensional, certifiably collision-free regions and contributes to the increasingly large literature demonstrating the utility of SOS programming in robotics.

Our second set of contributions focused on the relaxation step itself. In Chapter 6, we developed a lightweight linear-algebraic procedure for discovering candidate redundant equalities for a polynomial optimization problem. Adding such equalities to a POP formulation can substantially strengthen the relaxation. The method automatically discovers candidate relaxations, filters out equalities that cannot strengthen the relaxation, recovers several strengthened formulations already known in the robotics literature, and is fast enough to serve as a preprocessing step before solving the relaxation. The contributions of this chapter emphasize that the original problem formulation cannot be isolated from the relaxation procedure; modeling and relaxation should be considered jointly.

The remaining chapters focused both general convex programs and particular convex relaxations. In Chapter 7, we introduced `CCosmo`, a family of `C++` implementations of a first-order conic optimization method with support for quadratic objectives, standard conic form, matrix standard conic form, and batched GPU execution. The standard-form implementation is competitive with established open-source solvers on several benchmark families, while

the matrix standard form and GPU implementations demonstrate the benefit of specialized numerical paths when the problem structure matches the solver architecture.

Chapter 8 studied numerical linear algebra subroutines that make exploiting matrix structure possible. We studied positive semidefinite generalized Sylvester equations, which appear as a subproblem in `CCosmo`, and demonstrated that such equations can be solved much more efficiently by exploiting both the matrix structure and the positive semidefinite structure. We also studied a combinatorially structured matrix equation that arises from the poset least-squares problem. Our solver exploits order-theoretic structure to obtain an elimination process that is substantially more efficient than a generic sparse solver baseline. Conic solvers tailored to particular structure depend on linear algebra tailored to that structure. When the structure of a relaxation is preserved all the way down to the linear system solve, large apparent problem dimensions can sometimes be replaced by smaller, more meaningful algebraic dimensions.

Finally, in Chapter 9, we returned to Graphs of Convex Sets as a practically important family of tensor-product relaxations. We introduced the `GcsCc` library to provide a structured way to model, serialize, and distribute GCS instances and collected instances from robotics applications into a benchmark called `GcsBench`. We then introduced `VEGA`, a decomposition-based solver for solving large GCS instances. `VEGA` decomposes the GCS relaxation into vertex programs, edge programs, and a graph program, solving these pieces in parallel while driving them to consensus. On small and moderate instances, `VEGA` is modestly slower than general-purpose convex optimization solvers. However, `VEGA` is able to scale to substantially larger instances than the centralized, general-purpose solvers.

The `GcsCc` library and `GcsBench` benchmark provide a structured way to model, serialize, and compare GCS instances drawn from robotics applications. `VEGA` then decomposes the GCS relaxation into vertex programs, edge programs, and a graph program, solving these pieces in parallel while driving them to consensus. The results are deliberately mixed. On small and moderate instances, centralized solvers remain difficult to beat, and the decomposition can introduce enough additional iteration burden to offset the cheaper local steps. On the largest contact-graph instance solved in this thesis, however, centralized solvers were blocked by the scale of the global linear algebra while the decomposed method could still produce a usable low-accuracy solution. This is the intended regime for the method and an instructive example of the thesis theme: decomposition is most valuable when it changes the limiting resource, not merely when it rewrites a problem in distributed form.

Taken together, the thesis argues that though the powerful SOS/TPR framework demonstrates poor scaling in general, pragmatic choices throughout the pipeline can transform seemingly intractable problems into computational problems with efficient, tractable solutions. Each chapter of this thesis addressed a different part of this pipeline, from high-level modeling to low-level linear algebra.

There are many possible directions of future work. First, the ubiquity of polynomial structure is not unique to robotics; it is a pervasive language useful throughout a wide variety of engineering computational domains. Many of the results of this thesis are agnostic to the application area and could be of independent interest other fields including quantum computing, quantitative finance, nuclear science, and more. Bringing the current tools to bear in other fields is an exciting future direction.

Next, in this thesis we have been primarily interested in the static, a-priori construction

of convex relaxations. Future work should consider how to adaptively build better convex relaxations, starting with low-degree, easy-to-solve SOS relaxation and iteratively building tighter relaxations. This is common in the mixed-integer linear and mixed-integer convex programming literature through the branch, cut, and bound algorithm. Its success in those fields has been based on a synergy between research into heuristics for designing strong cuts and excellent software for efficiently resolving the new relaxations.

Finally, in very predictable, highly structured environments, optimization-based methods that rely on careful models continue to dominate. However, robotics has recently seen an immense surge in learning-based methods especially when operating unstructured environments and for tasks that are difficult to model efficiently such as dexterous manipulation. The proposed optimization based methods can continue to be relevant in robotics, even in the era of foundation models and AI agents, as a tool that can be applied and well-understood. Understanding the advantages of synergyzing SOS-relaxation based methods with learning-based policy driven and agentic workflows remains an interesting future direction.

References

[1] T. Marcucci, J. Umenberger, P. Parrilo, and R. Tedrake. "Shortest paths in graphs of convex sets." *SIAM Journal on Optimization*, **34**(1), 2024, pp. 507–532.

[2] I. Newton. *Methodus Fluxionum et Serierum Infinitarum*. Written in 1671, published posthumously. London: Henry Woodfall, 1736.

[3] A.-L. Cauchy. "Méthode générale pour la résolution des systèmes d'équations simultanées." *Comptes Rendus de l'Académie des Sciences*, **25**, 1847, pp. 536–538.

[4] J.-L. Lagrange. *Mécanique Analytique*. Paris: La Veuve Desaint, 1788.

[5] G. B. Dantzig. "Maximization of a linear function of variables subject to linear inequalities." *Activity analysis of production and allocation*, **13**, 1951, pp. 339–347.

[6] W. Karush. "Minima of Functions of Several Variables with Inequalities as Side Constraints." MA thesis. University of Chicago, 1939.

[7] H. W. Kuhn and A. W. Tucker. "Nonlinear Programming." In: *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*. University of California Press, 1951, pp. 481–492.

[8] F. L. Hitchcock. "The Distribution of a Product from Several Sources to Numerous Localities." *Journal of Mathematics and Physics*, **20**(1–4), 1941, pp. 224–230.

[9] Y. Nesterov and A. Nemirovskii. *Interior-point polynomial algorithms in convex programming*. SIAM, 1994.

[10] N. Karmarkar. "A new polynomial-time algorithm for linear programming." *Combinatorica*, **4**(4), 1984, pp. 373–395.

[11] L. G. Khachiyan. "Polynomial algorithms in linear programming." *USSR Computational Mathematics and Mathematical Physics*, **20**(1), 1980, pp. 53–72.

[12] S. J. Wright. *Primal-Dual Interior-Point Methods*. Society for Industrial and Applied Mathematics, 1997.

[13] P. E. Gill, W. Murray, M. A. Saunders, J. A. Tomlin, and M. H. Wright. "On projected Newton barrier methods for linear programming and an equivalence to Karmarkar's projective method." *Mathematical Programming*, **36**(2), 1986, pp. 183–209.

[14] E. W. Dijkstra. "A note on two problems in connexion with graphs." *Numerische Mathematik*, **1**(1), 1959, pp. 269–271.

[15] R. Bellman. "On a routing problem." *Quarterly of applied mathematics*, **16**(1), 1958, pp. 87–90.

[16] L. R. Ford. *Network Flow Theory*. Tech. rep. P-923. Rand Corporation, 1956.

[17] A. V. Goldberg and R. E. Tarjan. "A new approach to the maximum flow problem." *Journal of the ACM (JACM)*, **35**(4), 1988, pp. 921–940.

[18] R. M. Karp. "Reducibility Among Combinatorial Problems." In: *Complexity of Computer Computations*. Ed. by R. E. Miller and J. W. Thatcher. Plenum Press, 1972, pp. 85–103.

[19] J. Canny. "Some algebraic and geometric computations in PSPACE." In: *Proceedings of the twentieth annual ACM symposium on Theory of computing*. 1988, pp. 460–467.

[20] R. T. Rockafellar. *Convex Analysis*. Vol. 28. Princeton Mathematical Series. Princeton, NJ: Princeton University Press, 1970.

[21] J. Eckstein and D. P. Bertsekas. "On the Douglas–Rachford splitting method and the proximal point algorithm for maximal monotone operators." *Mathematical Programming*, **55**(1), 1992, pp. 293–318.

[22] P. J. Goulart and Y. Chen. "Clarabel: An interior-point solver for conic programs with quadratic objectives." *arXiv preprint arXiv:2405.12762*, 2024.

[23] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. "OSQP: An operator splitting solver for quadratic programs." *Mathematical Programming Computation*, **12**(4), 2020, pp. 637–672.

[24] L. Lovász and A. Schrijver. "Cones of matrices and set-functions and 0-1 optimization." *SIAM Journal on Optimization*, **1**(2), 1991, pp. 166–190.

[25] H. D. Sherali and W. P. Adams. "A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems." *SIAM Journal on Discrete Mathematics*, **3**(3), 1990, pp. 411–430.

[26] P. A. Parrilo. "Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization." PhD thesis. California Institute of Technology, 2000.

[27] J. B. Lasserre. "Global optimization with polynomials and the problem of moments." *SIAM Journal on Optimization*, **11**(3), 2001, pp. 796–817.

[28] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts. "LQR-trees: Feedback motion planning via sums-of-squares verification." *The International Journal of Robotics Research*, **29**(8), 2010, pp. 1038–1052.

[29] D. Henrion and M. Korda. "Convex computation of the region of attraction of polynomial control systems." *IEEE Transactions on Automatic Control*, **59**(2), 2013, pp. 297–312.

[30] S. Prajna. "Optimization-based methods for nonlinear and hybrid systems verification." PhD thesis. California Institute of Technology, 2005.

[31] B. Landry, M. Chen, S. Hemley, and M. Pavone. "Reach-avoid problems via sum-of-squares optimization and dynamic programming." In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 4325–4332.

[32] H. Yang, J. Shi, and L. Carlone. "TEASER: Fast and certifiable point cloud registration." *IEEE Transactions on Robotics*, **37**(2), 2020, pp. 314–333.

[33] D. M. Rosen, L. Carlone, A. S. Bandeira, and J. J. Leonard. "SE-Sync: A certifiably correct algorithm for synchronization over the special Euclidean group." *The International Journal of Robotics Research*, **38**(2-3), 2019, pp. 95–125.

[34] S. Kang, G. Liu, and H. Yang. "Global contact-rich planning with sparsity-rich semidefinite relaxations." *arXiv preprint arXiv:2502.02829*, 2025.

[35] A. Majumdar, A. A. Ahmadi, and R. Tedrake. "Control design along trajectories with sums of squares programming." In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 4054–4061.

[36] A. Majumdar, G. Hall, and A. A. Ahmadi. "Recent Scalability Improvements for Semidefinite Programming with Applications in Machine Learning, Control, and Robotics." *Annual Review of Control, Robotics, and Autonomous Systems*, **3**, 2020, pp. 331–360. DOI: 10.1146/annurev-control-091819-074326.

[37] T. Marcucci. "A Unified and Scalable Method for Optimization over Graphs of Convex Sets." *arXiv preprint arXiv:2510.20184*, 2025.

[38] T. Marcucci. "Graphs of Convex Sets with Applications to Optimal Control and Motion Planning." PhD thesis. Massachusetts Institute of Technology, 2024.

[39] D. Gabay and B. Mercier. "A dual algorithm for the solution of nonlinear variational problems via finite element approximation." *Computers & Mathematics with Applications*, **2**(1), 1976, pp. 17–40.

[40] M. Garstka, M. Cannon, and P. Goulart. "COSMO: A conic operator splitting method for convex conic problems." *Journal of Optimization Theory and Applications*, **190**(3), 2021, pp. 779–810.

[41] R. A. Horn and C. R. Johnson. *Matrix analysis*. Cambridge University Press, 2012.

[42] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[43] A. Ben-Tal and A. Nemirovski. *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications*. MPS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics, 2001.

[44] D. P. Bertsekas. *Convex Optimization Theory*. Belmont, MA: Athena Scientific, 2009.

[45] S. Roman. *Advanced linear algebra*. Springer, 2005.

[46] D. S. Dummit and R. M. Foote. *Abstract Algebra*. 3rd ed. Hoboken, NJ: John Wiley & Sons, 2003.

[47] R. A. Horn and C. R. Johnson. *Topics in Matrix Analysis*. Cambridge, UK: Cambridge University Press, 1994.

[48] G. Aubrun, L. Lami, C. Palazuelos, and M. Plávala. "Entangleability of cones." *Geometric and Functional Analysis*, **31**(2), 2021, pp. 181–205.

[49] L. Gurvits. "Classical deterministic complexity of Edmonds' problem and quantum entanglement." In: *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*. 2003, pp. 10–19.

[50] R. Hildebrand. "An LMI description for the cone of Lorentz-positive maps II." *Linear and Multilinear Algebra*, **59**(7), 2011, pp. 719–731.

[51] R. Hildebrand. "An LMI description for the cone of Lorentz-positive maps." *Linear and Multilinear Algebra*, **55**(6), 2007, pp. 551–573.

[52] G. Blekherman, P. A. Parrilo, and R. R. Thomas. *Semidefinite optimization and convex algebraic geometry*. SIAM, 2012.

[53] P. Comon, G. Golub, L.-H. Lim, and B. Mourrain. "Symmetric Tensors and Symmetric Tensor Rank." *SIAM Journal on Matrix Analysis and Applications*, **30**(3), 2008, pp. 1254–1279.

[54] K. G. Murty and S. N. Kabadi. "Some NP-complete problems in quadratic and nonlinear programming." *Mathematical Programming*, **39**(2), 1987, pp. 117–129.

[55] A. A. Ahmadi, A. Olshevsky, P. A. Parrilo, and J. N. Tsitsiklis. "NP-hardness of deciding convexity of quartic polynomials and related problems." *Mathematical Programming*, **137**(1), 2013, pp. 453–476.

[56] D. Hilbert. "Über die darstellung definiter formen als summe von formenquadraten." *Mathematische Annalen*, **32**(3), 1888, pp. 342–350.

[57] T. S. Motzkin. "The arithmetic-geometric inequality." In: *Inequalities*. Ed. by O. Shisha. New York: Academic Press, 1967, pp. 205–224.

[58] M.-D. Choi. "Positive semidefinite biquadratic forms." *Linear Algebra and Its Applications*, **12**(2), 1975, pp. 95–100.

[59] C. J. Hillar and L.-H. Lim. "Most tensor problems are NP-hard." *Journal of the ACM (JACM)*, **60**(6), 2013, pp. 1–39.

[60] MOSEK ApS. *MOSEK Modeling Cookbook*. Chapter: Conic quadratic optimization. 2026. URL: https://docs.mosek.com/modeling-cookbook/cqo.html.

[61] P. A. Parrilo. "Semidefinite programming relaxations for semialgebraic problems." *Mathematical Programming*, **96**(2), 2003, pp. 293–320.

[62] J. B. Lasserre. *Moments, positive polynomials and their applications*. Vol. 1. World Scientific, 2009.

[63] P. A. Parrilo. "Exploiting structure in sum of squares programs." In: *42nd IEEE International Conference on Decision and Control*. Vol. 5. 2003, pp. 4664–4669.

[64] Y. Nesterov. "Squared functional systems and optimization problems." In: *High Performance Optimization*. Springer, 2000, pp. 405–440.

[65] L. Vandenberghe and S. Boyd. "Semidefinite programming." *SIAM Review*, **38**(1), 1996, pp. 49–95.

[66] T. Weisser, B. Legat, C. Coey, L. Kapelevich, and J. P. Vielma. "Polynomial and Moment Optimization in Julia and JuMP." In: *JuliaCon*. 2019. URL: https://pretalx.com/juliacon2019/talk/QZBKAU/.

[67] B. Legat, C. Coey, R. Deits, J. Huchette, and A. Perry. "Sum-of-squares optimization in Julia." In: *The First Annual JuMP-dev Workshop*. 2017.

[68] C. Yuan. *SumOfSquares.py*. 2024. URL: https://github.com/yuanchenyang/SumOfSquares.py.

[69] R. Tedrake and the Drake Development Team. *Drake: Model-based design and verification for robotics*. 2019. URL: https://drake.mit.edu.

[70] A. Papachristodoulou, J. Anderson, G. Valmorbida, S. Prajna, P. Seiler, P. A. Parrilo, M. M. Peet, and D. Jagt. *SOSTOOLS: Sum of squares optimization toolbox for MATLAB*. Available from https://github.com/oxfordcontrol/SOSTOOLS. 2021. URL: http://arxiv.org/abs/1310.4716.

[71] G. Stengle. "A Nullstellensatz and a Positivstellensatz in semialgebraic geometry." *Mathematische Annalen*, **207**(2), 1974, pp. 87–97.

[72] J.-L. Krivine. "Anneaux préordonnés." *Journal d'Analyse Mathématique*, **12**(1), 1964, pp. 307–326.

[73] K. Schmüdgen. "The K-moment problem for compact semi-algebraic sets." *Mathematische Annalen*, **289**(1), 1991, pp. 203–206.

[74] M. Putinar. "Positive polynomials on compact semi-algebraic sets." *Indiana University Mathematics Journal*, **42**(3), 1993, pp. 969–984.

[75] G. Stengle. "Complexity estimates for the Schmüdgen Positivstellensatz." *Journal of Complexity*, **12**(2), 1996, pp. 167–174.

[76] A. Prestel and C. Delzell. *Positive polynomials: From Hilbert's 17th problem to real algebra*. Springer Science & Business Media, 2013.

[77] E. W. Mayr and A. R. Meyer. "The complexity of the word problems for commutative semigroups and polynomial ideals." *Advances in Mathematics*, **46**(3), 1982, pp. 305–329.

[78] J. Koh. "Ideals generated by quadrics exhibiting double exponential degrees." *Journal of Algebra*, **200**(1), 1998, pp. 225–245.

[79] J. Krajíček. *Proof complexity*. Vol. 170. Cambridge University Press, 2019.

[80] M. Clegg, J. Edmonds, and R. Impagliazzo. "Using the Groebner basis algorithm to find proofs of unsatisfiability." In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. ACM Press, 1996, pp. 174–183. URL: https://doi.org/10.1145/237814.237860.

[81] M. Soltys and S. Cook. "The proof complexity of linear algebra." *Annals of Pure and Applied Logic*, **130**(1-3), 2004, pp. 277–323. URL: https://doi.org/10.1016/j.apal.2003.10.018.

[82] D. Grigoriev, E. A. Hirsch, and D. V. Pasechnik. "Complexity of semi-algebraic proofs." In: *Annual Symposium on Theoretical Aspects of Computer Science*. Springer. 2002, pp. 419–430.

[83] A. Fawzi, M. Malinowski, H. Fawzi, and O. Fawzi. "Learning dynamic polynomial proofs." *Advances in Neural Information Processing Systems*, **32**, 2019.

[84] Y. Chen, D. Tse, P. Nobel, P. Goulart, and S. Boyd. "CuClarabel: GPU acceleration for a conic optimization solver." *arXiv preprint arXiv:2412.19027*, 2024.

[85] MOSEK ApS. *The MOSEK optimization toolbox for MATLAB manual. Version 9.0.* 2019. URL: http://docs.mosek.com/9.0/toolbox/index.html.

[86] N. Z. Shor. "Quadratic optimization problems." *Soviet Journal of Computer and Systems Sciences*, **25**, 1987, pp. 1–11.

[87] H. Fawzi. "On polyhedral approximations of the positive semidefinite cone." *Mathematics of Operations Research*, **46**(4), 2021, pp. 1479–1489.

[88] D. Song and P. A. Parrilo. "On approximations of the PSD cone by a polynomial number of smaller-sized PSD cones." *arXiv preprint arXiv:2105.02080*, 2021.

[89] H. Fawzi. "On representing the positive semidefinite cone using the second-order cone." *Mathematical Programming*, **175**(1), 2019, pp. 109–118.

[90] A. C. Doherty, P. A. Parrilo, and F. M. Spedalieri. "Detecting multipartite entanglement." *Physical Review A: Atomic, Molecular, and Optical Physics*, **71**(3), 2005, p. 032333.

[91] A. A. Ahmadi and A. Majumdar. "DSOS and SDSOS optimization: more tractable alternatives to sum of squares and semidefinite optimization." *SIAM Journal on Applied Algebra and Geometry*, **3**(2), 2019, pp. 193–230.

[92] T. Marcucci, M. Petersen, D. von Wrangel, and R. Tedrake. "Motion planning around obstacles with convex optimization." *Science Robotics*, **8**(84), 2023, eadf7843.

[93] T. Cohn, S. Shaw, M. Simchowitz, and R. Tedrake. "Constrained bimanual planning with analytic inverse kinematics." In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2024, pp. 6935–6942.

[94] R. Natarajan, C. Liu, H. Choset, and M. Likhachev. "Implicit graph search for planning on graphs of convex sets." *arXiv preprint arXiv:2410.08909*, 2024.

