THE ALTERNATING DIRECTION METHOD OF MULTIPLIERS FOR MIXED-INTEGER OPTIMIZATION APPLICATIONS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Reza Takapoui
June 2017

This dissertation is online at: http://purl.stanford.edu/hw534wk5693

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Stephen Boyd, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

John Duchi

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Sanjay Lall

Approved for the Stanford University Committee on Graduate Studies.

Patricia J. Gumport, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.

Abstract

A very large number of practical optimization problems have been expressed as minimization of a convex objective function over a nonconvex set, specifically discrete sets such as the set of integer points. These problems arise in a variety of fields such as statistics and modeling, data analysis, and control. Two general approaches have been used for solving mixed integer optimization problems. One approach is to solve the problem globally using methods such as branch-and-bound, which suffer from exponential worst case time complexity. Another approach is to use heuristics such as semidefinite relaxation hierarchies which terminate in polynomial time, but do not guarantee finding the global solution.

In this dissertation, we discuss a generic system of heuristic solutions for such problems. We describe an implementation of these methods in a package called NCVX, as an extension of CVXPY, a Python package for formulating and solving convex optimization problems. Our heuristics, which employ convex relaxations, convex restrictions, local neighbor search methods, and the alternating direction method of multipliers (ADMM), require the solution of a modest number of convex problems, and are meant to apply to general problems, without much tuning. Several numerical examples such as regressor selection, circle packing, the traveling salesman problem, and factor analysis modeling are studied.

We also describe a fast optimization algorithm for approximately minimizing convex quadratic functions over the intersection of affine and separable constraints. We discuss the favorable computational aspects of our algorithm, which allow it to run quickly even on very modest computational platforms such as embedded processors. We study numerical examples including hybrid vehicle control and power converter

control, and we compare our methods with existing open source and commercial alternatives.

Finally we discuss how randomized linear programming heuristics can be used to solve graph isomorphism problems. We motivate our heuristics by showing guarantees under some conditions, and present numerical experiments that show effectiveness of these heuristics in the general case.

Acknowledgments

I believe I wouldn't be writing these paragraphs today if it weren't for the privilege of having many amazing people in my life. Being born and raised in a culture that promotes education, I have constantly been encouraged to pursue higher education, and I feel indebted to countless individuals who always reminded me of the importance in acquiring knowledge and new skills in life.

Over the course of my studies at Stanford University, I was fortunate to interact with many exceptional individuals, who helped facilitate my personal development. If I wrote about all of these individuals, this section would be longer than the body of the dissertation. However, I will try to simply name a few that had a major influence in my graduate school life.

Advisor. Working with Stephen Boyd was an incredible experience, and was better than anything I could possibly wish for. He is an outstanding scholar who fosters an environment for his students to flourish, while challenging them intellectually, and inspiring them after every single meeting (which could be a short chat in the hallway). Most importantly, he is a person of upstanding morals and ethics, and has consistently been a supportive advisor through hardships and difficulties.

I have enjoyed his philosophy of *'keep things simple'*, and thinking about problems from a fundamental point of view, then representing them rigorously yet simply. Other than academic supervision, I am extremely thankful to Stephen for showing me excellence in style. As much as my potentials allow, I have tried to learn from him regarding writing, presentation, planning, and timing. I admire Stephen's passion to make educational material accessible for everyone, and also his contributions to open

source projects.

Orals committee. I'd like to express my sincere gratitude to Sanjay Lall, John Duchi, Ashok Srivastava, and Amin Saberi for their thoughtful advice and valuable feedback to my dissertation. I have learned a great deal from Sanjay and Amin through classes, and I had an amazing learning opportunity as a teaching assistant for John and Sanjay.

Labmates. I am thankful to Madeleine Udell, Nicholas Moehle, and Steven Diamond for all they taught me through our collaborations. My gratitude extends to AJ Friend, Thomas Lipp, Ernest Ryu, Corinne Horn, Jaehyun Park, Baris Ungun, Enzo Busseti, Bill Sun, and Jonathan Tuck.

Administrative masterminds. These individuals have always supported me behind the scenes and helped me by answering questions quickly and planning logistics. I'm thankful to Douglas Chaffee, Andrea Kuduk, LaToya Powell, and Amy Duncan.

Persian Student Association (PSA) fellows. I have spent significant time with my PSA fellows, and have been fortunate to work and become friends with Koosha Nassiri, Yasaman Shirian, Ramtin Keramati, Kian Katan, and Khashayar Khosravi while serving on the PSA Board of Directors. My acknowledgement extends to everyone in the Persian Student Association.

Generous people of the United States of America. My sincere gratitude goes to everyone who contributed to supporting my studies financially, and also people of the US who have constantly welcomed scholars from different parts of the world, and celebrated diversity.

Friends. I can't emphasize enough how lucky I feel to have had such a great community and support system at Stanford. I'd like to thank my old and new friends specially Hossein Karkeh Abadi, Omid Javidbakht, Ali Makhdoumi, Hamidreza Hakim

Javadi, Hadi Pour Ansari, Holly Deremo, Alireza Tahmaseb, Hosna Akhlaghpour, and Delara Fadavi.

Family. Without any doubt, I feel most indebted to my family. Even though an ocean has been separating us for the past five years, I have never felt emotionally distant from them. I'd like to thank my sisters, Pegah and Gilda, and their families, who have supported me in all stages of my life. Most importantly, I must thank my parents. Any attempt to express my gratitude to my parents would be futile. It's hard for me to comprehend how much sacrifice a person can make to help someone else grow, and yet, I realize my parents never hesitated to trade their comfort to see my success. I have said it repeatedly and I state it here once again: I wouldn't be who I am without their constant support.

Contents

List of Figures

Chapter 1

Overview

This dissertation introduces a framework for finding approximate solutions to several families of nonconvex problems. The heuristics that make the foundation of this framework are based on the alternating direction method of multipliers (ADMM), combined with local search methods. Several numerical examples such as regressor selection, circle packing, the traveling salesman problem, factor analysis modeling, as well as practical examples from model predictive control, and graph isomorphism are studied.

While others have studied the theory behind ADMM in nonconvex setting (such as finding bounds, convergence guarantees, *etc.*) we focus on a number of practical problems and propose sophisticated heuristics to approximately solve those problems. Many of the problems that we examine are studied extensively by other researchers, and there has been several algorithms tailored to each of these examples. We do not attempt to compete with specialized solvers, instead we show that our general framework can yield acceptable solutions to a wide variety of nonconvex problems with minimal or no tuning.

In chapter 2, which is based on [69], we describe general heuristics to approximately solve a wide variety of problems with convex objective and decision variables from a nonconvex set. The heuristics, which employ convex relaxations, convex restrictions, local neighbor search methods, and the alternating direction method of multipliers, require the solution of a modest number of convex problems, and are

meant to apply to general problems, without much tuning. We describe an implementation of these methods in a package called NCVX, as an extension of CVXPY, a Python package for formulating and solving convex optimization problems. We study several examples of well known nonconvex problems, and show that our general purpose heuristics are effective in finding approximate solutions to a wide variety of problems.

In chapter 3, which is based on [219], we propose a fast optimization algorithm for approximately minimizing convex quadratic functions over the intersection of affine and separable constraints (*i.e.*, the Cartesian product of possibly nonconvex real sets). This problem class contains many NP-hard problems such as mixed-integer quadratic programming. Our heuristic is based on a variation of the alternating direction method of multipliers.We discuss the favorable computational aspects of our algorithm, which allow it to run quickly even on very modest computational platforms such as embedded processors. We give several examples for which an approximate solution should be found very quickly, such as management of a hybrid-electric vehicle drivetrain and control of switched-mode power converters. Our numerical experiments suggest that our method is very effective in finding a feasible point with small objective value; indeed, we see that in many cases, it finds the global solution.

In chapter 4, we introduce effective probabilistic linear programming (LP) heuristics to solve the well-studied graph isomorphism problem. To recapitulate, an isomorphism between two graphs is a bijection between their vertices that preserves the edges. We consider the problem of determining whether two finite undirected weighted graphs are isomorphic, and finding an isomorphism relating them if the answer is positive. We discuss an efficient ADMM-based algorithm to solve the linear programming problems that arise in practice. We motivate our heuristics by showing guarantees under some conditions, and present numerical experiments that show the effectiveness of these heuristics in the general case.

Chapter 2

A General System of Heuristics

2.1 Introduction

2.1.1 The problem

We consider the optimization problem

$$\begin{array}{ll} \text{minimize} & f_0(x,z) \\ \text{subject to} & f_i(x,z) \leq 0, \quad i = 1,\ldots,m \\ & Ax + Bz = c \\ & z \in \mathcal{X}, \end{array} \tag{2.1}$$

where $x \in \mathbf{R}^n$ and $z \in \mathbf{R}^q$ are the decision variables, $A \in \mathbf{R}^{p\times n}$, $B \in \mathbf{R}^{p\times q}$, $c \in \mathbf{R}^p$ are problem data, and $\mathcal{X} \subseteq \mathbf{R}^q$ is closed. We assume that the objective and inequality constraint functions $f_0,\ldots,f_m : \mathbf{R}^n \times \mathbf{R}^q \to \mathbf{R}$ are jointly convex in x and z. When the set $\mathcal{X}$ is convex, (2.1) is a convex optimization problem, but we are interested here in the case where $\mathcal{X}$ is not convex. Roughly speaking, the problem (2.1) is a convex optimization problem, with some additional nonconvex constraints, $z \in \mathcal{X}$. We can think of x as the collection of decision variables that appear only in convex constraints, and z as the decision variables that are directly constrained to lie in the (generally) nonconvex set $\mathcal{X}$. The set $\mathcal{X}$ is often a Cartesian product, $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_k$, where $\mathcal{X}_i \subset \mathbf{R}^{q_i}$ are sets that are simple to describe, *e.g.*, $\mathcal{X}_i = \{0,1\}$. We denote the

optimal value of the problem (2.1) as $p^\star$, with the usual conventions that $p^\star = +\infty$ if the problem is infeasible, and $p^\star = -\infty$ if the problem is unbounded below.

2.1.2 Special cases

Mixed integer convex optimization. When $\mathcal{X} = \{0,1\}^q$, the problem (2.1) is a general mixed integer convex program, *i.e.*, a convex optimization problem in which some variables are constrained to be Boolean. ('Mixed Boolean' would be a more accurate name for such a problem, but 'mixed integer' is commonly used.) It follows that the problem (2.1) is NP-hard; it includes as a special case, for example, the general Boolean satisfaction problem.

Cardinality constrained convex optimization. As another broad special case of (2.1), consider the case $\mathcal{X} = \{z \in \mathbf{R}^q \mid \mathbf{card}(z) \leq k,\ \|z\|_\infty \leq M\}$, where $\mathbf{card}(z)$ is the number of nonzero elements of z, and k and M are given. We call this the general *cardinality-constrained convex problem*. It arises in many interesting applications, such as regressor selection.

Other special cases. As we will see in §3.3, many (hard) problems can be formulated in the form (2.1). More examples include regressor selection, 3-SAT, circle packing, the traveling salesman problem, factor analysis modeling, inexact graph isomorphism, and many more.

2.1.3 Convex relaxation

Convex relaxation of a set. For bounded sets $\mathcal{X}$ there usually is a manageable full or partial description of the convex hull of $\mathcal{X}$. By this we mean a (modest-sized) set of convex inequality and linear equality constraints that hold for every $z \in \mathcal{X}$:

$$z \in \mathcal{X} \implies h_i(z) \leq 0, \quad i = 1, \ldots, s, \qquad Fz = g.$$

We will assume that these relaxation constraints are included in the convex constraints of (2.1). Adding these relaxation constraints to the original problem yields

an equivalent problem (since the added constraints are redundant), but can improve the convergence of any method, global or heuristic. By tractable, we mean that the number of added constraints is modest, and in particular, polynomial in q.

For example, when $\mathcal{X} = \{0,1\}^q$, we have the inequalities $0 \leq z_i \leq 1$, $i = 1, \ldots, q$. (These inequalities define the convex hull of $\mathcal{X}$, *i.e.*, all other convex inequalities that hold for all $z \in \mathcal{X}$ are implied by them.) When

$$\mathcal{X} = \{z \in \mathbf{R}^q \mid \mathbf{card}(z) \leq k,\ \|z\|_\infty \leq M\},$$

we have the convex inequalities

$$\|z\|_1 \leq kM, \qquad \|z\|_\infty \leq M.$$

(These inequalities define the convex hull of $\mathcal{X}$.) For general bounded $\mathcal{X}$ the inequality $\|z\|_\infty \leq M$ will always be a convex relaxation for some value of M.

Relaxed problem. If we remove the nonconvex constraint $z \in \mathcal{X}$, we get a *convex relaxation* of the original problem:

$$\begin{array}{ll} \text{minimize} & f_0(x,z) \\ \text{subject to} & f_i(x,z) \leq 0, \quad i = 1, \ldots, m \\ & Ax + Bz = c. \end{array} \tag{2.2}$$

(Recall that convex equalities and inequalities known to hold for $z \in \mathcal{X}$ have been incorporated in the convex constraints.) The relaxed problem is convex; its optimal value is a lower bound on the optimal value $p^\star$ of (2.1). A solution (x^*, z^*) to problem (2.2) need not satisfy $z^* \in \mathcal{X}$, but if it does, the pair (x^*, z^*) is optimal for (2.1).

2.1.4 Projections and approximate projections

Our methods will make use of tractable projection, or tractable approximate projection, onto the set $\mathcal{X}$. The usual Euclidean projection onto $\mathcal{X}$ will be denoted Π. (It

need not be unique when $\mathcal{X}$ is not convex.) By approximate projection, we mean any function $\hat{\Pi} : \mathbf{R}^q \to \mathcal{X}$ that satisfies $\hat{\Pi}(z) = z$ for $z \in \mathcal{X}$. A useful approximate projection $\hat{\Pi}(z)$ will also approximately minimize $\|u - z\|_2^2$ over $u \in \mathcal{X}$, but since all the algorithms we present are heuristics, we do not formalize this requirement. We use approximate projections when computing an exact projection onto the set $\mathcal{X}$ is too expensive.

For example, when $\mathcal{X} = \{0,1\}^q$, exact projection is given by rounding the entries to $\{0,1\}$. As a less trivial example, consider the cardinality-constrained problem. The projection of z onto $\mathcal{X}$ is given by

$$(\Pi(z))_i = \begin{cases} M & z_i > M,\ i \in \mathcal{I} \\ -M & z_i < -M,\ i \in \mathcal{I} \\ z_i & |z_i| \leq M,\ i \in \mathcal{I} \\ 0 & i \notin \mathcal{I}, \end{cases}$$

where $\mathcal{I} \subseteq \{1, \ldots, q\}$ is a set of indices of k largest values of $|z_i|$. We will describe many projections, and some approximate projections, in §2.4.

2.1.5 Residual and merit functions

For any (x, z) with $z \in \mathcal{X}$, we define the *constraint residual* as

$$r(x,z) = \sum_{i=1}^{m} (f_i(x,z))_+ + \|Ax + Bz - c\|_1,$$

where $(u)_+ = \max\{u, 0\}$ denotes the positive part; (x, z) is feasible if and only if $r(x, z) = 0$. Note that $r(x, z)$ is a convex function of (x, z). We define the *merit function* of a pair (x, z) as

$$\eta(x,z) = f_0(x,z) + \lambda r(x,z),$$

where $\lambda > 0$ is a parameter. The merit function is also a convex function of (x, z).

When $\mathcal{X}$ is convex and the problem is feasible, minimizing $\eta(x, z)$ for large enough

λ yields a solution of the original problem (2.1) (that is, the residual is a so-called exact penalty function); when the problem is not feasible, it tends to find approximate solutions that satisfy many of the constraints [107, 67, 80].

We will use the merit function to judge candidate approximate solutions (x, z) with $z \in \mathcal{X}$; that is, we take a pair with lower merit function value to be a better approximate solution than one with higher merit function value. For some problems (for example, unconstrained problems) it is easy to find feasible points, so all candidate points will be feasible. The merit function then reduces to the objective value. At the other extreme, for feasibility problems the objective is zero, and the goal is to find a feasible point. In this case the merit function reduces to $\lambda r(x, z)$, *i.e.*, a positive multiple of the residual function.

2.1.6 Solution methods

In this section we describe various methods for solving the problem (2.1), either exactly (globally) or approximately.

Global methods. Depending on the set $\mathcal{X}$, the problem (2.1) can be solved globally by a variety of algorithms, including (or mixing) branch-and-bound [146, 181, 37], branch-and-cut [189, 221, 216], semidefinite hierarchies [208], or even direct enumeration when $\mathcal{X}$ is a finite set. In each iteration of these methods, a convex optimization problem derived from (2.1) is solved, with $\mathcal{X}$ removed, and (possibly) additional variables and convex constraints added. While for many applications these methods are effective, they are generally thought to have high worst-case complexities and indeed can be very slow for some problems.

Local solution methods and heuristics. A local method for (2.1) solves a modest number of convex problems, in an attempt to find a good approximate solution, *i.e.*, a pair (x, z) with $z \in \mathcal{X}$ and a low value of the merit function $\eta(x, z)$. For a feasibility problem, we might hope to find a solution; and if not, find one with a small constraint residual. For a general problem, we can hope to find a feasible point with low objective value, ideally near the lower bound on $p^\star$ from the relaxed problem. If

we cannot find any feasible points, we can settle for a pair (x, z) with $z \in \mathcal{X}$ and low merit function value. All of these methods are heuristics, in the sense that they cannot in general be guaranteed to find an optimal, or even good, or even feasible, point in only a modest number of iterations.

There are of course many heuristics for the general problem (2.1) and for many of its special cases. For example, any global optimization method can be stopped after some modest number of iterations; we then take the best point found (in terms of the merit function) as our approximate solution. (We will discuss some local search methods, including neighbor search and polishing, in §2.2.)

2.1.7 Our approach

The purpose of this chapter is to describe a general system for heuristic solution of (2.1), based on solving a *modest* number of convex problems derived from (2.1). By heuristic, we mean that the algorithm need not find an optimal point, or indeed, even a feasible point, even when one exists. We would hope that for many feasible problem instances from some application, the algorithm does find a feasible point, and one with objective not too far from the optimal value. The disadvantage of a heuristic over a global method is clear and simple: it need not find an optimal point. The advantage of a heuristic is that it can be (and often is) dramatically faster to carry out than a global method. Moreover there are many applications where a heuristic method for (2.1) is sufficient because the difference between a globally optimal solution and a solution that is only close to optimal is not significant in practice.

ADMM. One of the heuristic methods described in this chapter is based on the alternating directions method of multipliers (ADMM), an operator splitting algorithm originally devised to solve convex optimization problems. (See *e.g.*, [34] and [74] for comprehensive tutorials on ADMM.) We call this heuristic nonconvex alternating directions method of multipliers (NC-ADMM). ADMM was introduced in the mid-1970s [100, 88]. More recently, ADMM has found applications in a variety of distributed settings in machine learning such as model fitting, resource allocation,

and classification. (See *e.g.*, [231, 206, 232, 245, 177, 203, 11].) The idea of using ADMM as a general purpose heuristic to solve nonconvex problems was mentioned in [34, Ch. 9] and was further explored in [66]. Consensus ADMM has been used for nonconvex quadratically constrained quadratic programs in [120]. In [242], ADMM has been applied to nonnegative matrix factorization with missing values. ADMM also has been used for real and complex polynomial optimization models in [129], for constrained tensor factorization in [155], and for optimal power flow in [76]: all nonconvex problems. ADMM can be viewed as a version of the method of multipliers [109, 197, 22], where a Gauss-Seidel pass over x and z is used instead of the usual joint minimization. There is a long history of using the method of multipliers to (attempt to) solve nonconvex problems [46, 47, 115, 117, 195, 234, 152]. Instead of basing our heuristic on ADMM, which is Douglas-Rachford splitting [72] applied to the dual problem, we could also have used Spingarn's method [214], which is Douglas-Rachford splitting applied directly to the primal problem. For nonconvex problems the two approaches could yield different results.

Our contribution. Our main contribution is to identify a small number of concepts and methods for heuristics for nonconvex problems that can be applied across a very wide variety of problems. The only essential one is a projection, or even just an approximate projection, onto the nonconvex sets that appear in the problem. The others, which can dramatically improve the performance of the heuristic, are to identify a convex relaxation for each nonconvex set, a convex restriction at a general point in each nonconvex set, and a method to identify or list neighbors of a given point (in a discrete nonconvex set). We have implemented a general purpose system that uses just these four methods and handles a variety of different problems. Our implementation is readily extensible; the user only needs to implement these four methods for any new nonconvex set to be added.

Outline. The chapter has the following structure. In §2.2 we discuss local search methods and describe how they can be used as solution improvement methods. This will enable us to study simple but sophisticated methods such as relax-round-polish

and iterative neighbor search. In §2.3 we present a heuristic for problem (2.1) based on ADMM, which makes use of the solution improvement methods in §2.2. In §2.4 we catalog a variety of nonconvex sets for which Euclidean projection or approximate projection is easily evaluated and, when applicable, we discuss relaxations, restrictions, and distance functions that define the set of neighbors for a given point. In §2.5 we discuss an implementation of our general system for heuristic solution NCVX, as an extension of CVXPY [68], a Python package for formulating and solving convex optimization problems. The object-oriented features of CVXPY make the extension particularly simple to implement. Finally, in §3.3 we demonstrate the performance of our methods on several example problems.

2.2 Local improvement methods

In this section we describe some simple general local search methods. These methods take a point $z \in \mathcal{X}$ and by performing a local search on z they find a candidate pair $(\hat{x}, \hat{z})$, with $\hat{z} \in \mathcal{X}$ and a lower merit function. We will see that for many applications using these methods with a good initialization will result in an approximate solution. We will also see how we can use these methods to improve solution candidates from other heuristics, hence we refer to these methods as *solution improvement*.

2.2.1 Polishing

Convex restriction. For non-discrete $\mathcal{X}$, the idea of a *convex restriction* at a point is useful for local search methods. A convex restriction at a point $z \in \mathcal{X}$ is a convex set $\mathcal{S}^{\text{rstr}}(z)$ that satisfies $z \in \mathcal{S}^{\text{rstr}}(z) \subseteq \mathcal{X}$. The trivial restriction given by $\mathcal{S}^{\text{rstr}}(z) = \{z\}$ is valid for all $\mathcal{X}$. When $\mathcal{X}$ is discrete, for example $\mathcal{X} = \{0,1\}^q$, the trivial restriction is the only restriction. In other cases we can have interesting nontrivial restrictions. For example, with $\mathcal{X} = \{z \in \mathbf{R}^q \mid \mathbf{card}(z) \leq k,\ \|z\|_\infty \leq M\}$, we can take as restriction $\mathcal{S}^{\text{rstr}}(\tilde{z})$ the set of vectors z with the same sparsity pattern as $\tilde{z}$, and $\|z\|_\infty \leq M$.

Polishing. Given any point $\tilde{z} \in \mathcal{X}$, we can replace the constraint $z \in \mathcal{X}$ with $z \in \mathcal{S}^{\text{rstr}}(\tilde{z})$ to get the convex problem

$$\begin{array}{ll} \text{minimize} & \eta(x, z) \\ \text{subject to} & z \in \mathcal{S}^{\text{rstr}}(\tilde{z}), \end{array} \tag{2.3}$$

with variables x, z. (When the restriction $\mathcal{S}^{\text{rstr}}(\tilde{z})$ is the trivial one, *i.e.*, a singleton, this is equivalent to fixing $z = \tilde{z}$ and minimizing over x.) We call this problem the *convex restriction* of (2.1) at the point $\tilde{z}$. The restricted problem is convex, and its optimal value is an upper bound on $p^\star$ assuming λ is sufficiently large in $\eta(x, z) = f_0(x, z) + \lambda r(x, z)$ to ensure $r(x, z) = 0$.