[95] B. P. Graesdal, S. Y. C. Chia, T. Marcucci, S. Morozov, A. Amice, P. A. Parrilo, and R. Tedrake. "Towards tight convex relaxations for contact-rich manipulation." *arXiv preprint arXiv:2402.10312*, 2024.

[96] S. Y. C. Chia, R. H. Jiang, B. P. Graesdal, L. P. Kaelbling, and R. Tedrake. "GCS*: Forward heuristic search on implicit graphs of convex sets." *arXiv preprint arXiv:2407.08848*, 2024.

[97] J. Tang, Z. Mao, L. Yang, and H. Ma. "Space-time graphs of convex sets for multi-robot motion planning." In: *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2025, pp. 8683–8690.

[98] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows*. Tech. rep. Alfred P. Sloan School of Management, Massachusetts Institute of Technology, 1988.

[99] D. von Wrangel. "Guiding nonconvex trajectory optimization with hierarchical graphs of convex sets." PhD thesis. Massachusetts Institute of Technology, 2024.

[100] D. von Wrangel and R. Tedrake. "Using graphs of convex sets to guide nonconvex trajectory optimization." In: *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2024, pp. 9863–9870.

[101] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, et al. "Distributed optimization and statistical learning via the alternating direction method of multipliers." *Foundations and Trends® in Machine Learning*, **3**(1), 2011, pp. 1–122.

[102] Z. Frangella, T. Diamandis, B. Stellato, and M. Udell. "On the (linear) convergence of generalized Newton inexact ADMM." *arXiv preprint arXiv:2302.03863*, 2023.

[103] D. Sun, Y. Yuan, G. Zhang, and X. Zhao. "Accelerating preconditioned ADMM via degenerate proximal point mappings." *SIAM Journal on Optimization*, **35**(2), 2025, pp. 1165–1193.

[104] W. Deng and W. Yin. "On the global and linear convergence of the generalized alternating direction method of multipliers." *Journal of Scientific Computing*, **66**(3), 2016, pp. 889–916.

[105] P. Giselsson and S. Boyd. "Linear convergence and metric selection for Douglas-Rachford splitting and ADMM." *IEEE Transactions on Automatic Control*, **62**(2), 2016, pp. 532–544.

[106] D. Applegate, O. Hinder, H. Lu, and M. Lubin. "Faster first-order primal-dual methods for linear programming using restarts and sharpness." *Mathematical Programming*, **201**(1), 2023, pp. 133–184.

[107] S. Kang, X. Jiang, and H. Yang. "Local linear convergence of the alternating direction method of multipliers for semidefinite programming under strict complementarity." *arXiv preprint arXiv:2503.20142*, 2025.

[108] B. Halpern. "Fixed points of nonexpanding maps." *Bulletin of the American Mathematical Society*, **73**, 1967, pp. 957–961.

[109] R. I. Bot and D.-K. Nguyen. "Fast Krasnosel'skii-Mann algorithm with a convergence rate of the fixed point iteration of $o\left(\frac{1}{k}\right)$." *arXiv preprint arXiv:2206.09462*, 2022.

[110] H. Lu and J. Yang. "Restarted Halpern PDHG for linear programming." *arXiv preprint arXiv:2407.16144*, 2024.

[111] D. Applegate, M. Díaz, O. Hinder, H. Lu, M. Lubin, B. O'Donoghue, and W. Schudy. "PDLP: A Practical First-Order Method for Large-Scale Linear Programming." *arXiv preprint arXiv:2501.07018*, 2025.

[112] E. K. Ryu and W. Yin. *Large-scale convex optimization: algorithms & analyses via monotone operators*. Cambridge University Press, 2022.

[113] T. Lozano-Perez. "Spatial Planning: A Configuration Space Approach." *IEEE Transactions on Computers*, **100**(32), 1983.

[114] C. W. Wampler and A. J. Sommese. "Numerical algebraic geometry and algebraic kinematics." *Acta Numerica*, **20**, 2011.

[115] A. Amice, H. Dai, P. Werner, A. Zhang, and R. Tedrake. "Finding and optimizing certified, collision-free regions in configuration space for robot manipulators." In: *Algorithmic Foundations of Robotics XV: Proceedings of the Fifteenth Workshop on the Algorithmic Foundations of Robotics*. Springer. 2022, pp. 328–348.

[116] H. Dai, A. Amice, P. Werner, A. Zhang, and R. Tedrake. "Certified polyhedral decompositions of collision-free configuration space." *The International Journal of Robotics Research*, **43**(9), 2024, pp. 1322–1341.

[117] A. Amice, P. Werner, and R. Tedrake. "Certifying bimanual RRT motion plans in a second." In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2024, pp. 9293–9299.

[118] P. Werner, A. Amice, T. Marcucci, D. Rus, and R. Tedrake. "Approximating robot configuration spaces with few convex sets using clique covers of visibility graphs." In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2024, pp. 10359–10365.

[119] J. Canny. *The complexity of robot motion planning*. MIT Press, 1988.

[120] L. E. Kavraki. "Computation of configuration-space obstacles using the fast Fourier transform." *IEEE Transactions on Robotics and Automation*, **11**(3), 1995, pp. 408–413.

[121] M. S. Branicky and W. S. Newman. "Rapid computation of configuration space obstacles." In: *Proceedings of the IEEE International Conference on Robotics and Automation*. Vol. 1. 1990, pp. 304–310.

[122] J.-C. Latombe. *Robot motion planning*. Vol. 124. Springer Science & Business Media, 2012.

[123] R. Deits and R. Tedrake. "Efficient mixed-integer planning for UAVs in cluttered environments." In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 42–49.

[124] T. Schouwenaars, B. De Moor, E. Feron, and J. P. How. "Mixed integer programming for multi-vehicle path planning." In: *6th European Control Conference, ECC 2001, Porto, Portugal, September 4–7, 2001*. IEEE, 2001, pp. 2603–2608. DOI: 10.23919/ECC.2001.7076321.

[125] S. M. LaValle. *Rapidly-Exploring Random Trees: A New Tool for Path Planning*. Tech. rep. TR 98-11. Ames, IA, USA: Iowa State University, 1998.

[126] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces." *IEEE Transactions on Robotics and Automation*, **12**(4), 1996, pp. 566–580.

[127] J. Pan, S. Chitta, and D. Manocha. "FCL: A general purpose library for collision and proximity queries." In: *2012 IEEE International Conference on Robotics and Automation*. 2012, pp. 3859–3866.

[128] S. Tarbouriech and W. Suleiman. "On bisection continuous collision checking method: Spherical joints and minimum distance to obstacles." In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 7613–7619.

[129] P. Jiménez, F. Thomas, and C. Torras. "3D collision detection: a survey." *Computers & Graphics*, **25**(2), 2001, pp. 269–285.

[130] C. L. Jackins and S. L. Tanimoto. "Oct-trees and their use in representing three-dimensional objects." *Computer Graphics and Image Processing*, **14**(3), 1980, pp. 249–270.

[131] M. Tang, Y. J. Kim, and D. Manocha. "CCQ: Efficient local planning using connection collision query." In: *Algorithmic Foundations of Robotics IX: Selected Contributions of the Ninth International Workshop on the Algorithmic Foundations of Robotics*. Springer. 2011, pp. 229–247.

[132] J. Pan, L. Zhang, D. Manocha, and U. Hill. "Collision-free and curvature-continuous path smoothing in cluttered environments." *Robotics: Science and Systems VII*, **17**, 2012, p. 233.

[133] F. Schwarzer, M. Saha, and J.-C. Latombe. "Exact collision checking of robot paths." In: *Algorithmic Foundations of Robotics V*. Springer, 2004, pp. 25–41.

[134] X. Zhang, S. Redon, M. Lee, and Y. J. Kim. "Continuous collision detection for articulated models using Taylor models and temporal culling." *ACM Transactions on Graphics (TOG)*, **26**(3), 2007, 15–es.

[135] J. Canny. "Collision detection for moving polyhedra." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (2), 1986, pp. 200–209.

[136] A. Schweikard. "Polynomial time collision detection for manipulator paths specified by joint motions." *IEEE Transactions on Robotics and Automation*, **7**(6), 1991, pp. 865–870.

[137] M. Verghese, N. Das, Y. Zhi, and M. Yip. "Configuration Space Decomposition for Scalable Proxy Collision Checking in Robot Planning and Control." *arXiv preprint arXiv:2201.04314*, 2022.

[138] Y. Han, W. Zhao, J. Pan, Z. Ye, R. Yi, and Y.-J. Liu. "A configuration-space decomposition scheme for learning-based collision checking." *arXiv preprint arXiv:1911.08581*, 2019.

[139] T. H. Wong, G. Leach, and F. Zambetta. "An adaptive octree grid for GPU-based collision detection of deformable objects." *The Visual Computer*, **30**(6), 2014, pp. 729–738.

[140] M. Abrahamsen. "Covering polygons is even harder." In: *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE. 2022, pp. 375–386.

[141] A. Lingas. "The power of non-rectilinear holes." In: *International Colloquium on Automata, Languages, and Programming*. Springer. 1982, pp. 369–383.

[142] S. J. Eidenbenz and P. Widmayer. "An approximation algorithm for minimum convex cover with logarithmic performance guarantee." *SIAM Journal on Computing*, **32**(3), 2003, pp. 654–670.

[143] J.-M. Lien and N. M. Amato. "Approximate convex decomposition of polyhedra." In: *Proceedings of the 2007 ACM symposium on Solid and physical modeling*. 2007, pp. 121–131.

[144] M. Ghosh, N. M. Amato, Y. Lu, and J.-M. Lien. "Fast approximate convex decomposition using relative concavity." *Computer-Aided Design*, **45**(2), 2013, pp. 494–504.

[145] R. Deits and R. Tedrake. "Computing large convex regions of obstacle-free space through semidefinite programming." In: *Algorithmic Foundations of Robotics XI*. Springer, 2015, pp. 109–124.

[146] P. Werner, R. Cheng, T. Stewart, R. Tedrake, and D. Rus. "Superfast configuration-space convex set computation on GPUs for online motion planning." *arXiv preprint arXiv:2504.10783*, 2025.

[147] P. Werner, T. Cohn, R. H. Jiang, T. Seyde, M. Simchowitz, R. Tedrake, and D. Rus. "Faster algorithms for growing collision-free convex polytopes in robot configuration space." *arXiv preprint arXiv:2410.12649*, 2024.

[148] M. Marshall. *Positive polynomials and sums of squares*. 146. American Mathematical Soc., 2008.

[149] K. Mamou and F. Ghorbel. "A simple and efficient approach for 3D mesh approximate convex decomposition." In: *2009 16th IEEE international conference on image processing (ICIP)*. 2009, pp. 3501–3504.

[150] S. Brossette and P.-B. Wieber. "Collision avoidance based on separating planes for feet trajectory generation." In: *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*. 2017, pp. 509–514.

[151] X. Lin, G. I. Fernandez, and D. W. Hong. "Reduce: Reformulation of mixed integer programs using data from unsupervised clusters for learning efficient strategies." In: *2022 International Conference on Robotics and Automation (ICRA)*. 2022, pp. 4459–4465.

[152] K. Tracy, T. A. Howell, and Z. Manchester. "Differentiable collision detection for a set of convex primitives." In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 3663–3670.

[153] J. J. Craig. *Introduction to robotics: mechanics and control*. Pearson Educacion, 2005.

[154] R. Tedrake. *Robotic Manipulation. Perception, Planning, and Control*. 2021. URL: https://manipulation.mit.edu/pick.html.

[155] M. Spivak. *Calculus*. 3rd ed. Cambridge University Press, 1994.

[156] R. Tedrake. *Underactuated Robotics. Algorithms for Walking, Running, Swimming, Flying, and Manipulation*. 2022. URL: https://underactuated.csail.mit.edu.

[157] J. Nie. "Polynomial Matrix Inequality and Semidefinite Representation." *Mathematics of Operations Research*, **36**(3), 2011, pp. 398–415.

[158] P. A. Parrilo. "Sum of squares programs and polynomial inequalities." In: *SIAG/OPT Views-and-News: A Forum for the SIAM Activity Group on Optimization*. Vol. 15. 2. 2004, pp. 7–15.

[159] T. Roh and L. Vandenberghe. "Discrete transforms, semidefinite programming, and sum-of-squares representations of nonnegative polynomials." *SIAM Journal on Optimization*, **16**(4), 2006, pp. 939–964.

[160] M.-D. Choi, T.-Y. Lam, and B. Reznick. "Real zeros of positive semidefinite forms. I." *Mathematische Zeitschrift*, **171**, 1980, pp. 1–26.

[161] J. S. Provan and M. O. Ball. "The complexity of counting cuts and of computing the probability that a graph is connected." *SIAM Journal on Computing*, **12**(4), 1983, pp. 777–788.

[162] M. E. Dyer and A. M. Frieze. "On the complexity of computing the volume of a polyhedron." *SIAM Journal on Computing*, **17**(5), 1988.

[163] E. D. Andersen and K. D. Andersen. "The MOSEK interior point optimizer for linear programming: an implementation of the homogeneous algorithm." In: *High Performance Optimization*. Springer, 2000, pp. 197–232.

[164] M. Petersen and R. Tedrake. "Growing convex collision-free regions in configuration space using nonlinear programming." *arXiv preprint arXiv:2303.14737*, 2023.

[165] J. Mattingley and S. Boyd. "CVXGEN: A code generator for embedded convex optimization." *Optimization and Engineering*, **13**, 2012, pp. 1–27.

[166] S. Redon, M. C. Lin, D. Manocha, and Y. J. Kim. "Fast Continuous Collision Detection for Articulated Models." *Journal of Computing and Information Science in Engineering*, **5**(2), 2005, pp. 126–137. DOI: 10.1115/1.1884133.

[167] T. Siméon, J.-P. Laumond, and C. Nissoux. "Visibility-based probabilistic roadmaps for motion planning." *Advanced Robotics*, **14**(6), 2000, pp. 477–493.

[168] T. Lozano-Pérez and M. A. Wesley. "An algorithm for planning collision-free paths among polyhedral obstacles." *Communications of the ACM*, **22**(10), 1979, pp. 560–570.

[169] M. Abrahamsen, W. Bille Meyling, and A. Nusser. "Constructing Concise Convex Covers via Clique Covers (CG Challenge)." In: *39th International Symposium on Computational Geometry (SoCG 2023)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2023.

[170] G. D. da Fonseca. "Shadoks approach to convex covering." *arXiv preprint arXiv:2303.07696*, 2023.

[171] A. Varava, J. F. Carvalho, D. Kragic, and F. T. Pokorny. "Free space of rigid objects: Caging, path non-existence, and narrow passage detection." *The International Journal of Robotics Research*, **40**(10-11), 2021, pp. 1049–1067.

[172] A. Sarmiento, R. Murrieta-Cid, and S. Hutchinson. "A Sample-based Convex Cover for Rapidly Finding an Object in a 3-D Environment." In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation (ICRA 2005)*. 2005, pp. 3486–3491. DOI: 10.1109/ROBOT.2005.1570649.

[173] D. Strash and L. Thompson. "Effective data reduction for the vertex clique cover problem." In: *2022 Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX)*. SIAM. 2022, pp. 41–53.

[174] S. M. LaValle. *Planning algorithms*. Cambridge University Press, 2006.

[175] J. J. Kuffner and S. M. LaValle. "RRT-connect: An efficient approach to single-query path planning." In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*. Vol. 2. IEEE. 2000, pp. 995–1001.

[176] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. Version 11.0. 2024. URL: https://www.gurobi.com.

[177] T. Marcucci, P. Nobel, R. Tedrake, and S. Boyd. "Fast Path Planning Through Large Collections of Safe Boxes." *arXiv preprint arXiv:2305.01072*, 2023.

[178] A. Majumdar and R. Tedrake. "Funnel libraries for real-time robust feedback motion planning." *The International Journal of Robotics Research*, **36**(8), 2017, pp. 947–982.

[179] A. A. Ahmadi, M. Krstic, and P. A. Parrilo. "A globally asymptotically stable polynomial vector field with no polynomial Lyapunov function." In: *2011 50th IEEE Conference on Decision and Control and European Control Conference*. 2011, pp. 7579–7580.

[180] J. Nie and M. Schweighofer. "On the complexity of Putinar's Positivstellensatz." en. *Journal of Complexity*, **23**(1), Feb. 2007, pp. 135–150. ISSN: 0885-064X. (Visited on 10/12/2022).

[181] L. Baldi and B. Mourrain. "On Moment Approximation and the Effective Putinar's Positivstellensatz." *arXiv preprint arXiv:2111.11258*, 2021.

[182] W. Rudin. *Principles of mathematical analysis*. Vol. 3. McGraw-hill New York, 1976.

[183] D. Cifuentes and P. A. Parrilo. "Sampling algebraic varieties for sum of squares programs." *SIAM Journal on Optimization*, **27**(4), 2017, pp. 2381–2404.

[184] B. Sturmfels. "What is ... a Gröbner basis?" *Notices of the American Mathematical Society*, **52**(10), 2005, p. 1199.

[185] D. Cifuentes and P. A. Parrilo. "Chordal networks of polynomial ideals." *SIAM Journal on Applied Algebra and Geometry*, **1**(1), 2017, pp. 73–110.

[186] F. Permenter and P. A. Parrilo. "Basis selection for SOS programs via facial reduction and polyhedral approximations." In: *53rd IEEE Conference on Decision and Control*. 2014, pp. 6615–6620.

[187] J. Briales and J. Gonzalez-Jimenez. "Convex global 3D registration with Lagrangian duality." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 4960–4969.

[188] J. Briales, L. Kneip, and J. Gonzalez-Jimenez. "A certifiably globally optimal solution to the non-minimal relative pose problem." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 145–154.

[189] J. P. Ruiz and I. E. Grossmann. "Using redundancy to strengthen the relaxation for the global optimization of MINLP problems." *Computers & Chemical Engineering*, **35**(12), 2011, pp. 2729–2740.

[190] F. Dümbgen, C. Holmes, B. Agro, and T. D. Barfoot. "Toward globally optimal state estimation using automatically tightened semidefinite relaxations." *IEEE Transactions on Robotics*, **40**, 2024, pp. 4338–4358. URL: https://doi.org/10.1109/TRO.2024.3454570.

[191] H. Yang and L. Carlone. "A quaternion-based certifiably optimal solution to the Wahba problem with outliers." In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 1665–1674.

[192] R. Tron, D. M. Rosen, and L. Carlone. "On the inclusion of determinant constraints in lagrangian duality for 3D SLAM." In: *Robotics: Science and Systems (RSS), Workshop on the Problem of Mobile Sensors: Setting Future Goals and Indicators of Progress for SLAM*. Vol. 4. 2015.

[193] B. Buchberger. "Bruno Buchberger's PhD thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal." *Journal of Symbolic Computation*, **41**(3-4), 2006, pp. 475–511.

[194] J.-C. Faugère. "A new efficient algorithm for computing Gröbner bases (F4)." *Journal of Pure and Applied Algebra*, **139**(1-3), 1999, pp. 61–88.

[195] J.-C. Faugère. "A new efficient algorithm for computing Gröbner bases without reduction to zero (F5)." In: *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation (ISSAC '02)*. New York, NY, USA: ACM, 2002, pp. 75–83.

[196] D. Eisenbud. *Commutative algebra: with a view toward algebraic geometry*. Springer Science & Business Media, 2013.

[197] R. Fröberg. "An inequality for Hilbert series of graded algebras." *Mathematica Scandinavica*, **56**(2), 1985, pp. 117–144.

[198] S. Feng, X. Xinjilefu, C. G. Atkeson, and J. Kim. "Optimization-Based Controller Design and Implementation for the Atlas Robot in the DARPA Robotics Challenge Finals." In: *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. 2015, pp. 1028–1035.

[199] S. Kuindersma, R. Deits, M. Fallon, A. Valenzuela, H. Dai, F. Permenter, T. Koolen, P. Marion, and R. Tedrake. "Optimization-based Locomotion Planning, Estimation, and Control Design for the Atlas Humanoid Robot." *Autonomous Robots*, **40**(3), 2016, pp. 429–455.

[200] S. Fahmi, C. Mastalli, M. Focchi, and C. Semini. "Passive Whole-Body Control for Quadruped Robots: Experimental Validation Over Challenging Terrain." *IEEE Robotics and Automation Letters*, **4**(3), 2019, pp. 2553–2560.

[201] S. Mason, N. Rotella, S. Schaal, and L. Righetti. "An MPC Walking Framework With External Contact Forces." In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 1785–1790. URL: https://arxiv.org/abs/1712.09308.

[202] K. Nguyen, S. Schoedel, A. Alavilli, B. Plancher, and Z. Manchester. "TinyMPC: Model-Predictive Control on Resource-Constrained Microcontrollers." *arXiv preprint arXiv:2310.16985*, 2023. URL: https://arxiv.org/abs/2310.16985.

[203] S. Boyd and B. Wegbreit. "Fast Computation of Optimal Contact Forces." *IEEE Transactions on Robotics*, **23**(6), 2007, pp. 1117–1132.