As a simple example of polishing consider the mixed integer convex problem. The only restriction is the trivial one, so the polishing problem for a given Boolean vector $\tilde{z}$ simply fixes the values of the Boolean variables, and solves the convex problem over the remaining variables, *i.e.*, x. For the cardinality-constrained convex problem, polishing fixes the sparsity pattern of z and solves the resulting convex problem over z and x.

For problems with nontrivial restrictions, we can solve the polishing problem repeatedly until convergence. In other words we can use the output of the polishing problem as an initial point for another polishing problem and keep iterating until the merit function stops improving. This technique is called *iterated polishing* and described in algorithm 1.

Algorithm 1 Iterated polishing

1: Input: $(\tilde{x}, \tilde{z})$
2: **do**
3: $\quad (x^{\text{old}}, z^{\text{old}}) \leftarrow (\tilde{x}, \tilde{z})$.
4: $\quad$ Find $(\tilde{x}, \tilde{z})$ by solving the polishing problem with restriction $z \in \mathcal{S}^{\text{rstr}}(z^{\text{old}})$.
5: **while** $\eta(\tilde{x}, \tilde{z}) < \eta(x^{\text{old}}, z^{\text{old}})$
6: **return** $(\tilde{x}, \tilde{z})$.

If there exists a point $\tilde{x}$ such that $(\tilde{x}, \tilde{z})$ is feasible, the restricted problem is feasible too. The restricted problem need not be feasible in general, but if it is, with solution $(\hat{x}, \hat{z})$, then the pair $(\hat{x}, \hat{z})$ is feasible for the original problem (2.1) and

satisfies $f_0(\hat{x}, \hat{z}) \leq f_0(\tilde{x}, \tilde{z})$ for any $\tilde{x}$ for which $(\tilde{x}, \tilde{z})$ is feasible. So polishing can take a point $\tilde{z} \in \mathcal{X}$ (or a pair $(\tilde{x}, \tilde{z})$) and produce another pair $(\hat{x}, \hat{z})$ with a possibly better objective value.

2.2.2 Relax-round-polish

With the simple tools described so far (*i.e.*, relaxation, polishing, and projection) we can create several heuristics for approximately solving the problem (2.1). A basic version solves the relaxation, projects the relaxed value of z onto $\mathcal{X}$, and then iteratively polishes the result.

Algorithm 2 Relax-round-polish heuristic

1: Solve the convex relaxation (2.2) to obtain $(x^{\mathrm{rlx}}, z^{\mathrm{rlx}})$.
2: $z^{\mathrm{rnd}} \leftarrow \Pi(z^{\mathrm{rlx}})$.
3: Use algorithm 1 on $(x^{\mathrm{rlx}}, z^{\mathrm{rnd}})$ to get $(\hat{x}, \hat{z})$.

Note that in the first step we also obtain a lower bound on the optimal value $p^\star$; in the polishing step we obtain an upper bound and a feasible pair $(\hat{x}, \hat{z})$ that achieves the upper bound provided that polishing is successful. The best outcome is for these bounds to be equal, which means that we have found a (global) solution of (2.1) (for this problem instance). But relax-round-polish can fail; for example, it can fail to find a feasible point even though one exists.

Many variations on relax-round-polish are possible. We can introduce randomization by replacing the round step with

$$z^{\mathrm{rnd}} = \Pi(z^{\mathrm{rlx}} + w),$$

where w is a random vector. We can repeat this heuristic with N different random instances of w. For each of N samples of w, we polish, giving us a set of N candidate approximate solutions. We then take as our final approximate solution the best among these N candidates, *i.e.*, the one with least merit function.

2.2.3 Neighbor search

Neighbors. When $\mathcal{X}$ is discrete, convex restrictions are not useful for local search. Instead we use the concept of *neighbors* of a point $z \in \mathcal{X}$ as a discrete analogue to a restriction. As with a restriction, we do local search over the set of neighbors. Neighbors are defined in terms of a distance function $\mathcal{X}^{\text{dist}} : \mathcal{X} \times \mathcal{X} \to \mathbf{Z}_+ \cup \{+\infty\}$. The set of neighbors of a point $z \in \mathcal{X}$ within distance $D \in \mathbf{Z}_+$, denoted $\mathcal{S}^{\text{ngbr}}(z, D)$, is given by $\mathcal{S}^{\text{ngbr}}(z, k) = \{\mathcal{X}^{\text{dist}}(y, z) \leq D \mid y \in C\}$. We select a distance function and distance D such that the size of $\mathcal{S}^{\text{ngbr}}(z, D)$ is computationally tractable for all $z \in \mathcal{X}$. For non-discrete $\mathcal{X}$, we use the trivial distance function

$$\mathcal{X}^{\text{dist}}(z, y) = \begin{cases} 0 & z = y \\ +\infty & z \neq y, \end{cases}$$

for which $\mathcal{S}^{\text{ngbr}}(z, D) = \{z\}$ for all z and D.

For example, for the set of Boolean vectors in $\mathbf{R}^n$ we use *Hamming distance*, the number of entries in which two Boolean vectors differ. Hence the neighbors of a Boolean vector z within distance D are the set of vectors that differ from z in up to D components. We define the distance between two permutation matrices as the minimum number of swaps of adjacent rows and columns necessary to transform the first matrix into the second. With this distance, neighbors of a permutation matrix Z within distance D are the set of permutation matrices generated by swapping any two adjacent rows or columns in Z up to D times. We define distance in terms of swaps of adjacent rows and columns rather than swaps of arbitrary rows and columns to reduce the number of neighbors.

For Cartesian products of discrete sets we use the sum of distances. In this case, for $z = (z_1, z_2, \ldots, z_q) \in \mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \ldots \times \mathcal{X}_q$, neighbors of z within distance D are points of the form $(\tilde{z}_1, \tilde{z}_2, \ldots, \tilde{z}_q)$ where $\sum_{i=1}^{q} \mathcal{X}_i^{\text{dist}}(\tilde{z}_i, z_i) \leq D$.

Basic neighbor search. We introduced polishing as a tool that can find a pair $(\hat{x}, \hat{z})$ given an input $\tilde{z} \in \mathcal{X}$ by solving a sequence of convex problems. In basic neighbor search we solve the polishing problem for $\tilde{z}$ and all neighbors of $\tilde{z}$ (within

distance D) and return the pair (x^*, z^*) with the smallest merit function value. In practice, we can sample from $\mathcal{S}^{\text{ngbr}}(\tilde{z}, D)$ instead of iterating over all points in $\mathcal{S}^{\text{ngbr}}(\tilde{z})$ if $\left|\mathcal{S}^{\text{ngbr}}(\tilde{z}, D)\right|$ is large.

Algorithm 3 Basic neighbor search

Input: $\tilde{z}$
Initialize $(x_{\text{best}}, z_{\text{best}}) = \emptyset$, $\eta_{\text{best}} = \infty$.
for $\hat{z} \in \mathcal{S}^{\text{ngbr}}(\tilde{z}, D)$ **do**
    Find (x^*, z^*), by solving the polishing problem (2.3), with constraint $z \in \mathcal{S}^{\text{rstr}}(\hat{z})$.
    if $\eta(x^*, z^*) < \eta_{\text{best}}$ **then**
        $(x_{\text{best}}, z_{\text{best}}) \leftarrow (x^*, z^*)$, $\eta_{\text{best}} \leftarrow \eta(x^*, z^*)$.
    end if
end for
return $(x_{\text{best}}, z_{\text{best}})$.

Iterated neighbor search. As with polishing, we can apply basic neighbor search repeatedly until convergence. In other words we can feed the output of algorithm 3 back into algorithm 3 until the merit function stops improving. The technique is called *iterated neighbor search* and described in algorithm 4. Notice that for non-discrete sets where $\mathcal{S}^{\text{ngbr}}(z, D) = \{z\}$ for all z and D, this algorithm reduces to iterated polishing.

Algorithm 4 Iterated neighbor search

Input: $(\tilde{x}, \tilde{z})$
do
    $(x^{\text{old}}, z^{\text{old}}) \leftarrow (\tilde{x}, \tilde{z})$.
    Use algorithm 3 on z^{old} to get $(\tilde{x}, \tilde{z})$.
while $\eta(\tilde{x}, \tilde{z}) < \eta(x^{\text{old}}, z^{\text{old}})$
return $(\tilde{x}, \tilde{z})$.

2.3 NC-ADMM

We already can use the simple tools described in the previous section as heuristics to find approximate solutions to problem (2.1). In this section, we describe the

alternating direction method of multipliers (ADMM) as a mechanism to generate candidate points $\tilde{z}$ to carry out local search methods such as iterated neighbor search. We call this method nonconvex ADMM, or NC-ADMM.

2.3.1 ADMM

Define $\phi : \mathbf{R}^q \to \mathbf{R} \cup \{-\infty, +\infty\}$ such that $\phi(z)$ is the best objective value of problem (2.1) after fixing z. In other words,

$$\phi(z) = \inf_x \{f_0(x, z) \mid f_i(x, z) \leq 0,\ i = 1, \ldots, m,\ Ax + Bz = c\}.$$

Notice that $\phi(z)$ can be $+\infty$ or $-\infty$ in case the problem is not feasible for this particular value of z, or problem (2.2) is unbounded below after fixing z. The function ϕ is convex, since it is the partial minimization of a convex function over a convex set [35, §3.4.4]. It is defined over all points $z \in \mathbf{R}^q$, but we are interested in finding its minimum value over the nonconvex set $\mathcal{X}$. In other words, problem (2.1) can be formulated as

$$\begin{array}{ll} \text{minimize} & \phi(z) \\ \text{subject to} & z \in \mathcal{X}. \end{array} \tag{2.4}$$

As discussed in [34, Chapter 9], ADMM can be used as a heuristic to solve nonconvex constrained problems. ADMM has the form

$$\begin{aligned} w^{k+1} &:= \text{argmin}_z \left(\phi(z) + (\rho/2)\|z - z^k + u^k\|_2^2\right) \\ z^{k+1} &:= \Pi\left(w^{k+1} + u^k\right) \\ u^{k+1} &:= u^k + w^{k+1} - z^{k+1}, \end{aligned} \tag{2.5}$$

where $\rho > 0$ is an algorithm parameter, k is the iteration counter, and Π denotes Euclidean projection onto $\mathcal{X}$ (which need not be unique when $\mathcal{X}$ is not convex).

The initial values u^0 and z^0 are additional algorithm parameters. We always set $u^0 = 0$ and draw z^0 randomly from a normal distribution $\mathcal{N}(0, \sigma^2 I)$, where $\sigma > 0$ is an algorithm parameter.

2.3.2 Algorithm subroutines

Convex proximal step. Carrying out the first step of the algorithm, *i.e.*, evaluating the proximal operator of ϕ, involves solving the convex optimization problem

$$\begin{array}{ll} \text{minimize} & f_0(x,z) + (\rho/2)\|z - z^k + u^k\|_2^2 \\ \text{subject to} & f_i(x,z) \leq 0, \quad i = 1,\ldots,m, \\ & Ax + Bz = c, \end{array} \tag{2.6}$$

over the variables $x \in \mathbf{R}^n$ and $z \in \mathbf{R}^q$. This is the original problem (2.1), with the nonconvex constraint $z \in \mathcal{X}$ removed, and an additional convex quadratic term involving z added to the objective. We let (x^{k+1}, w^{k+1}) denote a solution of (2.6). If the problem (2.6) is infeasible, then so is the original problem (2.1); should this happen, we can terminate the algorithm with the certain conclusion that (2.1) is infeasible.

Projection. The (nonconvex) projection step consists of finding a closest point in $\mathcal{X}$ to $w^{k+1} + u^k$. If more than one point has the smallest distance, we can choose one of the minimizers arbitrarily. In cases where the projection onto $\mathcal{X}$ is too costly, we replace projection with approximate projection.