[204] T. Pang, H. J. T. Suh, L. Yang, and R. Tedrake. "Global planning for contact-rich manipulation via local smoothing of quasi-dynamic contact models." *IEEE Transactions on Robotics*, **39**(6), 2023, pp. 4691–4711.

[205] J. T. Betts. "Survey of numerical methods for trajectory optimization." *Journal of Guidance, Control, and Dynamics*, **21**(2), 1998, pp. 193–207.

[206] M. Toussaint. "A tutorial on Newton methods for constrained trajectory optimization and relations to SLAM, Gaussian Process smoothing, optimal control, and probabilistic inference." *Geometric and numerical foundations of movements*, 2017, pp. 361–392.

[207] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel. "Motion planning with sequential convex optimization and convex collision checking." *The International Journal of Robotics Research*, **33**(9), 2014, pp. 1251–1270.

[208] F. Dümbgen, C. Holmes, and T. D. Barfoot. "Exploiting Chordal Sparsity for Fast Global Optimality with Application to Localization." *arXiv preprint arXiv:2406.02365*, 2024. URL: https://arxiv.org/abs/2406.02365.

[209] A. L. Bishop, J. Z. Zhang, S. Gurumurthy, K. Tracy, and Z. Manchester. "ReLU-QP: A GPU-accelerated quadratic programming solver for model-predictive control." In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2024, pp. 13285–13292.

[210] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. "Conic Optimization via Operator Splitting and Homogeneous Self-Dual Embedding." *Journal of Optimization Theory and Applications*, **169**(3), June 2016, pp. 1042–1068. URL: http://stanford.edu/~boyd/papers/scs.html.

[211] Y. Ye, M. J. Todd, and S. Mizuno. "An $\mathcal{O}(\sqrt{n}L)$-iteration homogeneous and self-dual linear programming algorithm." *Mathematics of Operations Research*, **19**(1), 1994.

[212] R. M. Freund. "On the behavior of the homogeneous self-dual model for conic convex optimization." *Mathematical Programming*, **106**(3), 2006, pp. 527–545.

[213] F. Permenter, H. A. Friberg, and E. D. Andersen. "Solving conic optimization problems via self-dual embedding and facial reduction: a unified approach." *SIAM Journal on Optimization*, **27**(3), 2017, pp. 1257–1282.

[214] G. Banjac, P. Goulart, B. Stellato, and S. Boyd. "Infeasibility detection in the alternating direction method of multipliers for convex optimization." *Journal of Optimization Theory and Applications*, **183**, 2019, pp. 490–519.

[215] Google. *osqp-cpp: A C++ interface for the OSQP quadratic programming solver*. https://github.com/google/osqp-cpp. Accessed: 2026-04-11. 2026.

[216] S. Diamond and S. Boyd. "CVXPY: A Python-embedded modeling language for convex optimization." *Journal of Machine Learning Research*, **17**(83), 2016, pp. 1–5.

[217] M. Schubiger, G. Banjac, and J. Lygeros. "GPU acceleration of ADMM for large-scale quadratic programming." *Journal of Parallel and Distributed Computing*, **144**, 2020, pp. 55–67.

[218] B. Stellato, V. V. Naik, A. Bemporad, P. Goulart, and S. Boyd. "Embedded mixed-integer quadratic optimization using the OSQP solver." In: *2018 European Control Conference (ECC)*. IEEE. 2018, pp. 1536–1541.

[219] A. Bambade, F. Schramm, S. El-Kazdadi, S. Caron, A. Taylor, and J. Carpentier. "ProxQP: An Efficient and Versatile Quadratic Programming Solver for Real-Time Robotics Applications and Beyond." *IEEE Transactions on Robotics*, **40**, 2024, pp. 1617–1629.

[220] B. O'Donoghue. "Operator Splitting for a Homogeneous Embedding of the Linear Complementarity Problem." *SIAM Journal on Optimization*, **31**(3), Aug. 2021, pp. 1999–2023.

[221] Y. Ouyang, Y. Chen, G. Lan, and E. J. Pasiliao. "An accelerated linearized alternating direction method of multipliers." *SIAM Journal on Imaging Sciences*, **8**(1), 2015, pp. 644–681.

[222] S. Zhao, Z. Frangella, and M. Udell. "NysADMM: faster composite convex optimization via low-rank approximation." In: *International Conference on Machine Learning*. PMLR. 2022, pp. 26824–26840.

[223] J. Eckstein and W. Yao. "Relative-error approximate versions of Douglas–Rachford splitting and special cases of the ADMM." *Mathematical Programming*, **170**(2), 2018, pp. 417–444.

[224] T. Diamandis, Z. Frangella, S. Zhao, B. Stellato, and M. Udell. "GeNIOS: an (almost) second-order operator-splitting solver for large-scale convex optimization." *arXiv preprint arXiv:2310.08333*, 2023.

[225] A. Chambolle and T. Pock. "A first-order primal-dual algorithm for convex problems with applications to imaging." *Journal of Mathematical Imaging and Vision*, **40**(1), 2011, pp. 120–145.

[226] L. Condat. "A primal–dual splitting method for convex optimization involving Lipschitzian, proximable and linear composite terms." *Journal of Optimization Theory and Applications*, **158**(2), 2013, pp. 460–479.

[227] E. Esser, X. Zhang, and T. F. Chan. "A general framework for a class of first order primal-dual algorithms for convex optimization in imaging science." *SIAM Journal on Imaging Sciences*, **3**(4), 2010, pp. 1015–1046.

[228] M. Zhu and T. Chan. "An efficient primal-dual hybrid gradient algorithm for total variation image restoration." *UCLA CAM Report*, **34**(2), 2008.

[229] H. Lu and J. Yang. "cuPDLP.jl: A GPU Implementation of Restarted Primal-Dual Hybrid Gradient for Linear Programming in Julia." *Operations Research*, 2023. Forthcoming. arXiv preprint arXiv:2307.07585.

[230] H. Lu, Z. Peng, and J. Yang. "cuPDLPx: A further enhanced GPU-based first-order solver for linear programming." *arXiv preprint arXiv:2507.14051*, 2025.

[231] H. Lu, J. Yang, H. Hu, Q. Huangfu, J. Liu, T. Liu, Y. Ye, C. Zhang, and D. Ge. "cuPDLP-C: A strengthened implementation of cuPDLP for linear programming in C." *arXiv preprint arXiv:2312.14832*, 2023.

[232] Z. Lin, Z. Xiong, D. Ge, and Y. Ye. "PDCS: A primal-dual large-scale conic programming solver with GPU enhancements." *arXiv preprint arXiv:2505.00311*, 2025.

[233] P. Giselsson and S. Boyd. "Diagonal scaling in Douglas-Rachford splitting and ADMM." In: *53rd IEEE Conference on Decision and Control*. IEEE. 2014, pp. 5033–5039.

[234] J. Zhang, B. O'Donoghue, and S. Boyd. "Globally convergent type-I Anderson acceleration for nonsmooth fixed-point iterations." *SIAM Journal on Optimization*, **30**(4), 2020, pp. 3170–3197.

[235] M. Garstka, M. Cannon, and P. Goulart. "Safeguarded Anderson acceleration for parametric nonexpansive operators." In: *2022 European Control Conference (ECC)*. IEEE. 2022, pp. 435–440.

[236] J. Park and E. K. Ryu. "Accelerated infeasibility detection of constrained optimization and fixed-point iterations." In: *International Conference on Machine Learning*. PMLR. 2023, pp. 27294–27345.

[237] H. Lu, Z. Peng, and J. Yang. "MPAX: Mathematical Programming in JAX." *arXiv preprint arXiv:2412.09734*, 2024.

[238] K. Tracy and Z. Manchester. "On the differentiability of the primal-dual interior-point method." *arXiv preprint arXiv:2406.11749*, 2024.

[239] M. Blondel, Q. Berthet, M. Cuturi, R. Frostig, S. Hoyer, F. Llinares-López, F. Pedregosa, and J.-P. Vert. "Efficient and Modular Implicit Differentiation." In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 35. 2022, pp. 5230–5242.

[240] J. Nocedal and S. J. Wright. *Numerical Optimization*. 2nd ed. New York: Springer, 2006.

[241] M. Benzi, G. H. Golub, and J. Liesen. "Numerical solution of saddle point problems." *Acta Numerica*, **14**, 2005, pp. 1–137.

[242] R. J. Vanderbei. "Symmetric quasidefinite matrices." *SIAM Journal on Optimization*, **5**(1), 1995, pp. 100–113.

[243] J. H. Wilkinson. *Rounding errors in algebraic processes*. Courier Corporation, 1994.

[244] M. A. Saunders, H. D. Simon, and E. L. Yip. "Two Conjugate-Gradient-Type Methods for Unsymmetric Linear Equations." *SIAM Journal on Numerical Analysis*, **25**(4), 1988, pp. 927–940.

[245] M. R. Hestenes and E. Stiefel. "Methods of Conjugate Gradients for Solving Linear Systems." *Journal of Research of the National Bureau of Standards*, **49**(6), 1952, pp. 409–436.

[246] C. C. Paige and M. A. Saunders. "Solution of Sparse Indefinite Systems of Linear Equations." *SIAM Journal on Numerical Analysis*, **12**(4), 1975, pp. 617–629.

[247] J. D. Hogg, E. Ovtchinnikov, and J. A. Scott. "A sparse symmetric indefinite direct solver for GPU architectures." *ACM Transactions on Mathematical Software (TOMS)*, **42**(1), 2016, pp. 1–25.

[248] I. Duff, J. Hogg, and F. Lopez. "A new sparse LDL^T solver using a posteriori threshold pivoting." *SIAM Journal on Scientific Computing*, **42**(2), 2020, pp. C23–C42.

[249] V. Simoncini. "Computational methods for linear matrix equations." *SIAM Review*, **58**(3), 2016, pp. 377–441.

[250] Y. Voet. "Preconditioning techniques for generalized Sylvester matrix equations." *Numerical Linear Algebra with Applications*, **32**(2), 2025, e70020.

[251] J.-P. Chehab and M. Raydan. "An implicit preconditioning strategy for large-scale generalized Sylvester equations." *Applied Mathematics and Computation*, **217**(21), 2011, pp. 8793–8803.

[252] M. Hajarian. "Matrix form of the CGS method for solving general coupled matrix equations." *Applied Mathematics Letters*, **34**, 2014, pp. 37–42.

[253] T. Damm. "Direct methods and ADI-preconditioned Krylov subspace methods for generalized Lyapunov equations." *Numerical Linear Algebra with Applications*, **15**(9), 2008, pp. 853–871.

[254] D. Palitta, M. Iannacito, and V. Simoncini. "A subspace-conjugate gradient method for linear matrix equations." *SIAM Journal on Matrix Analysis and Applications*, **46**(4), 2025, pp. 2197–2225.

[255] A. Belloni and R. M. Freund. "On the second-order feasibility cone: Primal-dual representation and efficient projection." *SIAM Journal on Optimization*, **19**(3), 2008, pp. 1073–1092.

[256] D. Ruiz. *A scaling algorithm to equilibrate both rows and columns norms in matrices*. Tech. rep. CM-P00040415, 2001.

[257] B.-S. He, H. Yang, and S. Wang. "Alternating direction method with self-adaptive penalty parameters for monotone variational inequalities." *Journal of Optimization Theory and Applications*, **106**(2), 2000, pp. 337–356.

[258] B. Wohlberg. "ADMM penalty parameter selection by residual balancing." *arXiv preprint arXiv:1704.06209*, 2017.

[259] Z. Xiong and R. M. Freund. "Computational guarantees for restarted PDHG for LP based on "limiting error ratios" and LP sharpness." *arXiv preprint arXiv:2312.14774*, 2023.

[260] S. Kang and H. Yang. "Local Second-Order Limit Dynamics of the Alternating Direction Method of Multipliers for Semidefinite Programming." *arXiv preprint arXiv:2602.20103*, 2026.

[261] Y. E. Nesterov and M. J. Todd. "Self-scaled barriers and interior-point methods for convex programming." *Mathematics of Operations Research*, **22**(1), 1997, pp. 1–42.

[262] Y. E. Nesterov and M. J. Todd. "Primal-dual interior-point methods for self-scaled cones." *SIAM Journal on Optimization*, **8**(2), 1998, pp. 324–364.

[263] J. F. Sturm. "Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones." *Optimization Methods and Software*, **11**(1-4), 1999, pp. 625–653.

[264] R. Bhatia. "Positive definite matrices." In: *Positive Definite Matrices*. Princeton University Press, 2009.

[265] I. Maros and C. Mészáros. "A repository of convex quadratic programming problems." *Optimization Methods and Software*, **11**(1-4), 1999, pp. 671–681.

[266] G. Pataki and S. Schmieta. *The 7th DIMACS Implementation Challenge: Semidefinite and Related Optimization Problems*. http://dimacs.rutgers.edu/Challenges/Seventh/. 2000.

[267] H. D. Mittelmann. *Decision Tree for Optimization Software*. https://plato.asu.edu/guide.html. Accessed: 2026-04-08. 2024.

[268] H. Dai, A. Valenzuela, and R. Tedrake. "Whole-body motion planning with centroidal dynamics and full kinematics." In: *2014 IEEE-RAS International Conference on Humanoid Robots*. 2014, pp. 295–302.

[269] NVIDIA. *cuOpt: GPU-accelerated decision optimization*. Version as of access; open-source optimization engine for MILP, LP, and vehicle routing problems. 2025. URL: https://github.com/NVIDIA/cuopt.

[270] M. Schubiger. "GPU Acceleration of ADMM for Large-Scale Convex Optimization." MA thesis. ETH Zurich, Aug. 2019. DOI: 10.3929/ethz-b-000487703.

[271] B. Sundaralingam, S. K. S. Hari, A. Fishman, C. Garrett, K. Van Wyk, V. Blukis, A. Millane, H. Oleynikova, A. Handa, F. Ramos, et al. "cuRobo: Parallelized collision-free robot motion generation." In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 8112–8119.

[272] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter. "Differentiable convex optimization layers." *Advances in Neural Information Processing Systems*, **32**, 2019.

[273] NVIDIA Corporation. *NVIDIA CUDA C++ Programming Guide*. Version 12.4. NVIDIA Corporation. 2024. URL: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.

[274] NVIDIA Corporation. *NVIDIA CUDA C++ Best Practices Guide*. Version 13.2. NVIDIA Corporation. 2025. URL: https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html.

[275] G. H. Golub and C. F. Van Loan. *Matrix computations*. Johns Hopkins University Press, 2013.

[276] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1994.

[277] K. L. Ho and L. Greengard. "A fast direct solver for structured linear systems by recursive skeletonization." *SIAM Journal on Scientific Computing*, **34**(5), 2012, A2507–A2532.

[278] R. P. Brent. "4. Stability of Fast Algorithms for Structured Linear Systems." *Fast Reliable Algorithms for Matrices with Structure*, 1999, pp. 103–116.

[279] M. A. Epton. "Methods for the Solution of $AXD - BXC = E$ and its Application in the Numerical Solution of Implicit Ordinary Differential Equations." *BIT Numerical Mathematics*, **20**(3), 1980. Early application to implicit/differential-algebraic systems., pp. 341–345.

[280] L. Gao and V. M. Calo. "Preconditioners based on the alternating-direction-implicit algorithm for the 2D steady-state diffusion equation with orthotropic heterogeneous coefficients." *Journal of Computational and Applied Mathematics*, **273**, 2015, pp. 274–295.

[281] H. Ren and R. Dekany. "Fast wave-front reconstruction by solving the Sylvester equation with the alternating direction implicit method." *Optics Express*, **12**(14), 2004, pp. 3279–3296.

[282] J. D. Gardiner, M. R. Wette, A. J. Laub, J. J. Amato, and C. B. Moler. "Algorithm 705; a FORTRAN-77 software package for solving the Sylvester matrix equation $AXB^T + CXD^T = E$." *ACM Transactions on Mathematical Software (TOMS)*, **18**(2), 1992, pp. 232–238.

[283] R. H. Bartels and G. W. Stewart. "Algorithm 432 [C2]: Solution of the matrix equation $AX + XB = C$ [F4]." *Communications of the ACM*, **15**(9), 1972, pp. 820–826.

[284] G. Golub, S. Nash, and C. Van Loan. "A Hessenberg-Schur method for the problem $AX+XB = C$." *IEEE Transactions on Automatic Control*, **24**(6), 1979, pp. 909–913.

[285] K.-w. E. Chu. "The solution of the matrix equations $AXB-CXD = E$ AND $(YA-DZ, YC-BZ) = (E, F)$." *Linear Algebra and its Applications*, **93**, 1987, pp. 93–105.

[286] P. Benner, P. d. Boef, P. Kürschner, X. Liu, and J. Saak. "Reduced rank extrapolation for multi-term Sylvester equations." *arXiv preprint arXiv:2603.12979*, 2026.

[287] M. Iannacito, L. Piccinini, and V. Simoncini. "Subspace gradient descent method for linear tensor equations." *arXiv preprint arXiv:2602.21974*, 2026.

[288] F. A. Potra and S. J. Wright. "Interior-point methods." *Journal of Computational and Applied Mathematics*, **124**(1-2), 2000, pp. 281–302.

[289] A. Forsgren, P. E. Gill, and M. H. Wright. "Interior methods for nonlinear optimization." *SIAM Review*, **44**(4), 2002, pp. 525–597.

[290] B. Kågström. "A perturbation analysis of the generalized Sylvester equation $(AR - LB, DR - LE) = (C, F)$." *SIAM Journal on Matrix Analysis and Applications*, **15**(4), 1994, pp. 1045–1060.

[291] B. Kågström and P. Poromaa. "LAPACK-style algorithms and software for solving the generalized Sylvester equation and estimating the separation between regular matrix pairs." *ACM Transactions on Mathematical Software (TOMS)*, **22**(1), 1996, pp. 78–103.

[292] J. D. Gardiner, A. J. Laub, J. J. Amato, and C. B. Moler. "Solution of the Sylvester matrix equation $AXB^T+CXD^T = E$." *ACM Transactions on Mathematical Software (TOMS)*, **18**(2), 1992, pp. 223–231.

[293] C. B. Moler and G. W. Stewart. "An algorithm for generalized matrix eigenvalue problems." *SIAM Journal on Numerical Analysis*, **10**(2), 1973, pp. 241–256.

[294] T. Hopkins. "Remark on Algorithm 705: A FORTRAN-77 software package for solving the Sylvester matrix equation $AXB^T + CXD^T = E$." *ACM Transactions on Mathematical Software (TOMS)*, **28**(3), 2002, pp. 372–375.

[295] Y. P. Hong, R. A. Horn, and C. R. Johnson. "On the reduction of pairs of Hermitian or symmetric matrices to diagonal form by congruence." *Linear Algebra and its Applications*, **73**, 1986, pp. 213–226.

[296] K. Weierstrass. "Zur Theorie der quadratischen und bilinearen Formen." *Monatsber. Akad. Wiss., Berlin*, 1868.

[297] M. J. Todd, K.-C. Toh, and R. H. Tütüncü. "On the Nesterov–Todd direction in semidefinite programming." *SIAM Journal on Optimization*, **8**(3), 1998, pp. 769–796.

[298] B. Iannazzo. "The geometric mean of two matrices from a computational viewpoint." *Numerical Linear Algebra with Applications*, **23**(2), 2016, pp. 208–229.

[299] L. Vandenberghe and M. S. Andersen. "Chordal Graphs and Semidefinite Optimization." en. *Foundations and Trends in Optimization*, **1**(4), 2015, pp. 241–433. ISSN: 2167-3888, 2167-3918.

[300] L. Lessard and S. Lall. "An Algebraic Approach to the Control of Decentralized Systems." en. *IEEE Transactions on Control of Network Systems*, **1**(4), Dec. 2014, pp. 308–317. ISSN: 2325-5870, 2372-2533.

[301] J. Anderson, J. C. Doyle, S. H. Low, and N. Matni. "System level synthesis." *Annual Reviews in Control*, **47**, 2019, pp. 364–393.

[302] P. Shah and P. A. Parrilo. "$\mathcal{H}_2$-Optimal Decentralized Control over Posets: A State-Space Solution for State-Feedback." *IEEE Transactions on Automatic Control*, **58**(12), Dec. 2013. arXiv: 1111.1498, pp. 3084–3096.

[303] Y.-C. Ho and K. Chu. "Team decision theory and information structures in optimal control problems–Part I." *IEEE Transactions on Automatic Control*, **17**(1), Feb. 1972, pp. 15–22. ISSN: 1558-2523.

[304] L. Lessard and S. Lall. "Optimal Control of Two-Player Systems With Output Feedback." *IEEE Transactions on Automatic Control*, **60**(8), Aug. 2015, pp. 2129–2144. ISSN: 1558-2523.

[305] T. Tanaka and P. A. Parrilo. "Optimal output feedback architecture for triangular LQG problems." In: *2014 American Control Conference*. IEEE. 2014, pp. 5730–5735.

[306] R. P. Stanley. *Enumerative Combinatorics*. 2nd ed. Vol. 1. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2011.

[307] A. Nayyar and L. Lessard. "Structural Results for Partially Nested LQG Systems over Graphs." en. In: *2015 American Control Conference (ACC)*. Chicago, IL, USA, 2015, pp. 5457–5464. ISBN: 978-1-4799-8684-2.

[308] D. E. Crabtree and E. V. Haynsworth. "An identity for the Schur complement of a matrix." *Proceedings of the American Mathematical Society*, **22**(2), 1969, pp. 364–366.

[309] R. E. Tarjan. "Graph theory and Gaussian elimination." *Sparse Matrix Computations*, 1976, pp. 3–22.

[310] T. F. Coleman, A. Edenbrandt, and J. R. Gilbert. "Predicting fill for sparse orthogonal factorization." *Journal of the ACM (JACM)*, **33**(3), 1986, pp. 517–532.

[311] T. A. Davis. *Direct methods for sparse linear systems*. SIAM, 2006.

[312] ARM Institute. *Project Highlight: Time-Optimal Motion Planning Using Convex Sets*. https://arminstitute.org/news/motion-planning-convex-sets/. Accessed: 2026-04-14. 2024.

[313] T. Marcucci, M. Halm, W. Yang, D. Lee, and A. D. Marchese. "A biconvex method for minimum-time motion planning through sequences of convex sets." *arXiv preprint arXiv:2504.18978*, 2025.

[314] I. Dunning, J. Huchette, and M. Lubin. "JuMP: A modeling language for mathematical optimization." *SIAM Review*, **59**(2), 2017, pp. 295–320.

[315] M. Lubin, O. Dowson, J. D. Garcia, J. Huchette, B. Legat, and J. P. Vielma. "JuMP 1.0: recent improvements to a modeling language for mathematical optimization." *Mathematical Programming Computation*, **15**(4), 2023, pp. 581–589.

[316] H. A. Friberg. "CBLIB 2014: a benchmark library for conic mixed-integer and continuous optimization." *Mathematical Programming Computation*, **8**(2), 2016, pp. 191–214.

[317] K. Fujisawa, M. Kojima, K. Nakata, and M. Yamashita. *SDPA (Semidefinite Programming Algorithm) User's Manual*. Research Report B-308. Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 1995.

[318] IBM Corporation. *Mathematical Programming System Extended (MPSX) and Generalized Information System (GIS) Usage Manual*. Document Number SH20-0960-0. White Plains, NY, 1971.

[319] P. Trutman, S. E. D. Mohab, D. Henrion, and T. Pajdla. "Globally Optimal Solution to Inverse Kinematics of 7DOF Serial Manipulator." *arXiv preprint arXiv:2007.12550*, 2020.

[320] B. Sturmfels. "On the Newton polytope of the resultant." *Journal of Algebraic Combinatorics*, **3**(2), 1994, pp. 207–236.

Appendix A

Appendices for Chapter 5

A.1 Algebraic Kinematics

An in-depth review of algebraic kinematics and low order pairs can be found in [114, Chapter 4]. We include a brief review in this appendix for completeness.

A mechanism composed of $N+1$ links is considered algebraic if each link is connected by one of the following five joints:

- Revolute (R): a 1-DOF joint permitting revolution about an axis of symmetry. An example is a door handle.
- Prismatic (P): a 1-DOF joint permitting translation along an axis. An example is a linear rail.
- Cylindrical (C): a 2-DOF joint permitting both revolution about an axis of symmetry and independent translation along a given axis. An example is the rods of a Foosball table.
- Planar (E): A 3-DOF joint permitting translation and rotation in a two-dimensional plane. An example is a hockey puck moving on the surface of the ice.
- Spherical (S): A 3-DOF joint permitting free rotation between two links. An example is the human shoulder.

We recall from Section 5.3.2 that the pose of a point A expressed in the reference frame F, written as a function of the robot configuration q can be expressed as

$$\begin{bmatrix} {}^{F}R^{A}(q) & {}^{F}p^{A}(q) \\ 0_{1\times 3} & 1 \end{bmatrix} = \prod_{i \in \mathcal{J}_{F,A}} {}^{P_i}X^{C_i}(q_i)\ {}^{C_i}X^{P_{i+1}} \tag{A.1}$$

where ${}^{P_i}X^{C_i}(q_i)$ is a rigid transform describing the relative motion allowed by the i^{th} joint.

The matrices ${}^{P_i}X^{C_i}(q_i)$ are in general restrictions of the following forms

$$ {}^{P_i}X^{C_i}(q_i) = \begin{cases} \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & x_i \\ \sin(\theta_i) & \cos(\theta_i) & 0 & y_i \\ 0 & 0 & 1 & z_i \\ 0 & 0 & 0 & 1 \end{bmatrix} & \text{if } i^{\text{th}} \text{ joint is one of R, P, C, or E} \\ \begin{bmatrix} U(\psi_i) & 0_{3\times 1} \\ 0_{1\times 3} & 1 \end{bmatrix} & \text{if } i^{\text{th}} \text{ joint is S} \end{cases} \tag{A.2} $$

The specific restrictions for R, P, C, and E joints are given in Table A.1. The matrix U is an element of $SO(3)$ parametrized using Euler angles $\{\phi_{i,x}, \phi_{i,y}, \phi_{i,z}\}$.

Joint	Restriction	Definition of q_i
R	$x_i = y_i = z_i = 0$	$q_i = \{\theta_i\}$
P	$\theta_i = x_i = y_i = 0$	$q_i = \{z_i\}$
C	$x_i = y_i = 0$	$q_i = \{\theta_i, z_i\}$
E	$z_i = 0$	$q_i = \{\theta_i, x_i, y_i\}$
S	see equation (A.5)	$q_i = \{\phi_{i,x}, \phi_{i,y}, \phi_{i,z}\}$

Table A.1: Parameterization of algebraic joints in terms of the matrix given in (A.2).

We remark that the joints C, E, and S can be constructed by the composition of R and P joints.

- A C joint is a composition of an R joint and a P joint:

$$ \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & 0 \\ \sin(\theta_i) & \cos(\theta_i) & 0 & 0 \\ 0 & 0 & 1 & z_i \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & 0 \\ \sin(\theta_i) & \cos(\theta_i) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & z_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.3} $$

- An E joint is the composition of one R joint and two P joints

$$ \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & x_i \\ \sin(\theta_i) & \cos(\theta_i) & 0 & y_i \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & x_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & y_i \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & 0 \\ \sin(\theta_i) & \cos(\theta_i) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.4} $$

- An S joint is the composition of three R joints expressed as Euler angles.

$$ U(\psi_i) = \begin{bmatrix} \cos(\psi_{i,x}) & -\sin(\psi_{i,x}) & 0 \\ \sin(\psi_{i,x}) & \cos(\psi_{i,x}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\psi_{i,y}) & 0 & -\sin(\psi_{i,y}) \\ 0 & 1 & 0 \\ \sin(\psi_{i,y}) & 0 & \cos(\psi_{i,y}) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\psi_{i,z}) & -\sin(\psi_{i,z}) \\ 0 & \sin(\psi_{i,z}) & \cos(\psi_{i,z}) \end{bmatrix} \tag{A.5} $$

  Our approach, presented for a robot composed of R and P joints, can be extended to handle any algebraic mechanism by considering the other algebraic joints as compositions of R and P joints.

A.2 Semialgebraic Descriptions of Set Membership for Common Convex Bodies

A.3 Description of Set Membership for Common Convex Bodies

We collect the common convex bodies used to represent shapes in robot simulators and how to express that a point lies in that body.

Body	Psatz Condition for (5.18c)
V-rep Polytope with m vertices v_i at position ${}^F p^{v_i}(s) = \frac{{}^F f^{v_i}(s)}{{}^F g^{v_i}(s)}$	Enforce (5.14) for each vertex v_i.
Sphere with center o at position ${}^F p^{o}(s) = \frac{{}^F f^{o}(s)}{{}^F g^{o}(s)}$ and radius r	Enforce (5.17) for the center o with radius r. Also enforce (5.14) for the center o.
Capsule, the convex hull of two spheres with centers o_1 and o_2 at positions ${}^F p^{o_i}(s) = \frac{{}^F f^{o_i}(s)}{{}^F g^{o_i}(s)}$ and radii r_1, r_2	For $i \in \{1, 2\}$ enforce (5.17) for center o_i with radius r_i. Also enforce (5.14) for o_i.
Cylinder, the convex hull of two circles with centers o_1 and o_2, at position ${}^F p^{o_i}(s) = \frac{{}^F f^{o_i}(s)}{{}^F g^{o_i}(s)}$, lying in the plane normal to ${}^F p^{o_1}(s) - {}^F p^{o_2}(s)$, and with radii r_1 and r_2.	See Appendix A.4.

Table A.2: SOS conditions for the constraint (5.18c) and (5.18d) depending on the geometry of bodies $\mathcal{A}$ and $\mathcal{B}$.

A.3.1 Semialgebraic Description under Rational Forward Kinematics

Body	Variables	Description of $\mathcal{A}(s)$ as a semi-algebraic set
V-rep Polytope with m vertices v_i at position ${}^F p^{v_i}(s) = \frac{{}^F f^{v_i}(s)}{{}^F g^{v_i}(s)}$	$\{s, x, \mu\}$	$h_1(s,x,\mu) = \left(\prod_i {}^F g^{v_i}\right)\left(x - \sum_{i=1}^m \mu_i \left(\frac{{}^F f^{v_i}(s)}{{}^F g^{v_i}(s)}\right)\right)$ $h_2(\mu) = 1 - \sum_{i=1} \mu_i$ $\gamma_i(\mu_i) = \mu_i,\ i \in [m]$
Sphere with center o at position ${}^F p^o(s) = \frac{{}^F f^o(s)}{{}^F g^o(s)}$ and radius r	$\{s, x\}$	$\gamma_1(s,x) = \left({}^F g^o(s)\right)^2 \left(r^2 - \left\|x - \frac{{}^F f^o(s)}{{}^F g^o(s)}\right\|^2\right)$
Capsule, the convex hull of two spheres with centers c_1 and c_2 at positions ${}^F p^{o_i}(s) = \frac{{}^F f^{o_i}(s)}{{}^F g^{o_i}(s)}$ and radii r_1, r_2	$\{s, x, \mu\}$	$\frac{{}^F f^{o_\mu}}{{}^F g^{o_\mu}} = \mu \frac{{}^F f^{o_1}(s)}{{}^F g^{o_1}(s)} + (1-\mu)\frac{{}^F f^{o_2}(s)}{{}^F g^{o_2}(s)}$ $r_\mu = \mu r_1 + (1-\mu) r_2$ $\gamma_1(s,x,\mu) = \left({}^F g^{o_\mu}(s)\right)^2 \left(r_\mu^2 - \left\|x - \frac{{}^F f^{o_\mu}}{{}^F g^{o_\mu}}\right\|^2\right)$ $\gamma_2(\mu) = \mu$ $\gamma_3(\mu) = 1 - \mu$
Cylinder, the convex hull of two circles with centers o_1 and o_2, at position ${}^F p^{o_i}(s) = \frac{{}^F f^{o_i}(s)}{{}^F g^{o_i}(s)}$, lying in the plane normal to ${}^F p^{o_1}(s) - {}^F p^{o_2}(s)$, and with radii r_1 and r_2.	$\{s, x, v, \mu\}$	$\frac{{}^F f^{o_\mu}(s)}{{}^F g^{o_\mu}(s)} = \mu \frac{{}^F f^{o_1}(s)}{{}^F g^{o_1}(s)} + (1-\mu)\frac{{}^F f^{o_2}(s)}{{}^F g^{o_2}(s)}$ $r_\mu = \mu r_1 + (1-\mu) r_2$ $h_1(v,s) = v^T \left(\frac{{}^F f^{o_1}(s)}{{}^F g^{o_1}(s)} - \frac{{}^F f^{o_2}(s)}{{}^F g^{o_2}(s)}\right)$ $h_2(s,x,\mu,v) = x - \frac{{}^F f^{o_\mu}(s)}{{}^F g^{o_\mu}(s)} - v$ $\gamma_1(v,\mu) = r_\mu^2 - v^T v$ $\gamma_2(\mu) = \mu$ $\gamma_3(\mu) = 1 - \mu$

Table A.3: Parameterizations of the condition that x lies in a convex body that moves rigidly as a function of s.

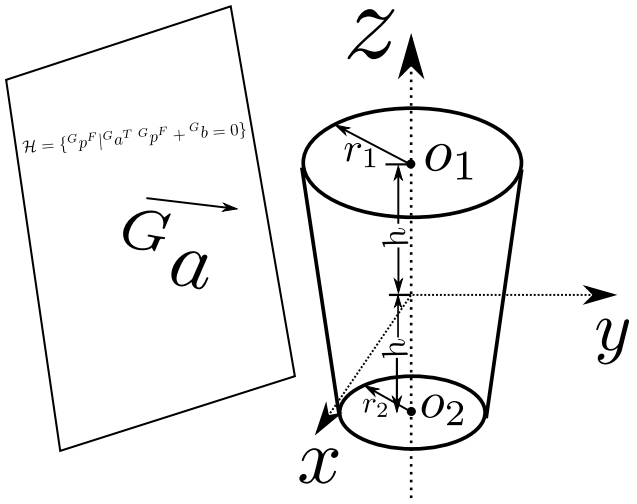


Figure A.1: Illustration of the cylinder on one side of the plane $\mathcal{H}$, with the plane normal being ${}^G a$, expressed in the cylinder's geometry frame G.

A.4 Parametrized Hyperplane Separation Condition for the Cylinder

To derive the hyperplane-separation condition for the cylinder, we first attach a geometric frame G to the cylinder, as shown in Fig. A.1. The cylinder's geometry frame G's origin coincides with the cylinder's center, with the z axis of the G frame along the cylinder axis. The height of the cylinder is $2h$, with the top/bottom circle radii being r_1 and r_2, respectively.

We first write the plane $\mathcal{H}$ with its parameters ${}^G a(s), {}^G b(s)$ in the cylinder's geometric frame G and derive the conditions on ${}^G a(s), {}^G b(s)$. The cylinder is on the positive side of the plane if and only if both its top and bottom rims are on the positive side of the plane, namely

$$\left({}^G a(s)\right)^T \begin{bmatrix} r_1 \cos\alpha \\ r_1 \sin\alpha \\ h \end{bmatrix} + {}^G b(s) \geq 0 \; \forall \alpha \tag{A.6a}$$

$$\left({}^G a(s)\right)^T \begin{bmatrix} r_2 \cos\alpha \\ r_2 \sin\alpha \\ -h \end{bmatrix} + {}^G b(s) \geq 0 \; \forall \alpha. \tag{A.6b}$$

Taking the infimum of both sides with respect to α makes the above conditions equivalent to

$${}^G a_z(s) h + {}^G b(s) \geq r_1 \left\| \begin{bmatrix} {}^G a_x(s) & {}^G a_y(s) \end{bmatrix} \right\| \tag{A.7a}$$

$$-{}^G a_z(s) h + {}^G b(s) \geq r_2 \left\| \begin{bmatrix} {}^G a_x(s) & {}^G a_y(s) \end{bmatrix} \right\| \tag{A.7b}$$

Next, we use the Schur complement to reformulate (A.7a) and (A.7b) as positive semidefinite matrix conditions. For example, (A.7a) is equivalent to

$$\begin{bmatrix} {}^{G}a_z(s)h + {}^{G}b(s) & 0 & r_1\ {}^{G}a_x(s) \\ 0 & {}^{G}a_z(s)h + {}^{G}b(s) & r_1\ {}^{G}a_y(s) \\ r_1\ {}^{G}a_x(s) & r_1\ {}^{G}a_y(s) & {}^{G}a_z(s)h + {}^{G}b(s) \end{bmatrix} \succeq 0 \tag{A.8a}$$

As explained in Section 5.4.1, this polynomial PSD condition can be reformulated as the condition

$$u^T \begin{bmatrix} {}^{G}a_z(s)h + {}^{G}b(s) & 0 & r_1\ {}^{G}a_x(s) \\ 0 & {}^{G}a_z(s)h + {}^{G}b(s) & r_1\ {}^{G}a_y(s) \\ r_1\ {}^{G}a_x(s) & r_1\ {}^{G}a_y(s) & {}^{G}a_z(s)h + {}^{G}b(s) \end{bmatrix} u \geq 0\ \forall u. \tag{A.8b}$$

To avoid the trivial solution ${}^{G}a(s) = 0, {}^{G}b(s) = 0$ (which is not in fact a separating plane), we add the extra constraint ${}^{G}a^T \left(\frac{{}^{G}p^{o_1} + {}^{G}p^{o_2}}{2} \right) + {}^{G}b \geq 1$. Here $\frac{{}^{G}p^{o_1} + {}^{G}p^{o_2}}{2}$ is the position of the cylinder center expressed in the geometric frame G which coincides with the frame's origin. Therefore $\frac{{}^{G}p^{o_1} + {}^{G}p^{o_2}}{2} = 0$, and so it is sufficient to introduce the constraint

$${}^{G}b(s) \geq 1 \tag{A.9}$$

to exclude the trivial solution ${}^{G}a = 0, {}^{G}b = 0$.

In our optimization program, we express the separating plane in a frame F (where the choice of frame F is discussed in A.5.1), not in the cylinder's geometric frame G. Hence, we need to compute ${}^{G}a(s), {}^{G}b(s)$ from their corresponding terms ${}^{F}a(s), {}^{F}b(s)$ expressed in frame F and the relative transform ${}^{F}X^{G}$ between the two frames

$${}^{G}a(s) = {}^{G}R^{F}(s)\ {}^{F}a(s) \tag{A.10a}$$

$${}^{G}b(s) = {}^{F}b(s) + \left({}^{F}a(s)\right)^T\ {}^{F}p^{G}(s) \tag{A.10b}$$

As described in Section 5.3.2, both the position ${}^{F}p^{G}(s)$ and orientation ${}^{G}R^{F}(s)$ are rational functions of s. By replacing ${}^{G}a(s), {}^{G}b(s)$ in (A.8b) and (A.9) with (A.10) and requiring the resulting numerator of the rational function to be non-negative, we derive that the plane separating cylinders can be enforced via a polynomial non-negativity condition which can be formulated as a sums-of-squares condition.

A.5 Practical aspects

In this section, we discuss some practical aspects essential for applying Algorithm 3 to realistic examples. These include the choice of reference frame in which to express the forward kinematics, the selection of a finite basis for the polynomials in our SOS programs, and which aspects of 3 can be parallelized.

A.5.1 Choosing the Reference Frame

The polynomial implications upon which the certification program (5.18) and polytope growth program (5.33) are based require choosing a coordinate frame between each collision pair $\mathcal{A}$ and $\mathcal{B}$. However, as the collision-free certificate between two different collision pairs can be computed independently of each other, we are free to choose a different coordinate frame to express the kinematics for each collision pair. This is important in light of (5.5) and (5.9) that indicate that the degree of the polynomials ${}^{F}f^{\mathcal{A}_j}$ and ${}^{F}g^{\mathcal{A}_j}$ are equal to two times the number of joints lying on the kinematic chain between frame F and the frame for $\mathcal{A}$. For example, the tangent-configuration-space polynomial in the variable s describing the position of the end-effector of a 7-DOF robot is of total degree 14 when written in the coordinate frame of the robot base. However, when written in the frame of the third link, the polynomial describing the position of the end effector is only of total degree $(7-3)\times 2 = 8$. This observation is also used in [319] to reduce the size of the optimization program.

The size of the semidefinite variables in (5.18) and (5.33) scales as the square of the degree of the polynomial used to express the forward kinematics. Supposing there are n links in the kinematic chain between $\mathcal{A}$ and $\mathcal{B}$, then choosing the jth link along the kinematic chain as the reference frame F leads to scaling of order $j^2 + (n-j)^2$. Choosing the reference frame in the middle of the chain minimizes this complexity to scaling of order $\frac{n^2}{2}$ and we therefore adopt this convention in our experiments.

A.5.2 Basis Selection

The condition that a polynomial can be written as a sum of squares can be equivalently formulated as an equality constraint between the coefficients of the polynomial and an associated semidefinite variable known as the Gram matrix [158]. Namely, a polynomial $p(s)$ is sums-of-squares if and only if $p(s) = z(s)^T X z(s), X \succeq 0$ where $z(s)$ is a vector of monomials and X is the Gram matrix. The number of rows in the positive semidefinite Gram matrix equals the size of the vector $z(s)$. In general, a sums-of-squares polynomial in k variables of total degree $2d$ requires a Gram matrix of size $\binom{k+d}{d}$ to represent it, which can quickly become prohibitively large. Fortunately, the polynomials in our programs contain substantially more structure which will allow us to select a small-sized vector of monomials $z(s)$, and hence drastically reduce the size of the Gram matrices and speed up the optimization problem.

A.5.2.1 Polytopic collision geometry

We begin with the separating plane condition for polytopic collision geometries. Note that from (5.9) that while both the numerator and denominator of the forward kinematics are of total degree $2n$, with n the number of links of the kinematics chain between frame A and F, both polynomials are of *coordinate* degree of at most two (i.e. the highest degree of s_i in any term is s_i^2). We will refer to this basis as $\nu(s)$ which is a vector containing terms of the form $\prod_{i=1}^{n} s_i^{\text{degree}(s_i)}$ with $\text{degree}(s_i) \in \{0,1,2\}$ for all 3^n possible permutations of the exponents $\text{degree}(s_i)$.