Dual update. The iterate $u^k \in \mathbf{R}^q$ can be interpreted as a scaled dual variable, or as the running sum of the error values $w^k - z^k$.

2.3.3 Discussion

Convergence. When $\mathcal{X}$ is convex (and a solution of (2.1) exists), this algorithm is guaranteed to converge to a solution, in the sense that $f_0(x^{k+1}, w^{k+1})$ converges to the optimal value of the problem (2.1), and $w^{k+1} - z^{k+1} \to 0$, *i.e.*, $w^{k+1} \to \mathcal{X}$. See [34, §3] and the references therein for a more technical description and details. But in the general case, when $\mathcal{X}$ is not convex, the algorithm is not guaranteed to converge, and even when it does, it need not be to a global, or even local, minimum. Some recent

progress has been made on understanding convergence of ADMM in the nonconvex case [152].

Parameters. Another difference with the convex case is that the convergence and the quality of solution depends on ρ, whereas for convex problems this algorithm is guaranteed to converge to the optimal value regardless of the choice of ρ. In other words, in the convex case the choice of parameter ρ only affects the speed of the convergence, while in the nonconvex case the choice of ρ can have a critical role in the quality of approximate solution, as well as the speed of convergence.

The optimal parameter selection for ADMM is still an active research area in the convex case; even less is known about it in the nonconvex case. In [93] the optimal parameter selection for convex quadratic problems is discussed. In a more general setting, Giselsson discusses the optimal parameter selection for ADMM for strongly convex functions in [95, 96, 97]. The dependence of global and local convergence properties of ADMM on parameter choice has been studied in [116, 31].

Initialization. In the convex case the choice of initial point z^0 affects the number of iterations to find a solution, but not the quality of the solution. Unsurprisingly, the nonconvex case differs in that the choice of z^0 has a major effect on the quality of the approximate solution. As with the choice of ρ, the initialization in the nonconvex case is currently an active area of research; see, *e.g.*, [120, 152, 219]. A reasonable generic method is to draw initial points randomly from $\mathcal{N}(0, \sigma^2 I)$ (assuming reasonable scaling of the original problem).

2.3.4 Solution improvement

Now we describe two techniques to obtain better solutions after carrying out ADMM. The first technique relies on iterated neighbor search and the second one uses multiple restarts with random initial points in order to increase the chance of obtaining a better solution.

Iterated neighbor search. After each iteration, we can carry out iterated neighbor search (as described in §2.2.3) with $\mathcal{S}^{\text{ngbr}}(z^{k+1}, D)$ to obtain $(\hat{x}^{k+1}, \hat{z}^{k+1})$. We will return the pair with the smallest merit function as the output of the algorithm. The distance D is a parameter that can be increased so that the neighbor search considers more points.

Multiple restarts. We choose the initial value z^0 from a normal distribution $\mathcal{N}(0, \sigma^2 I)$. We can run the algorithm multiple times from different initial points to increase the chance of finding a feasible point with a smaller objective value.

2.3.5 Overall algorithm

The following is a summary of the algorithm with solution improvement.

Algorithm 5 NC-ADMM heuristic

Initialize $u^0 = 0$, $(x_{\text{best}}, z_{\text{best}}) = \emptyset$, $\eta_{\text{best}} = \infty$.
for algorithm repeats $1, 2, \ldots, M$ **do**
  Initialize $z^0 \sim \mathcal{N}(0, \sigma^2 I)$, $k = 0$.
  do
    $(x^{k+1}, w^{k+1}) \leftarrow \operatorname{argmin}_z \left(\phi(z) + (\rho/2)\|z - z^k + u^k\|_2^2\right)$.
    $z^{k+1} \leftarrow \Pi\left(w^{k+1} + u^k\right)$.
    $u^{k+1} \leftarrow u^k + w^{k+1} - z^{k+1}$.
    Use algorithm 4 on (x^{k+1}, z^{k+1}) to get the improved iterate $(\hat{x}, \hat{z})$.
    if $\eta(\hat{x}, \hat{z}) < \eta_{\text{best}}$ **then**
      $(x_{\text{best}}, z_{\text{best}}) \leftarrow (\hat{x}, \hat{z})$, $\eta_{\text{best}} = \eta(\hat{x}, \hat{z})$.
    end if
    $k \leftarrow k + 1$.
  while $k \leq N$ and $(\hat{x}, \hat{z})$ has not repeated P times in a row.
end for
return $x_{\text{best}}, z_{\text{best}}$.

Convergence, stopping criteria, and optimality. As described in §2.3.3, ADMM need not converge for arbitrary nonconvex $\mathcal{X}$. The output of our heuristic is not the direct output of ADMM, however, but the output of ADMM after local search. In

algorithm 5, ADMM may be viewed as a procedure for generating sample points, which we run through algorithm 4 to get different local optima. Our heuristic may therefore be useful even on problems where ADMM fails to converge. We terminate algorithm 5 when local search returns the same point P times in a row, where P is a parameter. Given the lack of convergence guarantees for ADMM with nonconvex $\mathcal{X}$, the only formal notion of optimality provided by our heuristic is that the solution is optimal among all points considered by the local search method.

2.4 Projections onto nonconvex sets

In this section we catalog various nonconvex sets with their implied convex constraints, which will be included in the convex constraints of problem (2.1). We also provide a Euclidean projection (or approximate projection) Π for these sets. Also, when applicable, we introduce a nontrivial restriction and distance function.

2.4.1 Subsets of R

Booleans. For $\mathcal{X} = \{0, 1\}$, a convex relaxation (in fact, the convex hull of $\mathcal{X}$) is $[0, 1]$. Projection is simple rounding: $\Pi(z) = 0$ for $z \leq 1/2$, and $\Pi(z) = 1$ for $z > 1/2$. ($z = 1/2$ can be mapped to either point.) Moreover, $\mathcal{X}^{\text{dist}}(y, z) = |y - z|$ for $y, z \in \mathcal{X}$.

Finite sets. If $\mathcal{X}$ has M elements, the convex hull of $\mathcal{X}$ is the interval from the smallest to the largest element. We can project onto $\mathcal{X}$ with no more than $\log_2 M$ comparisons. For $y, z \in \mathcal{X}$, the distance function is given by $\mathcal{X}^{\text{dist}}(y, z) = |[y, z] \cap \mathcal{X}| - 1$.

2.4.2 Subsets of $\mathbf{R}^n$

Boolean vectors with fixed cardinality. Let $\mathcal{X} = \{z \in \{0, 1\}^n \mid \mathbf{card}(z) = k\}$. Any $z \in \mathcal{X}$ satisfies $0 \leq z \leq 1$ and $\mathbf{1}^T z = k$. We can project $z \in \mathbf{R}^n$ onto $\mathcal{X}$ by setting the k entries of z with largest value to one and the remaining entries to zero.

For $y, z \in \mathcal{X}$, the distance $\mathcal{X}^{\text{dist}}(y, z)$ is defined as the minimum number of swaps of entries needed to transform y into z, or half the Hamming distance.

Vectors with bounded cardinality. Let $\mathcal{X} = \{x \in [-M, M]^n \mid \mathbf{card}(x) \leq k\}$, where $M > 0$ and $k \in \mathbf{Z}_+$. (Vectors $z \in \mathcal{X}$ are called k-sparse.) Any point $z \in \mathcal{X}$ satisfies $-M \leq z \leq M$ and $-Mk \leq \mathbf{1}^T z \leq Mk$. The projection $\Pi(z)$ is found as follows

$$(\Pi(z))_i = \begin{cases} M & z_i > M,\ i \in \mathcal{I} \\ -M & z_i < -M,\ i \in \mathcal{I} \\ z_i & |z_i| \leq M,\ i \in \mathcal{I} \\ 0 & i \notin \mathcal{I}, \end{cases}$$

where $\mathcal{I} \subseteq \{1, \ldots, n\}$ is a set of indices of k largest values of $|z_i|$. A restriction of $\mathcal{X}$ at $z \in \mathcal{X}$ is the set of all points in $[-M, M]^n$ that have the same sparsity pattern as z.

Quadratic sets. Let $\mathbf{S}^n_+$ and $\mathbf{S}^n_{++}$ denote the set of $n \times n$ symmetric positive semidefinite and symmetric positive definite matrices, respectively. Consider the set

$$\mathcal{X} = \{z \in \mathbf{R}^n \mid \alpha \leq z^T A z + 2b^T z \leq \beta\},$$

where $A \in \mathbf{S}^n_{++}$, $b \in \mathbf{R}^n$, and $\beta \geq \alpha \geq -b^T A^{-1} b$. We assume $\alpha \geq -b^T A^{-1} b$ because $z^T A z + 2b^T z \geq -b^T A^{-1} b$ for all $z \in \mathbf{R}^n$. Any point $z \in \mathcal{X}$ satisfies the convex inequality $z^T A z + 2b^T z \leq \beta$.

We can find the projection onto $\mathcal{X}$ as follows. If $z^T A z + 2b^T z > \beta$, it suffices to solve

$$\begin{array}{ll} \text{minimize} & \|x - z\|_2^2 \\ \text{subject to} & x^T A x + 2b^T x \leq \beta, \end{array} \tag{2.7}$$

and if $z^T A z + 2b^T z < \alpha$, it suffices to solve

$$\begin{array}{ll} \text{minimize} & \|x - z\|_2^2 \\ \text{subject to} & x^T A x + 2b^T x \geq \alpha. \end{array} \tag{2.8}$$

(If $\alpha \leq z^T Az + 2b^T z \leq \beta$, clearly $\Pi(z) = z$.) The first problem is a convex quadratically constrained quadratic program and the second problem can be solved by solving a simple semidefinite program as described in [35, Appendix B]. Furthermore, there is a more efficient way to find the projection by finding the roots of a single-variable polynomial of degree $2p + 1$, where p is the number of distinct eigenvalues of A [120, 111]. Note that the projection can be easily found even if A is not positive definite; we assume $A \in \mathbf{S}^n_{++}$ only to make $\mathcal{X}$ bounded and have a useful convex relaxation.

A restriction of $\mathcal{X}$ at $z \in \mathcal{X}$ is the set

$$\mathcal{S}^{\mathrm{rstr}}(z) = \{x \in \mathbf{R}^n \mid \frac{x^T Az + b^T(x+z) + b^T A^{-1} b}{\sqrt{z^T Az + 2b^T z + b^T A^{-1} b}} \geq \sqrt{\alpha + b^T A^{-1} b}, \quad x^T Ax + 2b^T x \leq \beta\}.$$

Recall that $z^T Az + 2b^T z + b^T A^{-1} b \geq 0$ for all $z \in \mathbf{R}^n$ and we assume $\alpha \geq -b^T A^{-1} b$, so $\mathcal{S}^{\mathrm{rstr}}(z)$ is always well defined.

Annulus and sphere. Consider the set

$$\mathcal{C} = \{z \in \mathbf{R}^n \mid r \leq \|z\|_2 \leq R\},$$

where $R \geq r$.

Any point $z \in \mathcal{X}$ satisfies $\|z\|_2 \leq R$. We can project $z \in \mathbf{R}^n \setminus \{0\}$ onto $\mathcal{X}$ by the following scaling

$$\Pi(z) = \begin{cases} rz/\|z\|_2 & \text{if } \|z\|_2 < r \\ z & \text{if } z \in \mathcal{X} \\ Rz/\|z\|_2 & \text{if } \|z\|_2 > R. \end{cases}$$

If $z = 0$, any point with Euclidean norm r is a valid projection.

A restriction of $\mathcal{X}$ at $z \in \mathcal{X}$ is the set

$$\mathcal{S}^{\mathrm{rstr}}(z) = \{x \in \mathbf{R}^n \mid x^T z \geq r\|z\|_2, \ \|x\|_2 \leq R\}.$$

Notice that if $r = R$, then $\mathcal{X}$ is a sphere and the restriction will be a singleton.

2.4.3 Subsets of $\mathbf{R}^{m\times n}$

Remember that the projection of a point $X \in \mathbf{R}^{m\times n}$ on a set $\mathcal{X} \subset \mathbf{R}^{m\times n}$ is a point $Z \in \mathcal{X}$ such that the Frobenius norm $\|X - Z\|_{\mathrm{F}}$ is minimized. As always, if there is more than one point Z that minimizes $\|X - Z\|_{\mathrm{F}}$, we accept any of them.

Matrices with bounded singular values and orthogonal matrices. Consider the set of $m \times n$ matrices whose singular values lie between 1 and α

$$\mathcal{X} = \{Z \in \mathbf{R}^{m\times n} \mid I \preceq Z^T Z \preceq \alpha^2 I\},$$

where $\alpha \geq 1$, and $A \preceq B$ means $B - A \in \mathbf{S}^n_+$. Any point $Z \in \mathcal{X}$ satisfies $\|Z\|_2 \leq \alpha$.

If $Z = U\Sigma V^T$ is the singular value decomposition of Z with singular values $(\sigma_z)_{\min\{m,n\}} \leq \cdots \leq (\sigma_z)_1$ and $X \in \mathcal{X}$ with singular values $(\sigma_x)_{\min\{m,n\}} \leq \cdots \leq (\sigma_x)_1$, according to the von Neumann trace inequality [182] we will have

$$\mathbf{Tr}(Z^T X) \leq \sum_{i=1}^{\min\{m,n\}} (\sigma_z)_i (\sigma_x)_i.$$

Hence

$$\|Z - X\|_F^2 \geq \sum_{i=1}^{\min\{m,n\}} \left((\sigma_z)_i - (\sigma_x)_i\right)^2,$$

with equality when $X = U\,\mathbf{diag}(\sigma_x)V^T$. This inequality implies that $\Pi(Z) = U\tilde{\Sigma}V^T$, where $\tilde{\Sigma}$ is a diagonal matrix and $\tilde{\Sigma}_{ii}$ is the projection of Σ_{ii} on the interval $[1, \alpha]$. When $Z = 0$, the projection $\Pi(Z)$ is any matrix.

Given $Z = U\Sigma V^T \in \mathcal{X}$, we have the following restriction [33]

$$\mathcal{S}^{\mathrm{rstr}}(Z) = \{X \in \mathbf{R}^{m\times n} \mid \|X\|_2 \leq \alpha,\ V^T X^T U + U^T X V \succeq 2I\}.$$

(Notice that $X \in \mathcal{S}^{\mathrm{rstr}}(Z)$ satisfies $X^T X \succeq I + (X - UV^T)^T(X - UV^T) \succeq I$.)

There are several noteworthy special cases. When $\alpha = 1$ and $m = n$ we have the set of orthogonal matrices. In this case, the restriction will be a singleton. When $n = 1$, the set $\mathcal{X}$ is equivalent to the annulus $\{z \in \mathbf{R}^m \mid 1 \leq \|z\|_2 \leq \alpha\}$.

Matrices with bounded rank. Let $\mathcal{X} = \{Z \in \mathbf{R}^{m\times n} \mid \mathbf{Rank}(Z) \leq k,\ \|Z\|_2 \leq M\}$. Any point $Z \in \mathcal{X}$ satisfies $\|Z\|_2 \leq M$ and $\|Z\|_* \leq Mk$, where $\|\cdot\|_*$ denotes the trace norm. If $Z = U\Sigma V^T$ is the singular value decomposition of Z, we will have $\Pi(Z) = U\tilde{\Sigma}V^T$, where $\tilde{\Sigma}$ is a diagonal matrix with $\tilde{\Sigma}_{ii} = \min\{\Sigma_{ii}, M\}$ for $i = 1, \ldots k$, and $\tilde{\Sigma}_{ii} = 0$ otherwise.

Given a point $Z \in \mathcal{X}$, we can write the singular value decomposition of Z as $Z = U\Sigma V^T$ with $U \in \mathbf{R}^{m\times k}$, $\Sigma \in \mathbf{R}^{r\times r}$ and $V \in \mathbf{R}^{n\times k}$. A restriction of $\mathcal{X}$ at Z is

$$\mathcal{S}^{\text{rstr}}(Z) = \{U\tilde{\Sigma}V^T \mid \tilde{\Sigma} \in \mathbf{R}^{r\times r}, \|\tilde{\Sigma}\|_2 \leq M\}.$$

Assignment and permutation matrices. *Assignment matrices* are Boolean matrices with exactly one nonzero element in each column and at most one nonzero element in each row. (They represent an assignment of the columns to the rows.) In other words, the set of assignment matrices on $\{0,1\}^{m\times n}$, where $m \geq n$, satisfy

$$\begin{array}{ll} \sum_{j=1}^{n} Z_{ij} \leq 1, & i = 1, \ldots, m \\ \sum_{i=1}^{m} Z_{ij} = 1, & j = 1, \ldots, n. \end{array}$$

These two sets of inequalities, along with $0 \leq Z_{ij} \leq 1$ are the implied convex inequalities. When $m = n$, this set becomes the set of permutation matrices, which we denote by $\mathcal{P}_n$.

Projecting $Z \in \mathbf{R}^{m\times n}$ (with $m \geq n$) onto the set of assignment matrices involves choosing an entry from each column of Z such that no two chosen entries are from the same row and the sum of chosen entries is maximized. Assuming that the entries of Z are the weights of edges in a bipartite graph, the projection onto the set of assignment matrices will be equivalent to finding a maximum-weight matching in a bipartite graph. The Hungarian method [143] is a well-known polynomial time algorithm to find the maximum weight matching, and hence also the projection onto assignment matrices.

For $Y, Z \in \mathcal{X}$, the distance $\mathcal{X}^{\text{dist}}(Y, Z)$ is defined as the minimum number of swaps of adjacent rows and columns necessary to transform Y into Z. We define distance in terms of swaps of adjacent rows and columns rather than arbitrary rows and columns

to reduce the number of neighbors. For example, the restriction that swaps must be of adjacent rows and columns reduces $|\mathcal{S}^{\text{ngbr}}(Z,1)|$ from $O(mn)$ to $O(m+n)$ for $Z \in \mathcal{X}$.

Hamiltonian cycles. A *Hamiltonian cycle* is a cycle in a graph that visits every node exactly once. Every Hamiltonian cycle in a complete graph can be represented by its adjacency matrix, for example

$$\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

represents a Hamiltonian cycle that visits nodes $(3,2,4,1)$ sequentially. Let $\mathcal{H}_n$ be the set of $n \times n$ matrices that represent a Hamiltonian cycle.

Every point $Z \in \mathcal{H}_n$ satisfies $0 \leq Z_{ij} \leq 1$ for $i,j = 1,\ldots,n$, and $Z = Z^T$, $(1/2)Z\mathbf{1} = \mathbf{1}$, and

$$2\mathbf{I} - Z + 4\frac{\mathbf{1}\mathbf{1}^T}{n} \succeq 2\left(1 - \cos\frac{2\pi}{n}\right)\mathbf{I},$$

where $\mathbf{I}$ denotes the identity matrix. In order to see why the last inequality holds, it is enough to note that $2\mathbf{I} - Z$ is the Laplacian of the cycle represented by Z [169, 6]. It can be shown that the smallest eigenvalue of $2\mathbf{I} - Z$ is zero (which corresponds to the eigenvector $\mathbf{1}$), and the second smallest eigenvalue of $2\mathbf{I} - Z$ is $2(1 - \cos\frac{2\pi}{n})$. Hence all eigenvalues of $2\mathbf{I} - Z + 4\frac{\mathbf{1}\mathbf{1}^T}{n}$ must be no smaller than $2(1 - \cos\frac{2\pi}{n})$.

We are not aware of a polynomial time algorithm to find the projection of a given real $n \times n$ matrix onto $\mathcal{H}_n$. We can find an *approximate projection* of Z by the following greedy algorithm: construct a graph with n vertices where the edge between i and j is weighted by z_{ij}. Start with the edge with largest weight and at each step, among all the edges that do not create a cycle, choose the edge with the largest weight (except for the last step where a cycle is created).

For $Y, Z \in \mathcal{H}_n$, the distance $\mathcal{X}^{\text{dist}}(Y,Z)$ is defined as the minimum number of adjacent nodes that must be swapped to transform Y into Z. Swapping adjacent

nodes i and j means replacing Y with $P_{(i,j)}YP_{(i,j)}^T$ where $Y_{ij} = 1$ and $P_{(i,j)}$ is a permutation matrix that swaps nodes i and j and leaves other nodes unchanged. As with assignment matrices, we define distance in terms of swaps of adjacent nodes rather than arbitrary nodes to reduce the number of neighbors.

2.4.4 Combinations of sets

Cartesian product. Let $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_k \subset \mathbf{R}^n$, where $\mathcal{X}_1, \ldots, \mathcal{X}_k$ are closed sets with known projections (or approximate projections). A convex relaxation of $\mathcal{X}$ is the Cartesian product $\mathcal{S}_1^{\text{rlx}} \times \cdots \times \mathcal{S}_k^{\text{rlx}}$, where $\mathcal{S}_i^{\text{rlx}}$ is the set described by the convex relaxation of $\mathcal{X}_i$. The projection of $z \in \mathbf{R}^n$ onto $\mathcal{X}$ is $(\Pi_1(z_1), \ldots, \Pi_k(z_k))$, where Π_i denotes the projection onto $\mathcal{X}_i$ for $i = 1, \ldots, k$.

A restriction of $\mathcal{X}$ at a point $z = (z_1, z_2, \ldots, z_k) \in \mathcal{X}$ is the Cartesian product $\mathcal{S}^{\text{rstr}}(z) = \mathcal{S}_1^{\text{rstr}}(z_1) \times \cdots \times \mathcal{S}_k^{\text{rstr}}(z_k)$. For $y = (y_1, y_2, \ldots, y_k) \in \mathcal{X}$ and $z = (z_1, z_2, \ldots, z_k) \in \mathcal{X}$, the distance function is given by $\mathcal{X}^{\text{dist}}(y, z) = \sum_{i=1}^k \mathcal{X}_i^{\text{dist}}(y_i, z_i)$.

2.5 Implementation

We have implemented the NCVX Python package for modeling problems of the form (2.1) and applying the NC-ADMM heuristic, along with the relax-round-polish and relax methods. The NCVX package is an extension of CVXPY [68]. The problem objective and convex constraints are expressed using standard CVXPY semantics. Nonconvex constraints are expressed implicitly by creating a variable constrained to lie in one of the sets described in §2.4. For example, the code snippet

```
x = Boolean()
```

creates a variable $x \in \mathbf{R}$ with the implicit nonconvex constraint $x \in \{0, 1\}$. The convex relaxation, in this case $x \in [0, 1]$, is also implicit in the variable definition. The source code for NCVX is available at `https://github.com/cvxgrp/ncvx`.

2.5.1 Variable constructors

The NCVX package provides the following functions for creating variables with implicit nonconvex constraints, along with many others not listed:

- `Boolean(n)` creates a variable $x \in \mathbf{R}^n$ with the implicit constraint $x \in \{0,1\}^n$.
- `Integer(n, M)` creates a variable $x \in \mathbf{R}^n$ with the implicit constraints $x \in \mathbf{Z}^n$ and $\|x\|_\infty \leq \lfloor M \rfloor$.
- `Card(n, k, M)` creates a variable $x \in \mathbf{R}^n$ with the implicit constraints that at most k entries are nonzero and $\|x\|_\infty \leq M$.
- `Choose(n, k)` creates a variable $x \in \mathbf{R}^n$ with the implicit constraints that $x \in \{0,1\}^n$ and has exactly k nonzero entries.
- `Rank(m, n, k, M)` creates a variable $X \in \mathbf{R}^{m \times n}$ with the implicit constraints $\mathbf{Rank}(X) \leq k$ and $\|X\|_2 \leq M$.
- `Assign(m, n)` creates a variable $X \in \mathbf{R}^{m \times n}$ with the implicit constraint that X is an assignment matrix.
- `Permute(n)` creates a variable $X \in \mathbf{R}^{n \times n}$ with the implicit constraint that X is a permutation matrix.
- `Cycle(n)` creates a variable $X \in \mathbf{R}^{n \times n}$ with the implicit constraint that X is the adjacency matrix of a Hamiltonian cycle.
- `Annulus(n,r,R)` creates a variable $x \in \mathbf{R}^n$ with the implicit constraint $r \leq \|x\|_2 \leq R$.
- `Sphere(n, r)` creates a variable $x \in \mathbf{R}^n$ with the implicit constraint $\|x\|_2 = r$.