We recall that we parametrize our hyperplane using polynomial entries. If $a_{\mathcal{A},\mathcal{B}}(s) =$

$a_{\mathcal{A},\mathcal{B}}^T \eta(s)$, $b_{\mathcal{A},\mathcal{B}}(s) = b_{\mathcal{A},\mathcal{B}}^T \eta(s)$ for some basis η in the variable s, and the position of $x(s) \in \mathcal{A}(s)$ is expressed in basis $\nu(s)$, then the left hand side of (5.14) can be expressed as a linear function of the basis $\gamma(s)$, where $\gamma(s)$ contains all the possible entries that appear in the outer product $\eta(s)\nu(s)^T$.

Example 19. *Suppose*

$$\eta(s) = \begin{bmatrix} 1 & s_1 & s_2 \end{bmatrix}^T$$

and

$$\nu(s) = \begin{bmatrix} 1 & s_1 & s_1^2 & s_2 & s_2^2 & s_1 s_2 & s_1^2 s_2 & s_1 s_2^2 & s_1^2 s_2^2 \end{bmatrix}^T$$

.

Then:

$$\gamma(s) = \begin{bmatrix} 1 & s_1 & s_1^2 & s_1^3 & s_2 & s_2^2 & s_2^3 & s_1 s_2 & s_1^2 s_2 & s_1^3 s_2 & s_1 s_2^2 & s_1^2 s_2^2 & s_1^3 s_2^2 & s_1 s_2^3 & s_1^2 s_2^3 \end{bmatrix}$$

Namely $\gamma(s)$ contains the monomials whose degree for each s_i is at most 3, and only one of s_i can have degree 3 (hence $s_1^3 s_2^3$ is not included in $\gamma(s)$).

Similarly, we must select a basis $\rho(s)$ for our multiplier polynomials $\lambda_{ij}^{\mathcal{A},\mathcal{B}}(s)$. The equality in (5.14) determines the minimum necessary basis $\rho(s)$. If the polynomial $p(s)$ is expressed in basis $\gamma(s)$, then the minimal such basis is related to an object known in computational algebra as the Newton polytope of γ denoted $\mathbf{New}(\gamma(s))$ [320]. Denoting the linear basis

$$l(s) = \begin{bmatrix} 1 & s_1 & s_2 & \dots & s_N \end{bmatrix},$$

the exact condition is that

$$\mathbf{New}(\gamma(s)) = \mathbf{New}(\eta(s)) + \mathbf{New}(\nu(s)) \subseteq \mathbf{New}(\rho(s)) + \mathbf{New}(l(s))$$

where the sum in this case is the Minkowski sum.

By using affine polynomials for separating plane parameters $a_{\mathcal{A},\mathcal{B}}(s), b_{\mathcal{A},\mathcal{B}}(s)$, we know that $\eta(s)$ is the same as the linear basis $l(s)$, then we obtain the condition that $\mathbf{New}(\rho(s)) = \mathbf{New}(\nu(s))$ and since $\nu(s)$ is a dense, even degree basis we must take $\rho(s) = \nu(s)$. A sums-of-squares polynomial in the basis of $\nu(s)$ has Gram matrix with 2^n rows. Choosing $\eta(s)$ as the constant basis would in fact result in the same condition, and therefore searching for separating planes which are linear functions of the tangent-configuration-space variable does not increase the size of the semidefinite variables. As the complexity of (5.18) and (5.33) are dominated by the size of these semidefinite variables, separating planes which are linear functions do not substantially affect the solve time but can dramatically increase the size of the regions which we can certify.

Because of this, we choose to parametrize all of our hyperplanes throughout our experiments as linear functions of the TC-space variables. We stress that in general, the choice of a linearly parametrized hyperplane, and the selection of $\rho(s)$ to be the minimum size to match the degree of the left hand side of (5.14) may not be sufficient to prove that a region $\mathcal{P}$ is collision-free, even if $\mathcal{P}$ truly is collision-free. Indeed due to many complexity-theoretic results, we expect that in general $\eta(s)$ and $\rho(s)$ may need to have exponentially high degree for some robots, scenes, and polytopes $\mathcal{P}$ [75]. However, in practice we have observed that the choices in this section are sufficient to certify many regions of interest, while keeping the optimization problem size tractable for state-of-the-art numerical solvers.

Remark 15. *Attempting to certify that the end-effector of a 7-DOF robot will not collide with the base using program (5.18) using linearly parametrized hyperplanes and choosing to express conditions (5.14) in the world frame with naively chosen bases would result in semidefinite variables of size* $\binom{7+7}{7} = 3432$. *Choosing to express the same conditions according to the discussion in Section A.5.1 and choosing the basis* $\gamma(s)$ *described in this section results in semidefinite matrices of rows at most* $2^{\lceil 7/2 \rceil} = 2^4 = 16$. *The division by 2 comes from choosing the middle link as the expressed frame, hence halving the kinematic chain length.*

A.5.2.2 Non-polytopic collision geometry

In this section, we use the sphere as a running example for explaining how we choose the monomial bases for certifying separation of the non-polytopic geometries; the monomial bases for capsules and cylinders can be derived in a similar manner.

As mentioned in (5.15), we need to impose

$$s \in \mathcal{P} \implies \begin{bmatrix} \left((a(s))^T \ {}^F f^o(s) + b(s) \ {}^F g^o(s)\right) I_3 & ra(s) \ {}^F g^o(s) \\ r(a(s))^T \ {}^F g^o(s) & (a(s))^T \ {}^F f^o(s) + b(s) \ {}^F g^o(s) \end{bmatrix} \succeq 0. \quad \text{(A.11)}$$

By the definition of positive semidefinite matrix [1] , we know that the 4×4 matrix on the right of $\implies$ in (A.11) is positive semidefinite if and only if

$$\forall \bar{u} \in \mathbb{R}^3, \underbrace{\begin{bmatrix} \bar{u} \\ 1 \end{bmatrix}^T \begin{bmatrix} \left((a(s))^T \ {}^F f^o(s) + b(s) \ {}^F g^o(s)\right) I_3 & ra(s) \ {}^F g^o(s) \\ r(a(s))^T \ {}^F g^o(s) & (a(s))^T \ {}^F f^o(s) + b(s) \ {}^F g^o(s) \end{bmatrix} \begin{bmatrix} \bar{u} \\ 1 \end{bmatrix}}_{\sigma(\bar{u}, s)} \geq 0. \quad \text{(A.12)}$$

We impose the following sufficient condition for (A.11), where $\mathcal{P} = \{s | c_j^T(s) \leq d_j, j = 1, \ldots, m\}$

$$\sigma(\bar{u}, s) = \lambda_0(\bar{u}, s) + \sum_{j=1}^{m} \lambda_j(\bar{u}, s)(d_j - c_j^T s) \quad \text{(A.13a)}$$

$$\text{for } j = 0, \ldots, m, \lambda_j(\bar{u}, s) \geq 0 \ \forall \bar{u}, s. \quad \text{(A.13b)}$$

Now we analyze the degree of the polynomial $\sigma(\bar{u}, s)$ defined in (A.12). As mentioned in the previous subsection, each monomial in ${}^F f^o(s), {}^F g^o(s)$ is of the form $\prod_{i=1}^n s_i^{\text{degree}(s_i)}$, $\text{degree}(s_i) \in \{0, 1, 2\}$. Combining this with the choice of a separating plane $a(s), b(s)$ being affine functions of s, we derive that each monomial in $\sigma(\bar{u}, s)$ is of the form $\bar{u}_j^{\text{degree}(\bar{u}_j)} \prod_{i=1}^n s_i^{\text{degree}(s_i)}$, where $\text{degree}(\bar{u}_j) \in \{0, 1, 2\}$, $\text{degree}(s_i) \in \{0, 1, 2, 3\}$, and at most one of $\text{degree}(s_i)$ can be 3. As an example, $\bar{u}_1^2 s_1^3 s_2 s_3^2$ is a valid monomial in $\sigma(\bar{u}, s)$ but $\bar{u}_1 \bar{u}_2$ is not (because $\sigma(\bar{u}, s)$ doesn't contain the cross product between $\bar{u}_j, \bar{u}_k, j \neq k$). Similarly, $s_1^3 s_2^3$ is not in the basis because at most one of s_i can have degree 3. Given these properties on the monomials in $\sigma(\bar{u}, s)$, specifically there

[1] A matrix X is positive semidefinite if and only if $\forall \bar{u}, \begin{bmatrix} \bar{u} \\ 1 \end{bmatrix}^T X \begin{bmatrix} \bar{u} \\ 1 \end{bmatrix} \geq 0$

being no cross-product term $\bar{u}_j \bar{u}_k, j \neq k$ in $\sigma(\bar{u}, s)$, we can write the positive polynomials $\lambda_j(\bar{u}, s)$ as the summation of three SOS polynomials

$$\lambda_j(\bar{u}, s) = \sum_{k=1}^{3} \lambda_{j,k}(\bar{u}_k, s) \tag{A.14a}$$

$$\lambda_{j,k}(\bar{u}_k, s) \in \mathbf{\Sigma}. \tag{A.14b}$$

For each monomial in the SOS polynomial $\lambda_{j,k}(\bar{u}_k, s)$, the degree of $\bar{u}_k$ and s_i for $i = 1, \ldots, n$ is either 0, 1, or 2. Hence the number of rows in the Gram matrix in $\lambda_{j,k}(\bar{u}_k, s)$ is of size 2^{n+1}. By choosing the reference frame according to the convention from Appendix A.5.1, n is no larger than $\lceil N/2 \rceil$ where N is the number of joints in the robot.

Remark 16. *For a 6-DOF UR3e robot whose collision geometries are approximated by cylinders, to certify the collision-avoidance between the robot and objects in the world (or self-collision), the largest positive semidefinite matrix in our optimization problem has rows at most* $2^{\lceil 6/2 \rceil + 1} = 2^4 = 16$*, where the division by 2 comes from choosing the middle link as the expressed link, hence halving the kinematic chain length to* $\lceil 6/2 \rceil$*.*

A.5.3 Parallelization

While it is attractive from a theoretical standpoint to write (5.18) as a single, large program, it is worth noting that it can, in fact, be viewed as K individual SOS programs, where K is the number of collision pairs in the environment. Indeed, certifying whether pairs $(\mathcal{A}_1, \mathcal{A}_2)$ are collision-free for all s in the polytope $\mathcal{P}$ can be done completely independently of the certification of another pair $(\mathcal{A}_1, \mathcal{A}_3)$ as the constraints are not coupled between any pairs. Similarly, the search for the largest inscribed ellipsoid can be done independently of the search for the separating hyperplanes.

Solving the certification program (5.18) as K individual SOS programs has several advantages. First, as written (5.18) has $2(m+1)K \sum_i |\mathcal{A}_i|$ semidefinite variables of various sizes, where m is the number of inequalities in $\mathcal{P}$ and $|\mathcal{A}_i|$ denotes the number of inequalities required to express that body $\mathcal{A}_i$ is on a particular side of the plane (see Table A.3). In the example from Section 5.6.1.2 this corresponds to 18,720 semidefinite variables. This can be prohibitively large to store in memory as a single program as the size of these semidefinite variables grows. Solving for the separating plane for each pair of collision bodies independently also enables us to determine which collision bodies cannot be certified as collision-free and allows us to terminate our search as soon as a single pair cannot be certified. Finally, decomposing the problems into subproblems enables us to increase computation speed by leveraging parallel processing.

The program (5.31) can also be solved completely independently of the certification program (5.18) and is in general a much smaller SDP than any individual certification program. Therefore, lines 3 and 4 of Algorithm 3 can be solved in parallel.

We note that (5.33) cannot be similarly decomposed because in this step the variables c_i^T and d_i affect all of the constraints. However, this program is substantially smaller as we have fixed $2mK \sum_i |\mathcal{A}_i|$ of the semidefinite variables as constants and replaced them with $2m$ linear variables representing the polytope. This program is much more amenable to being solved as a single program.

Appendix B

Appendices for Chapter 7

B.1 Supported Sets

We describe the current sets supported by `CCosmo`, their projection operators, and their automorphism groups. In short, we support the standard symmetric cones, bounding boxes, and zero constraints. For all the symmetric cones we support both isotropic scaling and the proposed rescaling from Section 7.4.2.2. For bounding boxes and the nonnegative orthant, we support arbitrary diagonal scalings.

Table B.1: Cone and set blocks used by the current `CCosmo` implementation. In the automorphism column, we mean invertible linear maps on the ambient space that preserve the set.

Set	Euclidean projection	Automorphism group
$\mathbb{R}^n$	$\Pi(z) = z$	$\mathrm{GL}(n)$
$\{0\} \subseteq \mathbb{R}^n$	$\Pi(z) = 0$	$\mathrm{GL}(n)$
$\mathbb{R}^n_+$	$(\Pi(z))_i = \max\{z_i, 0\}$	Matrices PD with P a permutation matrix and $D \succ 0$ diagonal
$[\ell, u] = \{z \in \mathbb{R}^n : \ell \leq z \leq u\}$	$(\Pi(z))_i = \min\{\max\{z_i, \ell_i\}, u_i\}$	generically trivial; otherwise permutations of identical intervals and sign flips on symmetric intervals
$\mathbb{L}^{n+1} = \{(t, x) \mid t \geq \|x\|\}$	$\Pi((t,x)) = \begin{cases} (t,x) & t \geq \|x\|_2 \\ (0,0) & \|x\|_2 \leq -t \\ (\tau, \frac{\tau}{\|x\|_2} x),\ \tau = \frac{t+\|x\|_2}{2} & \text{else} \end{cases}$	$T = sUO$ where $s > 0$, $U \in SO(n)$, and $O = \begin{bmatrix} \cosh(\theta) & \sinh(\theta) & 0 \\ \sinh(\theta) & \cosh(\theta) & 0 \\ 0 & 0 & I \end{bmatrix}$
$\mathbb{S}^n_+$	$\Pi(X) = U^T \Lambda_+ U$ where $X = U^T \Lambda U$ and $\Lambda_+ = \mathrm{diag}(\max\{\lambda_i, 0\})$	congruence maps $X \mapsto M^T X M$ with $M \in \mathrm{GL}(n)$

B.2 Locomotion MPC

The repeated locomotion benchmark uses planar center-of-mass state (c_t, v_t) with $c_t \in \mathbb{R}^3$, $v_t \in \mathbb{R}^3$, and contact forces $f_{i,t} = (f_{ix,t}, f_{iy,t}, f_{iz,t}) \in \mathbb{R}^3$ at stage-dependent foothold locations $p_{i,t} \in \mathbb{R}^3$. For one MPC solve, the decision variables are the state trajectory $\{c_t, v_t\}_{t=0}^{H}$ and the contact-force trajectory $\{f_{i,t}\}_{t=0}^{H-1}$.

For time step Δt, mass m, and per-stage force reference $\hat{f}_t$, each convex subproblem solves

$$
\begin{aligned}
\min \quad & \sum_{t=0}^{H} \tfrac{w_c}{2} \|c_t - c_t^{\mathrm{ref}}\|_2^2 + \sum_{t=0}^{H} \tfrac{w_v}{2} \|v_t - v_t^{\mathrm{ref}}\|_2^2 \\
& \quad + \sum_{t=0}^{H-1} \tfrac{w_{\mathrm{ref}}}{2} \|f_t - \hat{f}_t\|_2^2 + \sum_{t=0}^{H-1} \tfrac{w_f}{2} \|f_t\|_2^2 \\
& \quad + \sum_{t=1}^{H-1} \tfrac{w_{\Delta f}}{2} \|f_t - f_{t-1}\|_2^2 \\
\text{s.t.} \quad & c_{t+1} = c_t + \Delta t\, v_t, \\
& v_{t+1} = v_t + \Delta t\, \tfrac{1}{m} \sum_{i=1}^{k} (f_{ix,t}, f_{iy,t}, f_{iz,t}), \\
& \sum_{i=1}^{k} f_{iz,t} = mg, \\
& c_0 = c_{\mathrm{init}}^{(k)}, \quad v_0 = v_{\mathrm{init}}^{(k)}, \\
& \|(f_{ix,t}, f_{iy,t})\|_2 \le \mu f_{iz,t}, \qquad \forall i, t, \\
& 0 \le f_{iz,t} \le \bar{f}_{iz,t}, \qquad \forall i, t.
\end{aligned}
\tag{B.1}
$$

The paper configuration uses $H = 24$, a rollout length of 48 repeated MPC solves per regime, $\Delta t = 0.14\,\mathrm{s}$, $m = 24\,\mathrm{kg}$, $\mu = 0.55$, and active-contact normal-force bound $\bar{f}_{iz,t} = 94.176\,\mathrm{N}$. Swing feet use $\bar{f}_{iz,t} = 0$. The objective weights are

$$
\begin{aligned}
& w_c = 12.0, && w_v = 2.5, && w_{\mathrm{ref}} = 0.08, \\
& w_f = 0.002, && && w_{\Delta f} = 0.020.
\end{aligned}
$$

The initial footholds are

$$
\begin{aligned}
& p_1 = (0.30, 0.20, 0), && p_2 = (0.30, -0.20, 0), \\
& p_3 = (-0.28, 0.20, 0), && p_4 = (-0.28, -0.20, 0).
\end{aligned}
$$

The rollout starts from center of mass $(0, 0, 0.48)$ and tracks a forward transfer to $(0.72, 0, 0.48)$. On odd global stages, one foot is deactivated in the order $1, 4, 2, 3$ and lands one stage later after a $0.10\,\mathrm{m}$ forward step. Two regimes are used. In the nominal-tracking regime, the MPC initial state follows the reference rollout. In the push-recovery regime, deterministic velocity disturbances are injected at global stages 12, 24, and 36, and the MPC initial state is shifted by the resulting simulated center-of-mass error. At rollout index k, the MPC initial state $(c_{\mathrm{init}}^{(k)}, v_{\mathrm{init}}^{(k)})$ and the horizon references $(c_t^{\mathrm{ref}}, v_t^{\mathrm{ref}})$ are obtained by shifting a common global reference trajectory forward by k stages.

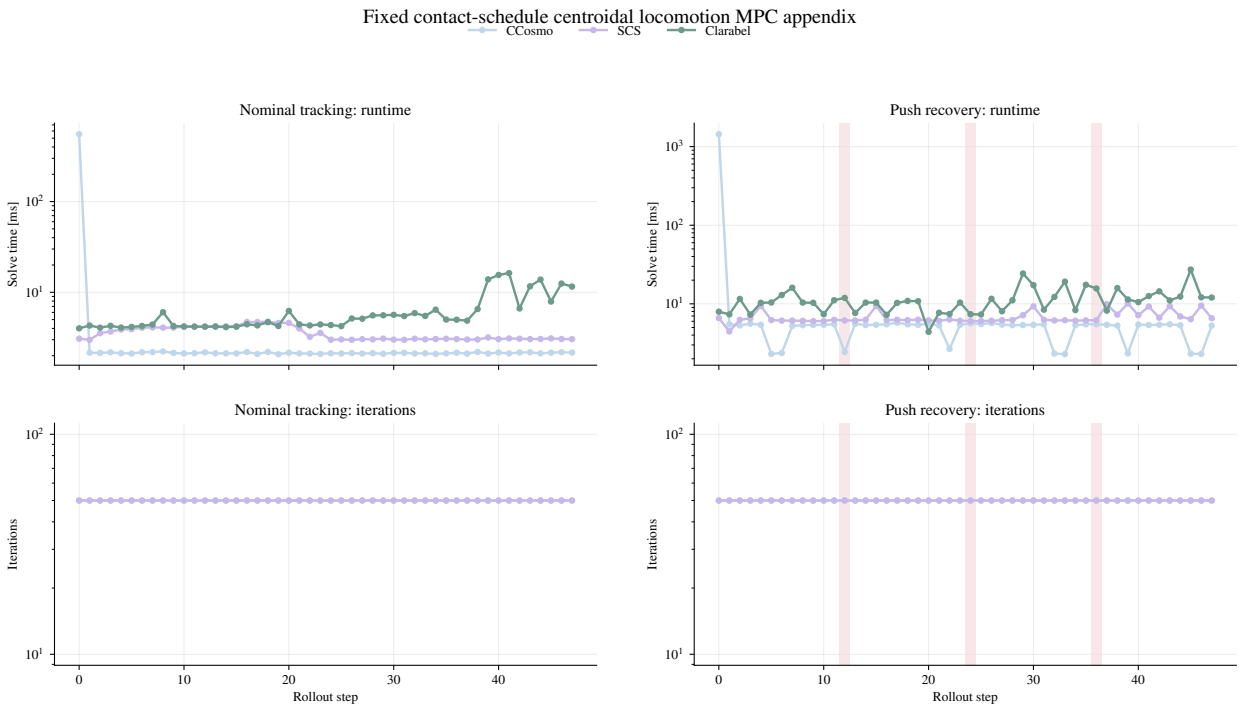


Figure B.1: View of the locomotion MPC benchmark showing per-solve runtime and iteration traces for the nominal-tracking and push-recovery regimes. Red bars indicate where the pushes occurred in the trajectory. Iterations are only shown for first-order solvers.

B.3 Breakdown of CPU/GPU Transfer Times for Section 7.6

We provide a breakdown of where the GPU implementation of `CCosmo` is spending time when solving a batch of programs. In what follows, the CPU solve column is the CPU local-solver solve time. The GPU solve column is the device factorization time plus the solve-kernel time only. The GPU total column is the host-to-device upload, device allocation/setup, GPU solve, and device-to-host readback. Host packing, host-side setup, constructor-time factorization, and host finalization are reported as excluded overhead rather than included in the speedup denominator. Here host finalization is CPU-side post-readback work that turns copied GPU buffers into public solver results: finalizing result vectors, forming the reported dual solution from the scaled dual, setting status and iteration fields, and recomputing the objective value. It does not include GPU factorization, GPU kernel execution, or raw device-to-host copy time. The effective PCIe usage column is the size of the copied solution payload divided by the measured device-to-host readback time; it should be interpreted as observed end-to-end copy-path utilization for this implementation, not as the hardware peak PCIe bandwidth.

The readback times in Tables B.3 and B.5 are poor because of the current result extraction implementation, not because the solution vectors are large. For the full `Maze` vertices-and-edges batch, only about 0.416 MiB is copied from device to host, but the implementation issues roughly 7838 small device-to-host copies. The `ContactGraph` batches have larger,

but still modest, solution payloads: `cg_simple_4` copies about 3.335 MiB through roughly 16994 small device-to-host copies, and the `cg_maze_b1` prefix copies about 9.552 MiB through roughly 49820 small device-to-host copies. The resulting effective bandwidth of only a few MiB/s indicates copy-launch and synchronization overhead from many tiny transfers rather than raw PCIe bandwidth.

Batch	Supported programs	CPU solve (ms)	GPU solve (ms)	H2D upload (ms)	Device setup (ms)	Solve factor (ms)	Kernel (ms)	D2H read (ms)	GPU total (ms)	Excluded overhead (ms)
`cg_simple_4`	8486	1830.34	227.14	43.12	117.33	35.83	191.31	213.65	601.23	463.68
`cg_maze_b1` (25k)	24899	5985.10	672.57	104.92	233.83	47.08	625.49	558.23	1569.55	1128.58

Table B.2: Detailed timing decomposition for the current regularized Shao `Cuda` local-program runs. The CPU solve column reports the CPU local-solver time over GPU-supported programs. GPU solve is only device factorization plus kernel execution. The GPU total is host-to-device upload, device allocation/setup, GPU solve, and device-to-host readback. The excluded-overhead column sums host packing, host setup, the constructor-time factorization, and host finalization.

Batch	D2H payload (MiB)	D2H copy calls	D2H time (ms)	Effective PCIe usage (MiB/s)	CPU/GPU solve	CPU/ GPU total
`cg_simple_4`	3.335	16994	213.65	15.61	8.06x	3.04x
`cg_maze_b1` (25k)	9.552	49820	558.23	17.11	8.90x	3.81x

Table B.3: Device-to-host readback diagnostics for the same Shao `Cuda` runs. Effective PCIe usage is the copied solution payload divided by the measured device-to-host readback time; it is not the hardware peak bandwidth. The GPU denominator excludes host packing, host setup, constructor-time factorization, and host finalization, and includes only host-to-device upload, device allocation/setup, GPU solve, and device-to-host readback.

Batch	Supported programs	CPU solve (ms)	GPU solve (ms)	H2D upload (ms)	Device setup (ms)	Solve factor (ms)	Kernel (ms)	D2H read (ms)	GPU total (ms)	Excluded overhead (ms)
`Maze 30_10_48`, vertices and edges	3916	78.25	21.34	1.71	162.19	14.75	6.58	120.62	305.86	432.89
`Maze 30_10_48`, edges only	3016	51.34	10.54	4.64	181.95	6.67	3.87	94.43	291.56	375.08
`Maze 30_10_48`, vertices only	900	34.94	13.38	0.24	210.29	10.73	2.66	30.51	254.43	415.36

Table B.4: Detailed timing decomposition for the current `Cuda` runs on the `Maze 30_10_48` local programs. The CPU solve column reports the CPU local-solver time over GPU-supported programs. GPU solve is only device factorization plus kernel execution. The GPU total is host-to-device upload, device allocation/setup, GPU solve, and device-to-host readback. The excluded-overhead column sums host packing, host setup, the constructor-time factorization, and host finalization.

Batch	D2H payload (MiB)	D2H copy calls	D2H time (ms)	Effective PCIe usage (MiB/s)	CPU/GPU solve	CPU/GPU total
`Maze 30_10_48`, vertices and edges	0.416	7838	120.62	3.45	3.67x	0.26x
`Maze 30_10_48`, edges only	0.299	6034	94.43	3.17	4.87x	0.18x
`Maze 30_10_48`, vertices only	0.117	1804	30.51	3.82	2.61x	0.14x

Table B.5: Device-to-host readback diagnostics for the same Tobia `Cuda` runs. Effective PCIe usage is the copied solution payload divided by the measured device-to-host readback time; it is not the hardware peak bandwidth. The GPU denominator excludes host packing, host setup, constructor-time factorization, and host finalization, and includes only host-to-device upload, device allocation/setup, GPU solve, and device-to-host readback.

Appendix C

Appendix for Chapter 9

C.1 `GcsBench` Instances

We collect information about each instance in the `GcsBench` benchmark. Table C.2 reports the graph size and the aggregate size of the local conic programs stored on the original GCS graph object before forming the shortest-path relaxation. The GCS variables, constraints, nonzeros, and cone description are sums over the vertex and edge programs attached to the graph.

Table C.4 reports the size of the centralized conic relaxation (3.17). For each instance, we report the stored optimal value of the convex relaxation when such a solve is available, preferring the interior point solver `Clarabel` [22] run to 10^{-8} relative and absolute tolerance. We also report the number of non-zeros in cost and constraint matrices, the number of variables, and break down the size of each cone in the conic constraint. The cone description is a tuple (z, l, q, s). The entry z gives the number of equality constraints, and the entry l gives the number of linear inequalities. The entries q and s represent the Lorentz and PSD cones, respectively. For these cones we give a list of $n \times d$ where n is the number of cones of that type of dimension d. For example, $q = 900 \times 3, 371 \times 4$ means the program has 900 Lorentz cones of size 3 and 371 of size 4. For the SDP cone, we report the packed dimension size which is the number of entries in the lower triangular matrix representing the cone. The sum of the dimensions in a cone column corresponds to the constraints column in the same table.

For each instance, we describe the problem and its naming convention.

Instance	$\|V\|$	$\|E\|$	GCS vars	GCS constraints	GCS nnz	GCS cones
Mazes: Piecewise Linear						
`5_0_5_1_False_2.66_0.184`	25	80	1,110	556	816	z=164; l=217 q=25x4, 25x3
`5_2_57_1_False_2.66_0.184`	25	92	1,254	580	864	z=188; l=217 q=25x4, 25x3
`10_5_5_1_False_2.12_1.877`	100	356	4,872	2,308	3,420	z=716; l=892 q=100x4, 100x3
Mazes: Cubic Curves						
`5_0_58_3_True_2.07_0.107`	25	70	2,145	1,276	3,061	z=284; l=417 q=25x8, 25x6, 75x3
`10_2_28_3_True_2.6_0.153`	100	322	9,672	5,284	12,760	z=1,292; l=1,692 q=100x8, 100x6, 300x3
`15_0_86_3_True_2.43_0.193`	225	782	23,257	12,124	29,405	z=3,132; l=3,817 q=225x8, 225x6, 675x3
Helicopter						
`75_0_0_5_2_0.04_0.12_1.6_4.0_1.0_0`	75	396	6,315	2,260	3,919	z=1; l=846 q=471x3
`150_0_0_10_2_0.03_0.078_1.8_4.0_0.88_68`	150	906	14,340	4,975	8,749	z=1; l=1,806 q=1056x3
`200_0_0_5_10_0.036_0.07_2.0_3.5_1.28_7`	200	1,024	16,360	5,897	10,193	z=1; l=2,224 q=1224x3
`300_0_0_2_5_0.014_0.057_1.78_4.65_1.34_45`	300	4,806	73,590	21,925	41,449	z=1; l=6,606 q=5106x3
Shelf: Order 1						
`Left Bin_to_Right Bin`	11	44	1,457	3,152	18,964	z=322; l=2,830

Continued on next page

Instance	$\|V\|$	$\|E\|$	GCS vars	GCS constraints	GCS nnz	GCS cones
Left to Shelve_to_Front to Shelve	11	45	1,480	3,159	18,978	z=329; l=2,830
Right to Shelve_to_Left to Shelve	11	44	1,457	3,152	18,964	z=322; l=2,830
Top Rack_to_Above Shelve	11	44	1,457	3,152	18,964	z=322; l=2,830
Shelf: Order 3						
Left Bin_to_Right Bin	11	44	2,787	6,086	37,654	z=322; l=5,764
Left to Shelve_to_Front to Shelve	11	45	2,824	6,093	37,668	z=329; l=5,764
Right to Shelve_to_Left to Shelve	11	44	2,787	6,086	37,654	z=322; l=5,764
Top Rack_to_Above Shelve	11	44	2,787	6,086	37,654	z=322; l=5,764
Bimanual						
0.017_0.041_5_1_0_92	15	38	1,602	1,385	6,815	z=320; l=948 q=13x9
0.033_0.043_1_2_0_82	15	38	2,385	2,035	10,183	z=320; l=1,481 q=26x9
0.036_0.046_5_3_2_45	15	40	3,312	3,365	16,861	z=944; l=2,070 q=39x9
Planar Pushing						
triangle_0.015_19_2	20	50	10,343	4,391	9,511	z=1,197; l=2,162 q=15x8, 93x3 s=3x15, 3x13
box_0.02_14_45	38	98	18,284	6,740	14,160	z=1,828; l=3,236 q=32x8, 192x3 s=4x15, 4x13

Continued on next page

Instance	$\lvert V\rvert$	$\lvert E\rvert$	GCS vars	GCS constraints	GCS nnz	GCS cones
`tee_0.015_27_43`	190	490	74,848	22,961	44,377	z=5,872; l=10,337 q=180x8, 1208x3 s=8x15, 8x13
Contact Graph						
`cg_simple_4`	194	8,328	198,546	41,216	78,992	z=33,316; l=6,940 q=192x5
`cg_trichal4`	200	12,678	299,990	58,882	114,966	z=50,716; l=7,176 q=198x5
`cg_maze_b1`	628	56,805	1,330,976	252,855	498,311	z=227,224; l=22,501 q=626x5

Table C.2: GCS graph-object instance information for `GcsBench`. Variable, constraint, and nonzero counts are sums over the vertex and edge conic programs stored on the graph before forming the shortest-path relaxation; cone descriptions aggregate those same local programs.

Shortest Path Convex Relaxation

Instance	Cost	relax. vars	relax. constraints	relax. nnz	Cones
Mazes: Piecewise Linear					
5_0_5_1_False_2.66_0.184	8.525	1,215	6,194	17,214	z=576; l=3,378 q=320x4, 320x3
5_2_57_1_False_2.66_0.184	6.556	1,371	7,034	19,728	z=624; l=3,834 q=368x4, 368x3
10_5_5_1_False_2.12_1.877	16.127	5,328	27,400	77,208	z=2,344; l=15,088 q=1424x4, 1424x3
Mazes: Cubic Curves					
5_0_58_3_True_2.07_0.107	7.972	2,240	12,668	49,782	z=1,062; l=5,166 q=280x8, 280x6, 840x3
10_2_28_3_True_2.6_0.153	31.202	10,094	57,908	230,502	z=4,320; l=23,964 q=1288x8, 1288x6, 3864x3
15_0_86_3_True_2.43_0.193	68.909	24,264	140,108	560,244	z=9,910; l=58,254 q=3128x8, 3128x6, 9384x3
Helicopter					
75_0_0_5_2_0.04_0.12_1.6_4.0_1.0_0	14.818	6,786	19,428	48,744	z=1,062; l=12,426 q=1980x3
150_0_0_10_2_0.03_0.078_1.8_4.0_0.88_68	30.113	15,396	44,088	111,144	z=2,112; l=28,386 q=4530x3
200_0_0_5_10_0.036_0.07_2.0_3.5_1.28_7	23.274	17,584	50,312	126,092	z=2,808; l=32,144 q=5120x3
300_0_0_2_5_0.014_0.057_1.78_4.65_1.34_45	11.323	78,696	225,888	581,544	z=4,212; l=149,586 q=24030x3
Shelf: Order 1					
Left Bin_to_Right Bin	1.796	1,512	54,626	600,208	z=664; l=53,962

Continued on next page

Instance	Cost	relax. vars	relax. constraints	relax. nnz	Cones
Left to Shelve_to_Front to Shelve	0.539	1,536	55,233	606,734	z=685; l=54,548
Right to Shelve_to_Left to Shelve	1.079	1,512	55,394	609,262	z=664; l=54,730
Top Rack_to_Above Shelve	0.874	1,512	55,038	605,296	z=664; l=54,374
Shelf: Order 3					
Left Bin_to_Right Bin	1.796	2,842	110,610	1,208,768	z=916; l=109,694
Left to Shelve_to_Front to Shelve	0.539	2,880	111,817	1,221,872	z=937; l=110,880
Right to Shelve_to_Left to Shelve	1.079	2,842	112,146	1,226,894	z=916; l=111,230
Top Rack_to_Above Shelve	0.874	2,842	111,434	1,218,944	z=916; l=110,518
Bimanual					
0.017_0.041_5_1_0_92	5.349	1,655	13,426	123,996	z=880; l=11,214 q=148x9
0.033_0.043_1_2_0_82	5.393	2,438	21,046	187,314	z=1,114; l=17,268 q=296x9
0.036_0.046_5_3_2_45	5.82	3,367	30,894	265,722	z=1,972; l=24,710 q=468x9
Planar Pushing					
triangle_0.015_19_2	50.387	10,413	50,628	190,158	z=13,368; l=25,336 q=160x8, 1016x3 s=36x15, 36x13
box_0.02_14_45	29.737	18,420	91,418	343,264	z=22,894; l=46,032 q=324x8, 2132x3 s=64x15, 64x13

Continued on next page

Instance	Cost	relax. vars	relax. constraints	relax. nnz	Cones
tee_0.015_27_43	101.636	75,528	397,756	1,518,823	z=87,478; l=199,384 q=1700x8, 14426x3 s=256x15, 256x13
Contact Graph					
cg_simple_4	–	207,068	1,438,382	4,864,408	z=38,662; l=1,233,400 q=33264x5
cg_trichal4	–	312,868	2,170,878	7,602,734	z=56,258; l=1,861,420 q=50640x5
cg_maze_b1	–	1,388,409	9,575,616	34,072,276	z=244,580; l=8,195,626 q=227082x5

Table C.4: Conic shortest-path relaxation information for GcsBench. These dimensions are measured after forming the centralized relaxation in conic standard form.

C.1.1 `Maze`

The `Maze` instances model finding a shortest path in a maze. The original instance for `GCSOPT` searches for piecewise-linear paths through the maze. We enhance the original formulation to search for paths composed of degree d Bezier curves, subject to velocity continuity constraints, a maximum velocity, and also a penalty on the acceleration. Mazes are generated randomly to have a unique path through them. Then the maze is modified by "knocking down" `knockdowns` many additional walls to potentially generate many paths through the maze. The `Maze` instance names have the form `mazesize_knockdowns_seed_degree_velocity_continuity_vmax_acc_weight`. Thus, `5_0_58_3_True_2.07_0.107` denotes a 5×5 maze, no knocked-down obstacles, random seed 58, cubic Bezier curves, enforced velocity continuity, velocity bound $v_{\max} = 2.07$, and acceleration penalty weight 0.107. The piecewise-linear paths are the degree-1 cases and cannot enforce velocity continuity without being infeasible.

C.1.2 `Helicopter`

The `Helicopter` instances model flight planning between circular islands that serve as charging stations. The helicopter starts with a full battery and may recharge after landing on an island; edges connect islands that are close enough to traverse on a single battery discharge.

The `Helicopter` instance names have the form `num_islands_bounds_x_min_bounds_y_min_bounds_x_max_bounds_y_max_min_radius_max_radius_speed_discharge_rate_charge_rate_seed`. Thus, `150_0_0_10_2_0.03_0.078_1.8_4.0_0.88_68` denotes an instance with 150 islands in the rectangle $[0, 10] \times [0, 2]$, island radii sampled between 0.03 and 0.078, helicopter speed 1.8, battery discharge rate 4.0, charging rate 0.88, and random seed 68. The curated instances in Table C.2 were screened so the start and goal islands lie in the same graph component, and each row has a finite optimal value from `Clarabel`.

C.1.3 `IiwaShelf`

The `IiwaShelf` examples are generated using a modification of the notebook from [154, §6.4.4] by growing convex sets in configuration space using IrisNp2 [147]. A collision-free motion planning problem is formulated through these sets. The `IiwaShelf` names have the form `order_p/source_to_target`, where p is the Bezier order used on each motion segment. For example, `order_3/Left Bin_to_Right Bin` is the order-3 shelf problem that moves between the IRIS regions named "Left Bin" and "Right Bin". The remaining route labels read literally: from "Left to Shelve" to "Front to Shelve," from "Right to Shelve" to "Left to Shelve," and from "Top Rack" to "Above Shelve."

C.1.4 `Bimanual`

Similar to `IiwaShelf`, the `Bimanual` instances are generated by growing convex sets in configuration space using IrisNp2 [147] and then planning a collision-free motion plan through the sets. They are adapted from [93].

The `Bimanual` instance names record, in order, IRIS `delta`, IRIS `epsilon`, IRIS `iteration cap`, Bezier curve degree used for planning, the order of continuity between segments, and the random seed. Thus, `0.036_0.046_5_3_2_45` denotes a benchmark with IRIS `delta` 0.036, IRIS `epsilon` 0.046, an IRIS `iteration cap` of five, cubic Bezier segments, continuity order two between adjacent segments, and random seed 45. The first three fields control the region-generation by IRISNP2, while the final three control the trajectory parameterization and randomized instance seed. Typically, the convex sets grown by IRISNP2 will have more faces as IRIS `delta` and IRIS `epsilon` decrease, and as IRIS `iteration cap` increases. A higher Bezier order increases the size of the trajectory program attached to each vertex, while a higher continuity order enlarges the edge programs because additional path-continuity constraints are imposed between adjacent segments.

C.1.5 `SdpPushing`

The `SdpPushing` examples are taken from [95]. The `SdpPushing` names record, in order, the slider shape, pusher radius, random seed, and boundary-condition index. Thus `triangle_0.015_19_2` denotes the triangle slider with pusher radius 0.015, random seed 19, and the third selected boundary condition from that seed's generated library.

C.1.6 `ContactGraph`

The `ContactGraph` examples are taken from [96]. These are intentionally designed to push the limits of using direct convex optimization to solve GCS instances. Indeed, in [96], the authors introduce an implicit graph search approach to solve these programs. We include the instances `cg_simple_4` and `cg_maze_b1`, which are referred to as `AROUND` and `SQUEEZE` in [96]. We also include `cg_trichal4` which is smaller than `cg_maze_b1` but larger than `cg_simple_4`.

For these instances, we do not include an optimal solution as `Clarabel` was unable to return one in under two hours.

C.2 Shortest Path in a GCS on `GcsBench`

The shorest path problem in a GCS can be challenging to solve to even moderately low accuracy for many solvers. In this section, we collect the performance of some popular open-source solvers on these relaxations. In Table C.5, we demonstrate the time it takes for `CCosmo`, `SCS`, and `Clarabel` to solve to 10^{-3} absolute and 10^{-4} relative accuracy. We also include `Clarabel` in high-accuracy mode, solving to 10^{-8} absolute and relative accuracy. In addition, in Table C.6 we provide a table showing how far off each low-accuracy solver's cost is from the high-accuracy baseline. This is relevant as many solvers report more than a 1% gap even when declaring optimality. Additionally, the metric gives a rough measure on whether the solver made any progress towards the solution. For example, `CCosmo` struggles substantially on `Helicopter`.