2.5.2 Variable methods

Additionally, each variable created by the functions in §2.5.1 supports the following methods:

- `variable.relax()` returns a list of convex constraints that represent a convex relaxation of the nonconvex set $\mathcal{X}$, to which the variable belongs.
- `variable.project(z)` returns the Euclidean (or approximate) projection of z onto the nonconvex set $\mathcal{X}$, to which the variable belongs.
- `variable.restrict(z)` returns a list of convex constraints describing the convex restriction $\mathcal{S}^{\mathrm{rstr}}(z)$ at z of the nonconvex set $\mathcal{X}$, to which the variable belongs.
- `variable.neighbors(z, D)` returns a list of neighbors $\mathcal{S}^{\mathrm{ngbr}}(z, D)$ of z contained in the nonconvex set $\mathcal{X}$, to which the variable belongs.

Users can add support for additional nonconvex sets by providing functions that implement these four methods.

2.5.3 Constructing and solving problems

To construct a problem of the form (2.1), the user creates variables $z_1, \ldots, z_k$ with the implicit constraints $z_1 \in \mathcal{X}_1, \ldots, z_k \in \mathcal{X}_k$, where $\mathcal{X}_1, \ldots, \mathcal{X}_k$ are nonconvex sets, using the functions described in §2.5.1. The variable z in problem (2.1) corresponds to the vector $(z_1, \ldots, z_k)$. The components of the variable x, the objective, and the constraints are constructed using standard CVXPY syntax.

Once the user has constructed a problem object, they can apply the following solve methods:

- `problem.solve(method="relax")` solves the convex relaxation of the problem.
- `problem.solve(method="relax-round-polish")` applies the relax-round-polish heuristic. Additional arguments can be used to specify the parameters N, D, σ, and λ. By default the parameter values are $N = 5$, $D = 1$, $\sigma = 1$, and $\lambda = 10^4$. When $N > 1$, the first sample $w_1 \in \mathbf{R}^q$ is always 0. Subsequent samples are drawn i.i.d. from $N(0, \sigma^2 I)$. Neighbor search looks at all neighbors within distance D.

- `problem.solve(method="nc-admm")` applies the NC-ADMM heuristic. Additional arguments can be used to specify the number of starting points, the number of iterations the algorithm is run from each starting point, and the values of the parameters ρ, D, σ, and λ. By default the algorithm is run from 5 starting points for 50 iterations, the value of ρ is drawn uniformly from $[0, 1]$, and the other parameter values are $D = 1$, $\sigma = 1$, and $\lambda = 10^4$. The first starting point is always $z^0 = 0$ and subsequent starting points are drawn i.i.d. from $\mathcal{N}(0, \sigma^2 I)$. Neighbor search looks at all neighbors within distance D.

The relax-round-polish and NC-ADMM methods record the best point found $(x_{\text{best}}, z_{\text{best}})$ according to the merit function. The methods return the objective value $f_0(x_{\text{best}}, z_{\text{best}})$ and the residual $r(x_{\text{best}}, z_{\text{best}})$, and set the `value` field of each variable to the appropriate segment of x_{best} and z_{best}.

For example, consider the *regressor selection* problem, which we will discuss in §2.6.1. This problem can be formulated as

$$\begin{array}{ll} \text{minimize} & \|Ax - b\|_2^2 \\ \text{subject to} & \mathbf{card}(x) \leq k, \quad \|x\|_\infty \leq M, \end{array} \tag{2.9}$$

with decision variable $x \in \mathbf{R}^n$ and problem data $A \in \mathbf{R}^{m \times n}$, $b \in \mathbf{R}^m$, $M > 0$, and $k \in \mathbf{Z}_+$. The following code attempts to approximately solve this problem using our heuristic.

```
x = Card(n,k,M)
prob = Problem(Minimize(sum_squares(A*x-b)))
objective, residual = prob.solve(method="nc-admm")
```

The first line constructs a variable $x \in \mathbf{R}^n$ with the implicit constraints that at most k entries are nonzero, $\|x\|_\infty \leq M$, and $\|x\|_1 \leq kM$. The second line creates a minimization problem with objective $\|Ax - b\|_2^2$ and no constraints. The last line applies the NC-ADMM heuristic to the problem and returns the objective value and residual of the best point found.

2.5.4 Limitations

Our implementation is designed to be simple and to generalize to as many problems as possible. As a result, the implementation has several limitations in terms of computational efficiency and exploiting problem specific structure. For example, no work is cached across solves of convex subproblems. Caching factorizations or warm starting would improve performance when the convex solver supports these features. The implementation runs NC-ADMM from different initial values in parallel, but a more sophisticated implementation would use finer grained parallelism.

Neighbor search and polishing can be made more efficient than the general purpose approach in our implementation by exploiting problem specific structure. For example, if the variable z is a Boolean vector, *i.e.*, $\mathcal{X} \in \{0,1\}^q$, then any neighbor $\tilde{z} \in \mathcal{S}^{\text{rstr}}(z,1)$ differs from z in only two entries. The change in the merit function $\eta(x,\tilde{z}) - \eta(x,z)$ can often be computed efficiently given $\eta(x,z)$, accelerating neighbor search. Polishing can also be accelerated by taking advantage of the structure of the convex restriction. For instance, the restriction of the cardinality constraint $\mathcal{X} = \{z \in \mathbf{R}^q \mid \mathbf{card}(z) \leq k,\ \|z\|_\infty \leq M\}$ fixes the sparsity pattern, which reduces the number of free entries of z from q to k. Our implementation imposes the restriction by adding convex constraints to the problem, but a more efficient implementation would replace z with a k-dimensional variable.

For several of the examples in §3.3, we implemented optimized versions of the NC-ADMM algorithm that remedy the limitations of our general purpose implementation. Even our problem specific implementations could be improved further by better exploiting parallelism and applying low-level code optimization, but the implementations are fast enough to compete with optimized general purpose mixed integer solvers like Gurobi [106].

2.6 Examples

In this section we apply the NC-ADMM heuristic to a wide variety of hard problems, *i.e.*, that generally cannot be solved in polynomial time. Extensive research has been

done on specialized algorithms for each of the problems discussed in this section. Our intention is not to seek better performance than these specialized algorithms, but rather to show that our general purpose heuristic can yield decent results with minimal tuning. The advantage of our heuristic is that it can be applied to problems that no one has studied before, not that it outperforms the state-of-the-art on well-studied problems.

Unless otherwise specified, the algorithm parameters are the defaults described in §2.5. In particular, we use random initialization for all examples. For most problems a well chosen problem specific initialization will improve the results of our method; see, *e.g.*, [120, 152, 219]. We use random initialization, however, because it better demonstrates that our heuristic can be effective with minimal tuning. Whenever possible, we compare our heuristic to Gurobi [106], a commercial global optimization solver. All runtimes reported are on a laptop with a four-core 2.3 GHz Intel Core i7 processor.

2.6.1 Regressor selection

We consider the problem of approximating a vector b with a linear combination of at most k columns of A with bounded coefficients. This problem can be formulated as

$$\begin{array}{ll} \text{minimize} & \|Ax-b\|_2^2 \\ \text{subject to} & \mathbf{card}(x) \leq k, \quad \|x\|_\infty \leq M, \end{array} \tag{2.10}$$

with decision variable $x \in \mathbf{R}^n$ and problem data $A \in \mathbf{R}^{m\times n}$, $b \in \mathbf{R}^m$, $k \in \mathbf{Z}_+$, and $M > 0$. Lasso (least absolute shrinkage and selection operator) is a well-known heuristic for solving this problem by adding ℓ_1 regularization and minimizing $\|Ax - b\|_2^2 + \lambda\|x\|_1$. The value of λ is chosen as the smallest value for which $\mathbf{card}(x) \leq k$. (See [85, §3.4] and [35, §6.3].) The nonconvex set from §2.4 in problem 2.10 is the set of vectors with bounded cardinality $\mathcal{X} = \{x \in [-M, M]^n \mid \mathbf{card}(x) \leq k\}$.

Problem instances. We first consider a family of random problem instances. We generated the matrix $A \in \mathbf{R}^{m\times 2m}$ with i.i.d. $\mathcal{N}(0, 1)$ entries, and chose $b = A\hat{x} + v$,

where $\hat{x}$ was drawn uniformly at random from the set of vectors satisfying $\mathbf{card}(\hat{x}) \leq \lfloor m/5 \rfloor$ and $\|x\|_\infty \leq 1$, and $v \in \mathbf{R}^m$ was a noise vector drawn from $\mathcal{N}(0, \sigma^2 I)$. We set $\sigma^2 = \|A\hat{x}\|^2/(400m)$ so that the signal-to-noise ratio was near 20. For each value of m, we generated 40 instances of the problem as described above. We solved the instances for $k = \lfloor m/5 \rfloor$.

In order to examine this method on a real dataset, we also used data from the University of California, Irvine (UCI) Machine Learning repository [83] to study the murder rate (per 100K people) of $m = 2215$ communities in the United States. Similar to [222], we had $n = 101$ attributes measured in each community, and our goal was to predict the murder rate as a linear function of only k attributes. To find a good prediction model one would use cross validation analysis in order to choose k; but we limited ourselves to the problem of finding $2 \leq k \leq 20$ regressors that minimize $\|Ax - b\|_2^2$.

Results. Figure 2.1 compares the average sum of squares error for the x^* values found by the Lasso heuristic, relax-round-polish, and NC-ADMM for the randomly generated instances. For Lasso, we solved the problem for 100 values of λ and then solved the polishing problem after fixing the sparsity pattern suggested by Lasso. For all m, the objective values found by the NC-ADMM heuristic were on average better than those found by the Lasso and relax-round-polish heuristics.

For our second problem (murder rate), we used our NC-ADMM heuristic and Gurobi to solve the problem. Our tailored implementation of NC-ADMM never took more than 40 milliseconds to run. The implementation is extremely efficient because the dominant computation is a single factorization of the matrix $A^T A + \rho I$. We use only one restart and hence only one value of ρ. Figure 2.2 shows the value found by NC-ADMM as well as the best value found by Gurobi after 10 seconds, 100 seconds, and 1000 seconds. For all k, the objective value found by NC-ADMM after only 40 milliseconds was better than those found by Gurobi after 10 or 100 seconds and comparable to those found after 1000 seconds. (Of course, Gurobi will eventually find the global optimal point, and therefore match or beat the point found by NC-ADMM.)

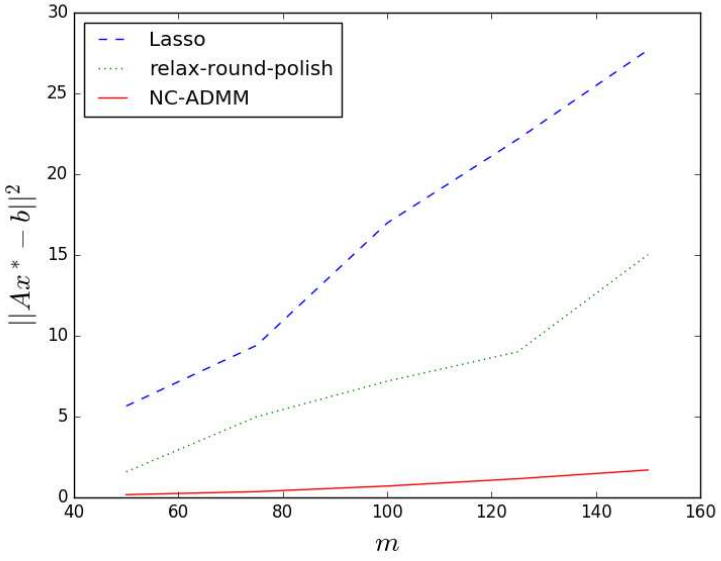

Figure 2.1: The average error of solutions found by Lasso, relax-round-polish, and NC-ADMM for 40 random instances of the regressor selection problem.