Problem	Clarabel high	Clarabel low	CCosmo	SCS
Maze 5_0_5_1_False_2.66_0.184	0.0438	0.0407	0.453	0.0859
Maze 5_2_57_1_False_2.66_0.184	0.069 (inacc)	0.0559	0.311	0.171
Maze 10_5_5_1_False_2.12_1.877	0.394 (inacc)	0.416	1.28	2.17
Maze 5_0_58_3_True_2.07_0.107	0.408 (inacc)	0.134	0.131	0.137
Maze 10_2_28_3_True_2.6_0.153	3.65 (inacc)	0.992	6.42	6.76
Maze 15_0_86_3_True_2.43_0.193	8.96 (inacc)	4.21	53.1 (inacc)	73 (inacc)
Helicopter 75_0_0_5_2_0.04_0.12_1.6_4.0_1.0_0	0.999	0.52	8.07	6.2
Helicopter 150_0_0_10_2_0.03_0.078_1.8_4.0_0.88_68	11.1	6.59	17.4 (max)	21.1 (inacc)
Helicopter 200_0_0_5_10_0.036_0.07_2.0_3.5_1.28_7	2.95	1.66	16.3	16.2
Helicopter 300_0_0_2_5_0.014_0.057_1.78_4.65_1.34_45	96	68.8	285 (max)	430 (inacc)
IiwaShelf Order 1 Left Bin_to_Right Bin	2.29	1.89	2.47	48.2
IiwaShelf Order 1 Left to Shelve_to_Front to Shelve	2.76 (inacc)	2.45	2.9	14.7
IiwaShelf Order 1 Right to Shelve_to_Left to Shelve	2.53	1.69	2.4	8.51
IiwaShelf Order 1 Top Rack_to_Above Shelve	3.81 (inacc)	3.75	44.3 (inacc)	48.6 (inacc)
IiwaShelf Order 3 Left Bin_to_Right Bin	5.7	5	11.2	108 (inacc)
IiwaShelf Order 3 Left to Shelve_to_Front to Shelve	7.63	5.6	13.6	97.4 (inacc)
IiwaShelf Order 3 Right to Shelve_to_Left to Shelve	6.52	4.88	11.2	92.6 (inacc)
IiwaShelf Order 3 Top Rack_to_Above Shelve	10.6 (inacc)	6.51	55.4 (inacc)	56.6 (inacc)
Bimanual 0.017_0.041_5_1_0_92	0.956 (inacc)	0.325	5.67 (max)	3.09
Bimanual 0.033_0.043_1_2_0_82	1.4 (inacc)	0.593	7.91 (max)	2.07
Bimanual 0.036_0.046_5_3_2_45	12.5 (err)	4.99	14.2 (max)	13.9 (inacc)
SdpPushing triangle_0.015_19_2	69.95 (inacc)	55.49	84.44 (inacc)	154
SdpPushing box_0.02_14_45	228.1 (inacc)	152.9	222.6 (max)	352 (inacc)
SdpPushing tee_0.015_27_43	456.7 (err)	385.7 (err)	703.6 (inacc)	529.7 (inacc)

Table C.5: Solve times, in seconds, for GcsBench instances. The low-accuracy Clarabel setting uses the same termination tolerances as the first-order solvers, $\epsilon_{\text{abs}} = 10^{-3}$ and $\epsilon_{\text{rel}} = 10^{-4}$; the high-accuracy Clarabel setting uses Clarabel defaults.

C.2.1 Regularized Problems

We also report the detailed centralized solver behavior on the edge-regularized $\epsilon = 10$ versions of the same GcsBench instances. Table C.7 gives the per-instance solve times, Table C.8 compares each regularized solve time to the corresponding original solve time, and Table C.9 reports the returned objective values and relative gaps to the high-accuracy Clarabel reference. These detailed tables include all 24 non-ContactGraph instances.

Problem	Ref.	Clarabel low	CCosmo	SCS
Maze 5_0_5_1_False_2.66_0.184	8.5246	8.5248 (+0.00%)	8.525 (+0.00%)	8.5246 (-0.00%)
Maze 5_2_57_1_False_2.66_0.184	6.5557 (inacc)	6.5559 (+0.00%)	6.5557 (+0.00%)	6.5573 (+0.03%)
Maze 10_5_5_1_False_2.12_1.877	16.127 (inacc)	16.128 (+0.01%)	16.125 (-0.02%)	16.123 (-0.02%)
Maze 5_0_58_3_True_2.07_0.107	7.9715 (inacc)	7.9729 (+0.02%)	7.9715 (+0.00%)	7.9716 (+0.00%)
Maze 10_2_28_3_True_2.6_0.153	31.201 (inacc)	31.204 (+0.01%)	31.195 (-0.02%)	31.191 (-0.03%)
Maze 15_0_86_3_True_2.43_0.193	68.89 (inacc)	68.892 (+0.00%)	68.855 (-0.05%/inacc)	68.867 (-0.03%/inacc)
Helicopter 75_0_0_5_2_0.04_0.12_1.6_4.0_1.0_0	14.818	14.818 (+0.00%)	14.817 (-0.00%)	14.806 (-0.08%)
Helicopter 150_0_0_10_2_0.03_0.078_1.8_4.0_0.88_68	30.113	30.115 (+0.01%)	30.07 (-0.15%/max)	30.109 (-0.02%/inacc)
Helicopter 200_0_0_5_10_0.036_0.07_2.0_3.5_1.28_7	23.274	23.275 (+0.00%)	23.27 (-0.02%)	23.271 (-0.01%)
Helicopter 300_0_0_2_5_0.014_0.057_1.78_4.65_1.34_45	11.323	11.324 (+0.01%)	0.21675 (-98.09%/max)	11.111 (-1.87%/inacc)
IiwaShelf Order 1 Left Bin_to_Right Bin	1.7965	1.7965 (+0.00%)	1.7965 (-0.00%)	1.7963 (-0.01%)
IiwaShelf Order 1 Left to Shelve_to_Front to Shelve	0.53926 (inacc)	0.53941 (+0.03%)	0.53926 (-0.00%)	0.53915 (-0.02%)
IiwaShelf Order 1 Right to Shelve_to_Left to Shelve	1.0785	1.0786 (+0.00%)	1.0785 (+0.00%)	1.0784 (-0.01%)
IiwaShelf Order 1 Top Rack_to_Above Shelve	0.874 (inacc)	0.8752 (+0.14%)	0.87237 (-0.19%/inacc)	0.82376 (-5.75%/inacc)
IiwaShelf Order 3 Left Bin_to_Right Bin	1.7965	1.7978 (+0.08%)	1.7965 (-0.00%)	1.7808 (-0.88%/inacc)
IiwaShelf Order 3 Left to Shelve_to_Front to Shelve	0.53926	0.54588 (+1.23%)	0.53926 (+0.00%)	0.43531 (-19.28%/inacc)
IiwaShelf Order 3 Right to Shelve_to_Left to Shelve	1.0785	1.079 (+0.04%)	1.0785 (-0.00%)	1.056 (-2.08%/inacc)
IiwaShelf Order 3 Top Rack_to_Above Shelve	0.874 (inacc)	0.88271 (+1.00%)	0.87281 (-0.14%/inacc)	0.81336 (-6.94%/inacc)
Bimanual 0.017_0.041_5_1_0_92	5.3494 (inacc)	5.3516 (+0.04%)	5.4503 (+1.89%/max)	5.3411 (-0.16%)
Bimanual 0.033_0.043_1_2_0_82	5.3929 (inacc)	5.3964 (+0.07%)	5.478 (+1.58%/max)	5.3813 (-0.22%)
Bimanual 0.036_0.046_5_3_2_45	5.832 (err)	5.8358 (–)	5.8811 (–/max)	5.798 (–/inacc)
SdpPushing triangle_0.015_19_2	44.85 (inacc)	29.89 (-33.36%)	44.93 (+0.18%/inacc)	44.94 (+0.20%)
SdpPushing box_0.02_14_45	28.96 (inacc)	12.8 (-55.82%)	29.06 (+0.35%/max)	29.03 (+0.24%/inacc)
SdpPushing tee_0.015_27_43	-1044 (err)	-1044 (–/err)	93.75 (–/inacc)	93.3 (–/inacc)

Table C.6: Returned cost, relative error to the high-accuracy `Clarabel` solution, and status for each solver on the GCS bench instances `GcsBench`. The reference column reports the high-accuracy `Clarabel` returned cost and status. Relative errors are omitted when this reference row did not return a successful status.

Problem	Clarabel high	Clarabel low	CCosmo	SCS
Maze 5_0_5_1_False_2.66_0.184	0.03861	0.0295	0.03645	0.0195
Maze 5_2_57_1_False_2.66_0.184	0.05174	0.04303	0.05201	0.06123
Maze 10_5_5_1_False_2.12_1.877	0.6126	0.3642	1.657	0.5819
Maze 5_0_58_3_True_2.07_0.107	0.1568	0.09962	0.06996	0.041
Maze 10_2_28_3_True_2.6_0.153	1.826	1.202	10.38	1.828
Maze 15_0_86_3_True_2.43_0.193	8.305	6.259	25.01	27.18
Helicopter 75_0_0_5_2_0.04_0.12_1.6_4.0_1.0_0	1.867	1.686	12.71 (max)	13.16 (inacc)
Helicopter 150_0_0_10_2_0.03_0.078_1.8_4.0_0.88_68	17.76	13.2	42.66 (max)	32.69 (inacc)
Helicopter 200_0_0_5_10_0.036_0.07_2.0_3.5_1.28_7	2.54	1.802	20.57 (max)	22.1 (inacc)
Helicopter 300_0_0_2_5_0.014_0.057_1.78_4.65_1.34_45	173.6	162.4	–	655.3 (inacc)
IiwaShelf Order 1 Left Bin_to_Right Bin	1.615 (inacc)	1.425	2.837	1.43
IiwaShelf Order 1 Left to Shelve_to_Front to Shelve	2.024	1.804	3.638	1.536
IiwaShelf Order 1 Right to Shelve_to_Left to Shelve	2.2	2.054	0.8872	1.126
IiwaShelf Order 1 Top Rack_to_Above Shelve	3.842	2.988	26.68	33.9
IiwaShelf Order 3 Left Bin_to_Right Bin	11.04 (inacc)	10.41	5.08	13.4
IiwaShelf Order 3 Left to Shelve_to_Front to Shelve	12.21	10.56	4.509	19.24
IiwaShelf Order 3 Right to Shelve_to_Left to Shelve	14.73 (inacc)	12.88	2.113	16.66
IiwaShelf Order 3 Top Rack_to_Above Shelve	23.2	17.27	45.02 (inacc)	181.8 (inacc)
Bimanual 0.017_0.041_5_1_0_92	0.3478	0.2169	3.229	0.1513
Bimanual 0.033_0.043_1_2_0_82	0.4599	0.3558	9.056 (max)	0.3386
Bimanual 0.036_0.046_5_3_2_45	3.039	2.275	7.289	0.5895
SdpPushing triangle_0.015_19_2	46.55 (inacc)	33.75	12.4	31.36
SdpPushing box_0.02_14_45	112.8 (inacc)	65.83	40.42 (inacc)	196.2 (inacc)
SdpPushing tee_0.015_27_43	483.2 (err)	488 (err)	214.3 (max)	403.4 (inacc)

Table C.7: Solve times, in seconds, for the edge-regularized $\epsilon = 10$ `GcsBench` instances. The low-accuracy `Clarabel` setting uses the same termination tolerances as the first-order solvers, $\epsilon_{\text{abs}} = 10^{-3}$ and $\epsilon_{\text{rel}} = 10^{-4}$; the high-accuracy `Clarabel` setting uses Clarabel defaults.

Problem	Clarabel high	Clarabel low	CCosmo	SCS
Maze 5_0_5_1_False_2.66_0.184	0.9341	0.7758	0.1018	0.2399
Maze 5_2_57_1_False_2.66_0.184	0.7809 (orig inacc)	0.8093	0.1166	0.3722
Maze 10_5_5_1_False_2.12_1.877	1.604 (orig inacc)	0.8966	1.32	0.2707
Maze 5_0_58_3_True_2.07_0.107	0.4075 (orig inacc)	0.7958	0.591	0.339
Maze 10_2_28_3_True_2.6_0.153	0.5097 (orig inacc)	1.244	1.633	0.2723
Maze 15_0_86_3_True_2.43_0.193	0.9405 (orig inacc)	1.507	0.473 (orig inacc)	0.3728 (orig inacc)
Helicopter 75_0_0_5_2_0.04_0.12_1.6_4.0_1.0_0	1.886	3.314	1.58 (reg max)	2.13 (reg inacc)
Helicopter 150_0_0_10_2_0.03_0.078_1.8_4.0_0.88_68	1.61	2.011	2.467 (reg max/orig max)	1.551 (reg inacc/orig inacc)
Helicopter 200_0_0_5_10_0.036_0.07_2.0_3.5_1.28_7	0.8751	1.108	1.264 (reg max)	1.368 (reg inacc)
Helicopter 300_0_0_2_5_0.014_0.057_1.78_4.65_1.34_45	1.81	2.363	–	1.53 (reg inacc/orig inacc)
IiwaShelf Order 1 Left Bin_to_Right Bin	0.7345 (reg inacc)	0.7781	1.291	0.02989
IiwaShelf Order 1 Left to Shelve_to_Front to Shelve	0.7568 (orig inacc)	0.7572	1.441	0.1067
IiwaShelf Order 1 Right to Shelve_to_Left to Shelve	0.9016	1.251	0.4182	0.1359
IiwaShelf Order 1 Top Rack_to_Above Shelve	1.034 (orig inacc)	0.8095	0.6062 (orig inacc)	0.704 (orig inacc)
IiwaShelf Order 3 Left Bin_to_Right Bin	2.001 (reg inacc)	2.139	0.5143	0.1252 (orig inacc)
IiwaShelf Order 3 Left to Shelve_to_Front to Shelve	1.64	1.924	0.3688	0.1989 (orig inacc)
IiwaShelf Order 3 Right to Shelve_to_Left to Shelve	2.333 (reg inacc)	2.702	0.2035	0.1816 (orig inacc)
IiwaShelf Order 3 Top Rack_to_Above Shelve	2.24 (orig inacc)	2.696	0.8252 (reg inacc/orig inacc)	3.244 (reg inacc/orig inacc)
Bimanual 0.017_0.041_5_1_0_92	0.3732 (orig inacc)	0.7006	0.5721 (orig max)	0.04931
Bimanual 0.033_0.043_1_2_0_82	0.3353 (orig inacc)	0.6206	1.15 (reg max/orig max)	0.1671
Bimanual 0.036_0.046_5_3_2_45	0.2443 (orig err)	0.4578	0.5161 (orig max)	0.04268 (orig inacc)
SdpPushing triangle_0.015_19_2	0.6662 (reg inacc/orig inacc)	0.6094	0.1473 (orig inacc)	0.2044
SdpPushing box_0.02_14_45	0.4948 (reg inacc/orig inacc)	0.4311	0.182 (reg inacc/orig max)	0.559 (reg inacc/orig inacc)
SdpPushing tee_0.015_27_43	1.059 (reg err/orig err)	1.267 (reg err/orig err)	0.3061 (reg max/orig inacc)	0.7647 (reg inacc/orig inacc)

Table C.8: Per-instance ratio of edge-regularized $\epsilon = 10$ solve time to original solve time for each centralized solver. Values below one indicate that the regularized problem solved faster. Parenthetical status labels identify non-optimal statuses in the regularized or original run.

Problem	Ref.	Clarabel low	CCosmo	SCS
Maze 5_0_5_1_False_2.66_0.184	98.7	98.71 (+0.00%)	98.7 (-0.00%)	98.7 (+0.00%)
Maze 5_2_57_1_False_2.66_0.184	56.57	56.58 (+0.02%)	56.57 (+0.00%)	56.57 (-0.00%)
Maze 10_5_5_1_False_2.12_1.877	166.2	166.2 (+0.01%)	166.2 (+0.00%)	166.2 (+0.00%)
Maze 5_0_58_3_True_2.07_0.107	77.97	77.99 (+0.02%)	77.97 (+0.00%)	77.97 (-0.00%)
Maze 10_2_28_3_True_2.6_0.153	321.3	321.3 (+0.01%)	321.3 (+0.00%)	321.3 (+0.00%)
Maze 15_0_86_3_True_2.43_0.193	631.4	631.5 (+0.02%)	631.3 (-0.00%)	631.3 (-0.00%)
Helicopter 75_0_0_5_2_0.04_0.12_1.6_4.0_1.0_0	151.7	151.7 (-0.00%)	150.5 (-0.81%/max)	148.2 (-2.31%/inacc)
Helicopter 150_0_0_10_2_0.03_0.078_1.8_4.0_0.88_68	289.1	289.1 (+0.00%)	290.1 (+0.34%/max)	286.8 (-0.81%/inacc)
Helicopter 200_0_0_5_10_0.036_0.07_2.0_3.5_1.28_7	301.8	301.9 (+0.00%)	301.9 (+0.01%/max)	303 (+0.39%/inacc)
Helicopter 300_0_0_2_5_0.014_0.057_1.78_4.65_1.34_45	141.4	141.4 (+0.01%)	–	141.4 (-0.02%/inacc)
IiwaShelf Order 1 Left Bin_to_Right Bin	41.8 (inacc)	41.83 (+0.07%)	41.8 (+0.00%)	41.8 (-0.00%)
IiwaShelf Order 1 Left to Shelve_to_Front to Shelve	30.61	30.62 (+0.03%)	30.61 (-0.00%)	30.61 (+0.00%)
IiwaShelf Order 1 Right to Shelve_to_Left to Shelve	41.08	41.09 (+0.02%)	41.08 (+0.01%)	41.08 (-0.00%)
IiwaShelf Order 1 Top Rack_to_Above Shelve	40.88	41 (+0.31%)	40.88 (+0.00%)	40.87 (-0.02%)
IiwaShelf Order 3 Left Bin_to_Right Bin	41.8 (inacc)	41.83 (+0.07%)	41.8 (-0.00%)	41.8 (-0.00%)
IiwaShelf Order 3 Left to Shelve_to_Front to Shelve	30.61	30.67 (+0.21%)	30.61 (+0.00%)	30.61 (-0.00%)
IiwaShelf Order 3 Right to Shelve_to_Left to Shelve	41.08 (inacc)	41.28 (+0.49%)	41.08 (+0.00%)	41.08 (+0.00%)
IiwaShelf Order 3 Top Rack_to_Above Shelve	40.88	41.4 (+1.28%)	40.88 (-0.00%/inacc)	40.87 (-0.02%/inacc)
Bimanual 0.017_0.041_5_1_0_92	95.67	95.7 (+0.03%)	95.66 (-0.01%)	95.65 (-0.02%)
Bimanual 0.033_0.043_1_2_0_82	95.64	95.67 (+0.03%)	95.63 (-0.01%/max)	95.55 (-0.09%)
Bimanual 0.036_0.046_5_3_2_45	86.65	86.66 (+0.02%)	86.65 (-0.00%)	86.63 (-0.02%)
SdpPushing triangle_0.015_19_2	106.2 (inacc)	86.18 (-18.86%)	106.3 (+0.11%)	106.3 (+0.06%)
SdpPushing box_0.02_14_45	84.04 (inacc)	70.29 (-16.36%)	84.14 (+0.12%/inacc)	84.06 (+0.02%/inacc)
SdpPushing tee_0.015_27_43	-307.4 (err)	-307.4 (–/err)	71.6 (–/max)	163.6 (–/inacc)

Table C.9: Returned cost, relative error to the high-accuracy Clarabel solution, and status for each solver on the edge-regularized $\epsilon = 10$ GcsBench instances. The reference column reports the high-accuracy Clarabel returned cost and status. Relative errors are omitted when this reference row did not return a successful status.

C.3 Detailed VEGA Results on GcsBench

We collect the per-instance VEGA timing, iteration, and solution-quality rows used to produce the summary figures in Figures 9.7, 9.8, 9.9, 9.10, 9.11 and 9.12.

C.3.1 Original Problems

Problem	Status	Time [s]	CCosmo iters	VEGA outer iters	VEGA inner iters
Maze 5_0_5_1_False_2.66_0.184	opt	1.458	3,325	2,501	0
Maze 5_2_57_1_False_2.66_0.184	opt	2.996	2,875	4,401	0
Maze 10_5_5_1_False_2.12_1.877	opt	12.05	1,450	4,876	0
Maze 5_0_58_3_True_2.07_0.107	opt	2.069	400	2,301	0
Maze 10_2_28_3_True_2.6_0.153	opt	26.97	5,450	6,726	0
Maze 15_0_86_3_True_2.43_0.193	opt	24.64	10,000	1,951	0
Helicopter 75_0_0_5_2_0.04_0.12_1.6_4.0_1.0_0	opt	4.095	8,500	1,476	0
Helicopter 150_0_0_10_2_0.03_0.078_1.8_4.0_0.88_68	opt	19.24	10,000	3,101	0
Helicopter 200_0_0_5_10_0.036_0.07_2.0_3.5_1.28_7	opt	19.87	7,825	2,626	0
Helicopter 300_0_0_2_5_0.014_0.057_1.78_4.65_1.34_45	opt	247.4	10,000	7,601	0
IiwaShelf Order 1 Left Bin_to_Right Bin	opt	12.44	725	1,326	0
IiwaShelf Order 1 Left to Shelve_to_Front to Shelve	opt	15.17	575	1,776	0
IiwaShelf Order 1 Right to Shelve_to_Left to Shelve	opt	19.97	775	2,326	0
IiwaShelf Order 1 Top Rack_to_Above Shelve	opt	16.71	10,000	1,176	0
IiwaShelf Order 3 Left Bin_to_Right Bin	opt	30.12	850	1,401	0
IiwaShelf Order 3 Left to Shelve_to_Front to Shelve	opt	44.45	1,075	2,151	0
IiwaShelf Order 3 Right to Shelve_to_Left to Shelve	opt	38.73	1,050	1,876	0
IiwaShelf Order 3 Top Rack_to_Above Shelve	opt	48.49	10,000	2,376	0
Bimanual 0.017_0.041_5_1_0_92	opt	5.138	10,000	2,051	0
Bimanual 0.033_0.043_1_2_0_82	opt	8.756	10,000	2,551	0
Bimanual 0.036_0.046_5_3_2_45	opt	12.84	10,000	2,501	0
SdpPushing triangle_0.015_19_2	max iter.	91.02	10,000	10,000	0
SdpPushing box_0.02_14_45	opt	85.97	10,000	3,451	0
SdpPushing tee_0.015_27_43	opt	417.8	10,000	4,326	0

Table C.10: Iteration counts for original problems.

Problem	Instance	Status	VEGA cost	Ref. cost	Rel. gap
Mazes: Piecewise Linear	5_0_5_1_False_2.66_0.184	opt	8.526	8.525	+0.02%
Mazes: Piecewise Linear	5_2_57_1_False_2.66_0.184	opt	6.561	6.556	+0.08%
Mazes: Piecewise Linear	10_5_5_1_False_2.12_1.877	opt	16.29	16.13	+1.01%
Mazes: Cubic Curves	5_0_58_3_True_2.07_0.107	opt	7.985	7.972	+0.17%
Mazes: Cubic Curves	10_2_28_3_True_2.6_0.153	opt	31.35	31.2	+0.49%
Mazes: Cubic Curves	15_0_86_3_True_2.43_0.193	opt	70.79	68.89	+2.75%
Helicopter	75_0_0_5_2_0.04_0.12_1.6_4.0_1.0_0	opt	14.87	14.82	+0.36%
Helicopter	150_0_0_10_2_0.03_0.078_1.8_4.0_0.88_68	opt	30.65	30.11	+1.79%
Helicopter	200_0_0_5_10_0.036_0.07_2.0_3.5_1.28_7	opt	23.71	23.27	+1.86%
Helicopter	300_0_0_2_5_0.014_0.057_1.78_4.65_1.34_45	opt	11.1	11.32	-1.94%
Shelf: Order 1	Left Bin_to_Right Bin	opt	1.796	1.796	-0.03%
Shelf: Order 1	Left to Shelve_to_Front to Shelve	opt	0.5393	0.5393	+0.00%
Shelf: Order 1	Right to Shelve_to_Left to Shelve	opt	1.078	1.079	-0.03%
Shelf: Order 1	Top Rack_to_Above Shelve	opt	0.9829	0.874	+10.89%
Shelf: Order 3	Left Bin_to_Right Bin	opt	1.796	1.796	-0.05%
Shelf: Order 3	Left to Shelve_to_Front to Shelve	opt	0.5391	0.5393	-0.02%
Shelf: Order 3	Right to Shelve_to_Left to Shelve	opt	1.078	1.079	-0.05%
Shelf: Order 3	Top Rack_to_Above Shelve	opt	0.9835	0.874	+10.95%
Bimanual	0.017_0.041_5_1_0_92	opt	5.549	5.349	+3.72%
Bimanual	0.033_0.043_1_2_0_82	opt	5.547	5.393	+2.86%
Bimanual	0.036_0.046_5_3_2_45	opt	6.141	5.832 (err.)	–
Planar Pushing	triangle_0.015_19_2	max iter.	14.83	50.49 (err.)	–
Planar Pushing	box_0.02_14_45	opt	23.58	29.44 (err.)	–
Planar Pushing	tee_0.015_27_43	opt	14.5	0 (err.)	–

Table C.11: Solution quality on original problems.

C.3.2 Original Problems in Half-Step Mode

Problem	Status	Time [s]	CCosmo iters	VEGA outer iters	Graph solves
Maze 5_0_5_1_False_2.66_0.184	opt	3.785	3,325	6,326	6,326
Maze 5_2_57_1_False_2.66_0.184	opt	1.513	2,875	4,851	4,851
Maze 10_5_5_1_False_2.12_1.877	opt	10.79	1,450	7,276	7,276
Maze 5_0_58_3_True_2.07_0.107	max iter.	6.767	400	20,000	20,000
Maze 10_2_28_3_True_2.6_0.153	max iter.	51.18	5,450	20,000	20,000
Maze 15_0_86_3_True_2.43_0.193	opt	191.1	10,000	19,251	19,251
Helicopter 75_0_0_5_2_0.04_0.12_1.6_4.0_1.0_0	opt	17.94	8,500	10,001	10,001
Helicopter 150_0_0_10_2_0.03_0.078_1.8_4.0_0.88_68	opt	59.6	10,000	9,976	9,976
Helicopter 200_0_0_5_10_0.036_0.07_2.0_3.5_1.28_7	opt	117.8	7,825	15,026	15,026
Helicopter 300_0_0_2_5_0.014_0.057_1.78_4.65_1.34_45	max iter.	863.6	10,000	20,000	20,000
IiwaShelf Order 1 Left Bin_to_Right Bin	opt	42.24	725	14,151	14,151
IiwaShelf Order 1 Left to Shelve_to_Front to Shelve	opt	41.55	575	13,826	13,826
IiwaShelf Order 1 Right to Shelve_to_Left to Shelve	opt	42.22	775	14,176	14,176
IiwaShelf Order 1 Top Rack_to_Above Shelve	max iter.	60.62	10,000	20,000	20,000
IiwaShelf Order 3 Left Bin_to_Right Bin	opt	156.2	850	19,201	19,201
IiwaShelf Order 3 Left to Shelve_to_Front to Shelve	max iter.	161.9	1,075	20,000	20,000
IiwaShelf Order 3 Right to Shelve_to_Left to Shelve	max iter.	164.1	1,050	20,000	20,000
IiwaShelf Order 3 Top Rack_to_Above Shelve	max iter.	160.1	10,000	20,000	20,000
Bimanual 0.017_0.041_5_1_0_92	max iter.	10.68	10,000	20,000	20,000
Bimanual 0.033_0.043_1_2_0_82	max iter.	14.07	10,000	20,000	20,000
Bimanual 0.036_0.046_5_3_2_45	max iter.	21.19	10,000	20,000	20,000
SdpPushing triangle_0.015_19_2	max iter.	53.91	10,000	20,000	20,000
SdpPushing box_0.02_14_45	opt	47.88	10,000	6,551	6,551
SdpPushing tee_0.015_27_43	opt	437.5	10,000	12,601	12,601

Table C.12: Half-step iteration counts for original problems.

Problem	Instance	Status	VEGA cost	Ref. cost	Rel. gap
Mazes: Piecewise Linear	5_0_5_1_False_2.66_0.184	opt	8.733	8.525	+2.45%
Mazes: Piecewise Linear	5_2_57_1_False_2.66_0.184	opt	7.911	6.556	+20.67%
Mazes: Piecewise Linear	10_5_5_1_False_2.12_1.877	opt	17.74	16.13	+10.02%
Mazes: Cubic Curves	5_0_58_3_True_2.07_0.107	max iter.	8.256	7.972	+3.56%
Mazes: Cubic Curves	10_2_28_3_True_2.6_0.153	max iter.	32.47	31.2	+4.07%
Mazes: Cubic Curves	15_0_86_3_True_2.43_0.193	opt	74.24	68.89	+7.76%
Helicopter	75_0_0_5_2_0.04_0.12_1.6_4.0_1.0_0	opt	28.63	14.82	+93.19%
Helicopter	150_0_0_10_2_0.03_0.078_1.8_4.0_0.88_68	opt	55.02	30.11	+82.72%
Helicopter	200_0_0_5_10_0.036_0.07_2.0_3.5_1.28_7	opt	58.49	23.27	+151.31%
Helicopter	300_0_0_2_5_0.014_0.057_1.78_4.65_1.34_45	max iter.	50.66	11.32	+347.38%
Shelf: Order 1	Left Bin_to_Right Bin	opt	1.93	1.796	+7.43%
Shelf: Order 1	Left to Shelve_to_Front to Shelve	opt	1.19	0.5393	+65.08%
Shelf: Order 1	Right to Shelve_to_Left to Shelve	opt	1.751	1.079	+62.40%
Shelf: Order 1	Top Rack_to_Above Shelve	max iter.	1.578	0.874	+70.37%
Shelf: Order 3	Left Bin_to_Right Bin	opt	2.39	1.796	+33.04%
Shelf: Order 3	Left to Shelve_to_Front to Shelve	max iter.	1.721	0.5393	+118.15%
Shelf: Order 3	Right to Shelve_to_Left to Shelve	max iter.	2.529	1.079	+134.48%
Shelf: Order 3	Top Rack_to_Above Shelve	max iter.	2.81	0.874	+193.60%
Bimanual	0.017_0.041_5_1_0_92	max iter.	32.25	5.349	+502.94%
Bimanual	0.033_0.043_1_2_0_82	max iter.	18.41	5.393	+241.31%
Bimanual	0.036_0.046_5_3_2_45	max iter.	14.38	5.832 (err.)	–
Planar Pushing	triangle_0.015_19_2	max iter.	56.93	50.49 (err.)	–
Planar Pushing	box_0.02_14_45	opt	26.9	29.44 (err.)	–
Planar Pushing	tee_0.015_27_43	opt	21.17	0 (err.)	–

Table C.13: Half-step solution quality on original problems.

C.3.3 Edge-Regularized Problems

Problem	Status	Time [s]	CCosmo iters	VEGA outer iters	VEGA inner iters
Maze 5_0_5_1_False_2.66_0.184	opt	55.05	400	551	80,965
Maze 5_2_57_1_False_2.66_0.184	opt	196.2	475	7,576	91,225
Maze 10_5_5_1_False_2.12_1.877	opt	252.4	1,975	7,226	617,055
Maze 5_0_58_3_True_2.07_0.107	opt	8.866	150	576	86,300
Maze 10_2_28_3_True_2.6_0.153	opt	152.8	4,700	5,351	902,970
Maze 15_0_86_3_True_2.43_0.193	opt	149.3	2,375	3,451	3,991,985
Helicopter 75_0_0_5_2_0.04_0.12_1.6_4.0_1.0_0	opt	170	10,000	9,201	1,749,950
Helicopter 150_0_0_10_2_0.03_0.078_1.8_4.0_0.88_68	max iter.	340.8	10,000	10,000	4,058,095
Helicopter 200_0_0_5_10_0.036_0.07_2.0_3.5_1.28_7	max iter.	397.2	10,000	10,000	2,380,385
Helicopter 300_0_0_2_5_0.014_0.057_1.78_4.65_1.34_45	opt	367.7	–	4,901	4,298,655
IiwaShelf Order 1 Left Bin_to_Right Bin	opt	39.78	850	1,426	280,620
IiwaShelf Order 1 Left to Shelve_to_Front to Shelve	opt	47.6	1,175	1,651	175,885
IiwaShelf Order 1 Right to Shelve_to_Left to Shelve	opt	59.27	325	2,101	259,865
IiwaShelf Order 1 Top Rack_to_Above Shelve	opt	36.99	9,800	1,301	264,695
IiwaShelf Order 3 Left Bin_to_Right Bin	opt	58.33	1,050	1,851	313,025
IiwaShelf Order 3 Left to Shelve_to_Front to Shelve	opt	80.83	975	2,526	287,460
IiwaShelf Order 3 Right to Shelve_to_Left to Shelve	max iter.	350.4	475	10,000	1,626,440
IiwaShelf Order 3 Top Rack_to_Above Shelve	opt	76.08	10,000	1,726	342,970
Bimanual 0.017_0.041_5_1_0_92	opt	52.38	4,800	1,776	489,195
Bimanual 0.033_0.043_1_2_0_82	opt	99.12	10,000	4,176	576,450
Bimanual 0.036_0.046_5_3_2_45	opt	55.94	4,700	2,301	505,785
SdpPushing triangle_0.015_19_2	max iter.	121.1	7,475	10,000	0
SdpPushing box_0.02_14_45	opt	86.55	10,000	3,301	0
SdpPushing tee_0.015_27_43	max iter.	912.5	10,000	10,000	0

Table C.14: Iteration counts and solve times for VEGA on edge-regularized GcsBench programs with $\epsilon = 10$. Inner iterations are the aggregate vertex- and edge-local solver iterations recorded.

Problem	Instance	Status	VEGA cost	Ref. cost	Rel. gap
Mazes: Piecewise Linear	5_0_5_1_False_2.66_0.184	opt	98.7	98.7	-0.00%
Mazes: Piecewise Linear	5_2_57_1_False_2.66_0.184	opt	56.57	56.57	+0.00%
Mazes: Piecewise Linear	10_5_5_1_False_2.12_1.877	opt	166.3	166.2	+0.06%
Mazes: Cubic Curves	5_0_58_3_True_2.07_0.107	opt	77.96	77.97	-0.01%
Mazes: Cubic Curves	10_2_28_3_True_2.6_0.153	opt	321.3	321.3	+0.01%
Mazes: Cubic Curves	15_0_86_3_True_2.43_0.193	opt	632.2	631.4	+0.13%
Helicopter	75_0_0_5_2_0.04_0.12_1.6_4.0_1.0_0	opt	148.8	151.7	-1.93%
Helicopter	150_0_0_10_2_0.03_0.078_1.8_4.0_0.88_68	max iter.	281.9	289.1	-2.47%
Helicopter	200_0_0_5_10_0.036_0.07_2.0_3.5_1.28_7	max iter.	299.3	301.8	-0.85%
Helicopter	300_0_0_2_5_0.014_0.057_1.78_4.65_1.34_45	opt	141.3	141.4	-0.08%
Shelf: Order 1	Left Bin_to_Right Bin	opt	41.8	41.8	-0.00%
Shelf: Order 1	Left to Shelve_to_Front to Shelve	opt	30.61	30.61	-0.00%
Shelf: Order 1	Right to Shelve_to_Left to Shelve	opt	41.08	41.08	+0.00%
Shelf: Order 1	Top Rack_to_Above Shelve	opt	41.03	40.88	+0.38%
Shelf: Order 3	Left Bin_to_Right Bin	opt	41.8	41.8	+0.01%
Shelf: Order 3	Left to Shelve_to_Front to Shelve	opt	30.61	30.61	+0.01%
Shelf: Order 3	Right to Shelve_to_Left to Shelve	max iter.	41.08	41.08	+0.01%
Shelf: Order 3	Top Rack_to_Above Shelve	opt	41.03	40.88	+0.37%
Bimanual	0.017_0.041_5_1_0_92	opt	98.26	95.67	+2.71%
Bimanual	0.033_0.043_1_2_0_82	opt	95.66	95.64	+0.02%
Bimanual	0.036_0.046_5_3_2_45	opt	86.65	86.65	+0.00%
Planar Pushing	triangle_0.015_19_2	max iter.	65.56	111.3 (err.)	–
Planar Pushing	box_0.02_14_45	opt	73.72	84.28 (err.)	–
Planar Pushing	tee_0.015_27_43	max iter.	65.88	0 (err.)	–

Table C.15: Solution quality of VEGA on edge-regularized GcsBench programs with $\epsilon = 10$. The reference is the high-accuracy Clarabel row when it returned a successful status; relative gaps are omitted when that reference was unavailable or unsuccessful.

C.3.4 Edge-Regularized Problems in Half-Step Mode

Problem	Status	Time [s]	CCosmo iters	VEGA outer iters	Graph solves
Maze 5_0_5_1_False_2.66_0.184	opt	2.062	400	5,976	5,976
Maze 5_2_57_1_False_2.66_0.184	opt	1.841	475	5,351	5,351
Maze 10_5_5_1_False_2.12_1.877	opt	17.07	1,975	10,001	10,001
Maze 5_0_58_3_True_2.07_0.107	opt	5.878	150	15,901	15,901
Maze 10_2_28_3_True_2.6_0.153	max iter.	53.06	4,700	20,000	20,000
Maze 15_0_86_3_True_2.43_0.193	opt	170.7	2,375	16,651	16,651
Helicopter 75_0_0_5_2_0.04_0.12_1.6_4.0_1.0_0	opt	10.32	10,000	6,001	6,001
Helicopter 150_0_0_10_2_0.03_0.078_1.8_4.0_0.88_68	opt	87.05	10,000	14,026	14,026
Helicopter 200_0_0_5_10_0.036_0.07_2.0_3.5_1.28_7	opt	148.8	10,000	17,626	17,626
Helicopter 300_0_0_2_5_0.014_0.057_1.78_4.65_1.34_45	opt	208.3	–	4,376	4,376
IiwaShelf Order 1 Left Bin_to_Right Bin	opt	44.56	850	14,626	14,626
IiwaShelf Order 1 Left to Shelve_to_Front to Shelve	opt	38.7	1,175	12,601	12,601
IiwaShelf Order 1 Right to Shelve_to_Left to Shelve	opt	47.3	325	15,776	15,776
IiwaShelf Order 1 Top Rack_to_Above Shelve	max iter.	58.91	9,800	20,000	20,000
IiwaShelf Order 3 Left Bin_to_Right Bin	max iter.	167.7	1,050	20,000	20,000
IiwaShelf Order 3 Left to Shelve_to_Front to Shelve	opt	138.5	975	16,576	16,576
IiwaShelf Order 3 Right to Shelve_to_Left to Shelve	max iter.	162.3	475	20,000	20,000
IiwaShelf Order 3 Top Rack_to_Above Shelve	max iter.	161.3	10,000	20,000	20,000
Bimanual 0.017_0.041_5_1_0_92	max iter.	9.628	4,800	20,000	20,000
Bimanual 0.033_0.043_1_2_0_82	max iter.	14.95	10,000	20,000	20,000
Bimanual 0.036_0.046_5_3_2_45	max iter.	22.21	4,700	20,000	20,000
SdpPushing triangle_0.015_19_2	max iter.	51.95	7,475	20,000	20,000
SdpPushing box_0.02_14_45	opt	67.84	10,000	10,151	10,151
SdpPushing tee_0.015_27_43	opt	517.7	10,000	15,326	15,326

Table C.16: Iteration counts and solve times for half-step VEGA on edge-regularized GcsBench programs with $\epsilon = 10$.

Problem	Instance	Status	VEGA cost	Ref. cost	Rel. gap
Mazes: Piecewise Linear	5_0_5_1_False_2.66_0.184	opt	101.9	98.7	+3.27%
Mazes: Piecewise Linear	5_2_57_1_False_2.66_0.184	opt	60.98	56.57	+7.80%
Mazes: Piecewise Linear	10_5_5_1_False_2.12_1.877	opt	167.8	166.2	+0.98%
Mazes: Cubic Curves	5_0_58_3_True_2.07_0.107	opt	78.03	77.97	+0.07%
Mazes: Cubic Curves	10_2_28_3_True_2.6_0.153	max iter.	324.7	321.3	+1.06%
Mazes: Cubic Curves	15_0_86_3_True_2.43_0.193	opt	645.5	631.4	+2.24%
Helicopter	75_0_0_5_2_0.04_0.12_1.6_4.0_1.0_0	opt	151.6	151.7	-0.09%
Helicopter	150_0_0_10_2_0.03_0.078_1.8_4.0_0.88_68	opt	291.2	289.1	+0.73%
Helicopter	200_0_0_5_10_0.036_0.07_2.0_3.5_1.28_7	opt	304	301.8	+0.72%
Helicopter	300_0_0_2_5_0.014_0.057_1.78_4.65_1.34_45	opt	141.6	141.4	+0.16%
Shelf: Order 1	Left Bin_to_Right Bin	opt	44.03	41.8	+5.33%
Shelf: Order 1	Left to Shelve_to_Front to Shelve	opt	41.14	30.61	+34.42%
Shelf: Order 1	Right to Shelve_to_Left to Shelve	opt	50.5	41.08	+22.92%
Shelf: Order 1	Top Rack_to_Above Shelve	max iter.	44.65	40.88	+9.23%
Shelf: Order 3	Left Bin_to_Right Bin	max iter.	44.5	41.8	+6.47%
Shelf: Order 3	Left to Shelve_to_Front to Shelve	opt	41.78	30.61	+36.49%
Shelf: Order 3	Right to Shelve_to_Left to Shelve	max iter.	52.25	41.08	+27.20%
Shelf: Order 3	Top Rack_to_Above Shelve	max iter.	47.66	40.88	+16.59%
Bimanual	0.017_0.041_5_1_0_92	max iter.	143	95.67	+49.42%
Bimanual	0.033_0.043_1_2_0_82	max iter.	125.4	95.64	+31.12%
Bimanual	0.036_0.046_5_3_2_45	max iter.	116	86.65	+33.86%
Planar Pushing	triangle_0.015_19_2	max iter.	93.1	111.3 (err.)	–
Planar Pushing	box_0.02_14_45	opt	74.57	84.28 (err.)	–
Planar Pushing	tee_0.015_27_43	opt	76.87	0 (err.)	–

Table C.17: Solution quality of half-step VEGA on edge-regularized GcsBench programs with $\epsilon = 10$.