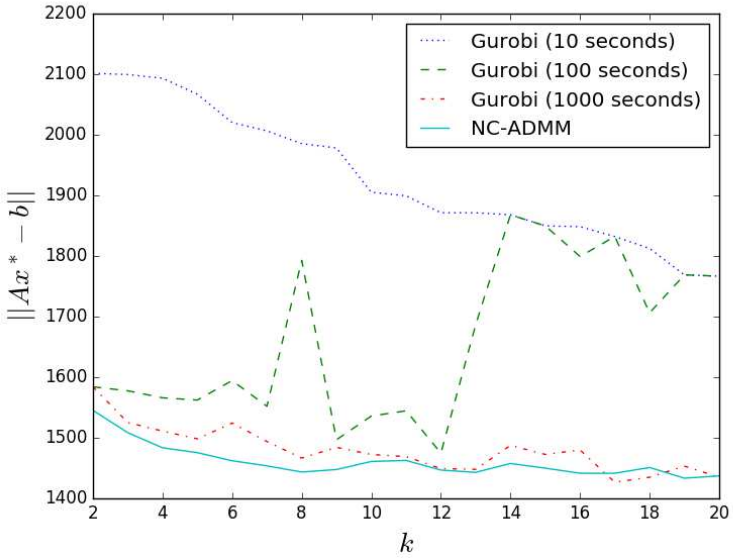

Figure 2.2: The best value found by NC-ADMM (usually done in 35 milliseconds) and Gurobi after 10 seconds, 100 seconds, and 1000 seconds.

2.6.2 3-satisfiability

Given Boolean variables $x_1, \ldots, x_n$, a *literal* is either a variable or the negation of a variable, for example x_1 and $\neg x_2$. A *clause* is disjunction of literals (or a single literal), for example $(\neg x_1 \vee x_2 \vee \neg x_3)$. Finally a formula is in conjunctive normal form (CNF) if it is a conjunction of clauses (or a single clause), for example $(\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2)$. Determining the satisfiability of a formula in conjunctive normal form where each clause is limited to at most three literals is called *3-satisfiability* or simply the *3-SAT* problem. It is known that 3-SAT is NP-complete, hence we do not expect to be able to solve 3-SAT in general using our heuristic. A 3-SAT problem can be formulated as the following

$$\begin{array}{ll} \text{minimize} & 0 \\ \text{subject to} & Az \leq b, \\ & z \in \{0,1\}^n, \end{array} \tag{2.11}$$

where entries of $A \in \mathbf{R}^{m \times n}$ are given by

$$a_{ij} = \begin{cases} -1 & \text{if clause } i \text{ contains } x_j \\ 1 & \text{if clause } i \text{ contains } \neg x_j \\ 0 & \text{otherwise,} \end{cases}$$

and the entries of b are given by

$$b_i = (\text{number of negated literals in clause } i) - 1.$$

The nonconvex set from §2.4 in problem 2.11 is the set of Boolean vectors $\mathcal{X} = \{0,1\}^n$.

Problem instances. We generated 3-SAT problems with varying numbers of clauses and variables randomly as in [176, 156]. As discussed in [59], there is a threshold around 4.25 clauses per variable when problems transition from being feasible to being infeasible. Problems near this threshold are generally found to be hard satisfiability problems. The SATLIB uniform random-3-SAT benchmark is constructed

by the same method [118]. We generated 10 instances for each choice of number of clauses m and variables n, verifying that each instance is feasible using Gurobi [106].

Results. We ran the NC-ADMM heuristic on each instance, with 10 restarts and 100 iterations, and $\rho = 10$. Figure 2.3 shows the fraction of instances solved correctly with NC-ADMM for each choice of number of clauses m and variables n. We see that using this heuristic, satisfying assignments can be found consistently for up to 3.2 constraints per variable, at which point success starts to decrease. Problems in the gray region in figure 2.3 were not tested since they are infeasible with high probability. We also tried the relax-round-polish heuristic, but it often failed to solve problems with more than 50 clauses.

For all instances, the runtime of the NC-ADMM heuristic with the parameters we chose was greater than the time it took Gurobi to find a solution. A specialized SAT solver would of course be even faster. We include the example nonetheless because it shows that the NC-ADMM heuristic can be effective for feasibility problems, even though the algorithm is not guaranteed to find a feasible point.

2.6.3 Circle packing

In the *circle packing problem* we are interested in finding the smallest square in which we can place n non-overlapping circles with radii $r_1, \ldots, r_n$ [101]. This problem has been studied extensively [215, 43, 55] and a database of densest known packings (with all r_i equal) for different numbers of circles can be found in [134]. Variants of the problem arise in industrial packing and computer aided design [110]. The problem can be formulated as

$$\begin{array}{ll} \text{minimize} & l \\ \text{subject to} & r_i\mathbf{1} \leq x_i \leq (l - r_i)\mathbf{1}, \quad i = 1, \ldots, n \\ & x_i - x_j = z_{ij}, \quad i = 1, \ldots, n-1, \quad j = i+1, \ldots, n \\ & 2\sum_{k=1}^{n} r_i \geq \|z_{ij}\|_2 \geq r_i + r_j, \quad i = 1, \ldots, n-1, \quad j = i+1, \ldots, n, \end{array} \tag{2.12}$$

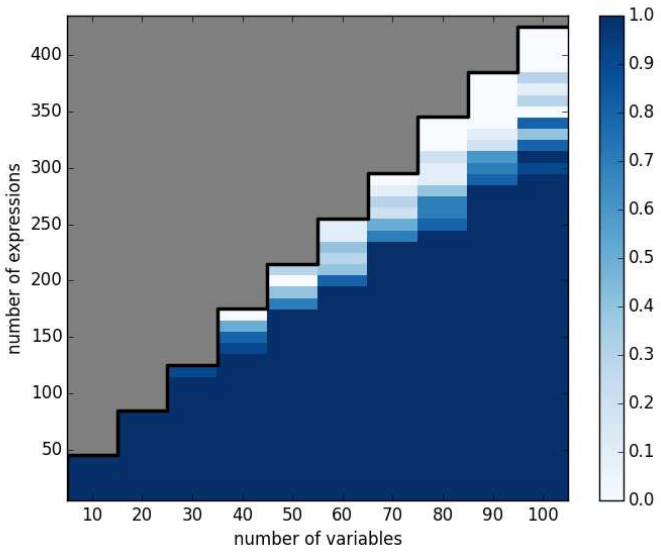

Figure 2.3: The fraction of the 10 3-SAT instances generated for each choice of number of clauses m and variables n for which NC-ADMM found a satisfying assignment. No instances were generated for (n, m) in the gray region.

where $x_1, \ldots, x_n \in \mathbf{R}^2$ are variables representing the circle centers and $z_{12}, z_{13}, \ldots, z_{n-1,n} \in \mathbf{R}^2$ are additional variables representing the offset between pairs (x_i, x_j). The non-convex set from §2.4 in problem 2.12 are the annuli

$$\mathcal{X}_{ij} = \{z_{ij} \in \mathbf{R}^2 \mid r_i + r_j \leq \|z_{ij}\|_2 \leq 2\sum_{k=1}^{n} r_i\},$$

for $i = 1, \ldots, n-1$ and $j = i+1, \ldots, n$.

Problem instances. We generated problems with different numbers of circles n, but with equal radii $r_1, \ldots, r_n$. Problem instances of this form are quite difficult to solve globally. The densest possible packing is unknown for most $n > 36$ [134].

Results. We ran the relax-round-polish heuristic for problems with $n = 1, \ldots, 100$. The heuristic is essentially equivalent to well-known methods like the convex-concave procedure and the majorization-minimization (MM) algorithm [156]. We observed that NC-ADMM is no more effective than relax-round-polish. Figure 2.4 shows the relative radius r_1/l of the packing found by our heuristic in comparison to the best packing known. Figure 2.5 shows the packing found by our heuristic for $n = 41$. The obtained packing covers 78.68% of the area of the bounding square, which is close to the densest known packing, which covers 79.27% of the area.

2.6.4 Traveling salesman problem

In the traveling salesman problem (TSP), we wish to find the minimum weight Hamiltonian cycle in a weighted graph. A Hamiltonian cycle is a path that starts and ends on the same vertex and visits each other vertex in the graph exactly once. Let G be a graph with n vertices and $D \in \mathbf{S}^n$ be the (weighted) adjacency matrix, *i.e.*, the real number d_{ij} denotes the distance between i and j. We can formulate the TSP problem for G as follows

$$\begin{array}{ll} \text{minimize} & (1/2)\,\mathbf{Tr}(D^T Z) \\ \text{subject to} & Z \in \mathcal{H}_n, \end{array} \tag{2.13}$$

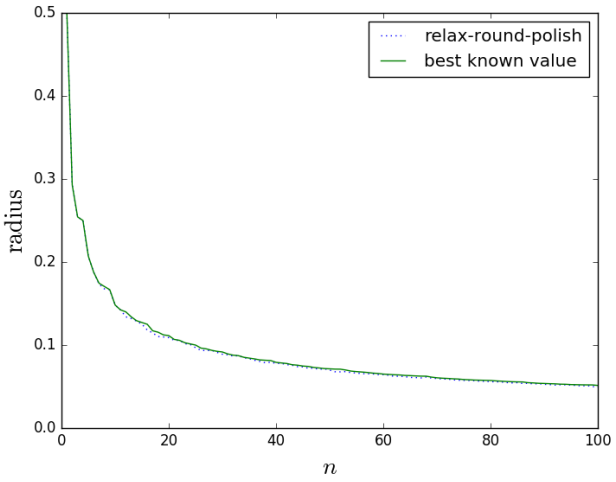

Figure 2.4: The relative radius r_1/l for the densest known packing and the packing found with the relax-round-polish heuristic for $n = 1, \ldots, 100$.

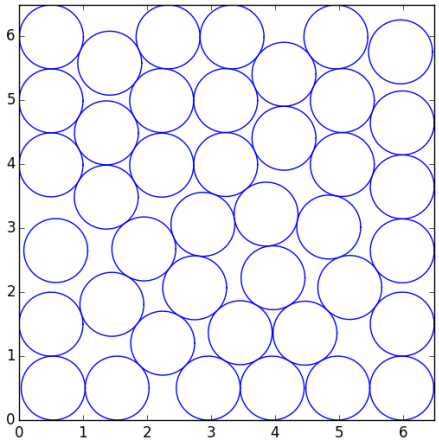

Figure 2.5: The packing for $n = 41$ circles with equal radii found with the relax-round-polish heuristic.

where Z is the decision variable [145, 142, 62, 114]. The nonconvex set from §2.4 in problem 2.13 is the set of Hamiltonian cycles $\mathcal{X} = \mathcal{H}_n$.

Problem instances. We generated problems with different numbers of vertices n by sampling n points from the uniform distribution on $[-1, 1]^2$. We set d_{ij} to be the Euclidean distance between points i and j. For each value of n, we generated 10 instances of the problem according to the above procedure.

Results. Figure 2.6 shows the average cost of the solutions found by relax-round-polish, NC-ADMM, and Gurobi with a time cutoff. We implemented an optimized version of NC-ADMM for the TSP problem. We ran NC-ADMM with 4 restarts and 25 iterations. We ran Gurobi on the standard MILP formulation of the TSP [191, §13] and gave it a time cutoff equal to the runtime of our NC-ADMM implementation. We ignored instances where Gurobi failed to find a feasible point within the runtime.

As n increases, the average cost of the solutions found by NC-ADMM goes below that of Gurobi with a time cutoff. Of course a specialized TSP solver like Concorde [7] could solve all the problem instances to global optimality within the runtime of NC-ADMM. We emphasize again, however, that our goal is not to outperform specialized solvers on every problem class, but simply for NCVX to compare favorably with other general purpose nonconvex solvers.

2.6.5 Factor analysis model

The factor analysis problem decomposes a matrix as a sum of a low-rank and a diagonal matrix and has been studied extensively (for example in [200, 183]). It is also known as the *Frisch* scheme in the system identification literature [136, 63]. The problem is the following

$$\begin{array}{ll} \text{minimize} & \|\Sigma - \Sigma^{\text{lr}} - D\|_F^2 \\ \text{subject to} & D = \mathbf{diag}(d), \quad d \geq 0 \\ & \Sigma^{\text{lr}} \succeq 0 \\ & \mathbf{Rank}(\Sigma^{\text{lr}}) \leq k, \end{array} \tag{2.14}$$

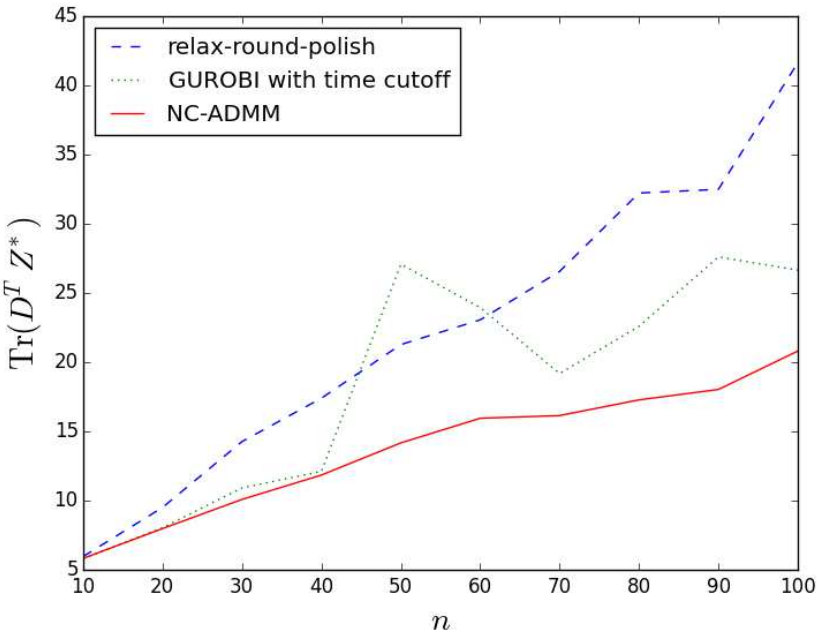

Figure 2.6: The average cost of the TSP solutions found by relax-round-polish, NC-ADMM, and Gurobi with a time cutoff equal to the runtime of NC-ADMM.

where $\Sigma^{\text{lr}} \in \mathbf{S}_+^n$ and diagonal matrix $D \in \mathbf{R}^{n\times n}$ with nonnegative diagonal entries are the decision variables, and $\Sigma \in \mathbf{S}_+^n$ and $k \in \mathbf{Z}_+$ are problem data. One well-known heuristic for solving this problem is adding $\|\cdot\|_*$, or nuclear norm, regularization and minimizing $\|\Sigma - \Sigma^{\text{lr}} - D\|_F^2 + \lambda\|\Sigma^{\text{lr}}\|_*$ [226, 200]. The value of λ is chosen as the smallest value possible such that $\mathbf{Rank}(\Sigma^{\text{lr}}) \leq k$. Since Σ^{lr} is positive semidefinite, $\|\Sigma^{\text{lr}}\|_* = \mathbf{Tr}(\Sigma^{\text{lr}})$. The nonconvex set from §2.4 for problem 2.14 is the set of matrices with bounded rank $\mathcal{X} = \{\Sigma^{\text{lr}} \in \mathbf{S}_+^n \mid \mathbf{Rank}(\Sigma^{\text{lr}}) \leq k\}$. Unlike in §2.4, we constrain Σ^{lr} to be positive semidefinite but impose no bound on the norm $\|\Sigma^{\text{lr}}\|_2$.

Problem instances. We constructed instances of the factor analysis problem using daily returns from stocks in the July 2016 S&P 500 over 2014 and 2015. There is a long history in finance of decomposing the covariance of stock returns into low-rank and diagonal components [207, 196]. We varied the number of stocks used n, the rank k, and the month of returns history considered. For each choice of n, k, and month, we generated an instance of problem 2.14 by setting Σ to be the covariance matrix of the daily percent returns over that month for the first n S&P 500 stocks, ordered alphabetically by NYSE ticker.

Results. We ran NC-ADMM, relax-round-polish, and the nuclear norm heuristic on each problem instance. For the nuclear norm heuristic, we solved the problem for 1000 values of λ and then polished the solution $\hat{\Sigma}^{\text{lr}}$. In the polishing problem we replaced the rank constraint in problem 2.14 with the convex restriction $\Sigma^{\text{lr}} \in \{Q_{1:k}\tilde{\Sigma}Q_{1:k}^T \mid \tilde{\Sigma} \in \mathbf{S}_+^k\}$, where $\hat{\Sigma} = Q\Lambda Q^T$ is the eigendecomposition of $\hat{\Sigma}^{\text{lr}}$ and $Q_{1:k}$ is the first k columns of Q.

The Σ^{lr} and d values found by the three methods were always feasible solutions to problem 2.14. For each problem instance and each method, we took the value of the objective $\|\Sigma - \Sigma^{\text{lr}} - D\|_F^2$ obtained by the method and subtracted the smallest objective value obtained by any method, p^{best}. Figure 2.7 shows the average $\|\Sigma - \Sigma^{\text{lr}} - D\|_F^2 - p^{\text{best}}$ across all 24 months of returns data, for a given n and k. NC-ADMM always gave the best objective value on average (though not for each specific problem instance). The performance of relax-round-polish relative to NC-ADMM increased as

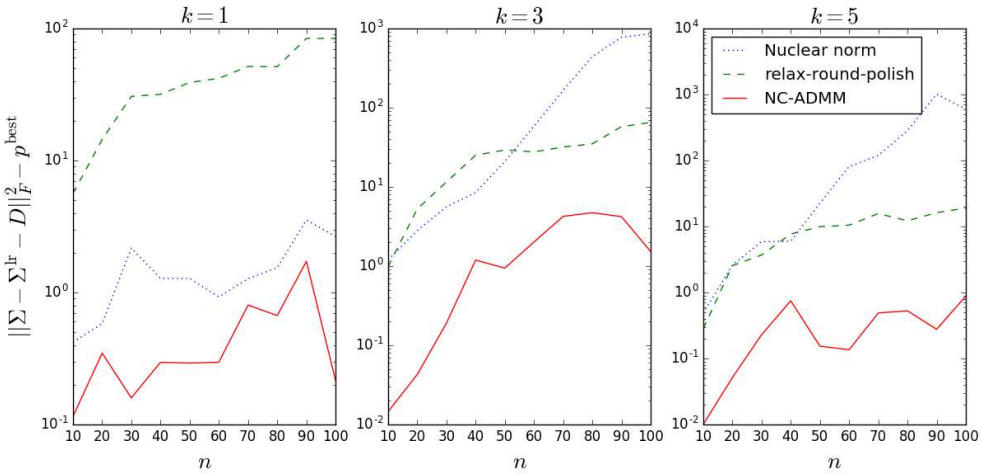

Figure 2.7: The average difference between the objective value found by the nuclear norm, relax-round-polish, and NC-ADMM heuristics and the best objective value found by any of the heuristics for instances of the factor analysis problem constructed from daily stock returns.

k increased, while the relative performance of the nuclear norm heuristic decreased as k increased.

2.6.6 Inexact graph isomorphism

Two (undirected) graphs are isomorphic if we can permute the vertices of one so it is the same as the other (i.e., the same pairs of vertices are connected by edges). If we describe them by their adjacency matrices A and B, isomorphism is equivalent to the existence of a permutation matrix $Z \in \mathbf{R}^{n\times n}$ such that $ZAZ^T = B$, or equivalently $ZA = BZ$.

Since in practical applications isomorphic graphs might be contaminated by noise, the inexact graph isomorphism problem is usually stated [4, 225, 60], in which we want to find a permutation matrix Z such that the disagreement $\|ZAZ^T - B\|_F^2$ between the transformed matrix and the target matrix is minimized. Solving inexact graph isomorphism problems is of interest in pattern recognition [57, 199], computer vision [202], shape analysis [205, 108], image and video indexing [148], and neuroscience

[229]. In many of the aforementioned fields graphs are used to represent geometric structures, and $\|ZAZ^T - B\|_F^2$ can be interpreted as the strength of geometric deformation.

Since $\|ZAZ^T - B\|_F^2 = \|ZA - BZ\|_F^2$ for any permutation matrix Z, the inexact graph isomorphism problem can be formulated as

$$\begin{array}{ll} \text{minimize} & \|ZA - BZ\|_F^2 \\ \text{subject to} & Z \in \mathcal{P}_n. \end{array} \tag{2.15}$$

If the optimal value of this problem is zero, it means that A and B are isomorphic. Otherwise, the solution of this problem minimizes the disagreement of ZAZ^T and B in the Frobenius norm sense. The nonconvex set from §2.4 in problem 2.15 is the the set of permutation matrices $\mathcal{X} = P_n$.

Problem instances. It can be shown that if A and B are isomorphic and A has distinct eigenvalues and all eigenvectors v of A satisfy $\mathbf{1}^T v \neq 0$, then the relaxed problem has a unique solution which is the permutation matrix that relates A and B [4]. Hence in our first experiment, in order to generate harder problems, we generated the matrix A such that it violated these conditions. In particular, we constructed A for the Peterson graph (3-regular with 10 vertices), icosahedral graph (5-regular with 12 vertices), Ramsey graph (8-regular with 17 vertices), dodecahedral graph (3-regular with 20 vertices), and the Tutte-Coxeter graph (3-regular with 30 vertices). For each example we randomly permuted the vertices to obtain two isomorphic graphs.

We also used random graphs from the SIVALab dataset [64] in our second experiment. These are Erdos-Renyi graphs that have been used for benchmarking different graph isomorphism algorithms. We ran our NCVX heuristic and Gurobi on 100 problems of size $n = 20, 40, 60, 80$.

Results. We implemented a faster version of NC-ADMM that caches work between convex solves. We ran our implementation with 25 iterations, 2 restarts, and no neighbor search. For all of our examples in the first experiment NC-ADMM was able to find the permutation relating the two graphs. It is interesting to notice that

running the algorithm multiple times can find different solutions if there is more than one permutation relating the two graphs.

We compared NC-ADMM with Gurobi on random example in our second experiment. We ran Gurobi with a time limit of 300 seconds. Whenever Gurobi found a permutation matrix that gave an objective value of 0 it immediately returned the solution since the lower bound 0 was evident. NC-ADMM found a permutation solution for 97 out of 100 examples.

Figure 2.8 shows the runtime performance of the two methods. Each point shows how long NC-ADMM or Gurobi ran on a particular problem instance. Points with time component of 300 seconds indicate instances that Gurobi was unable to find a solution within the time limit. The goal of this comparison is to test the performance of generic methods on the graph isomorphism problem; tailored methods for this problem are significantly faster than both of these methods.

2.7 Conclusions

We have discussed the relax-round-polish and NC-ADMM heuristics and demonstrated their performance on many different problems with convex objectives and decision variables from a nonconvex set. Our heuristics are easy to extend to additional problems because they rely on a simple mathematical interface for nonconvex sets. We need only know a method for (approximate) projection onto the set. We do not require but benefit from knowing a convex relaxation of the set, a convex restriction at any point in the set, and the neighbors of any point in the set under some discrete distance metric. Adapting our heuristics to any particular problem is straightforward, and we have fully automated the process in the NCVX package.

We do not claim that our heuristics give state-of-the-art results for any particular problem. Rather, the purpose of our heuristics is to give a fast and reasonable solution with minimal tuning for a wide variety of problems. Our heuristics also take advantage of the tremendous progress in technology for solving general convex optimization problems, which makes it practical to treat solving a convex problem as a black box.

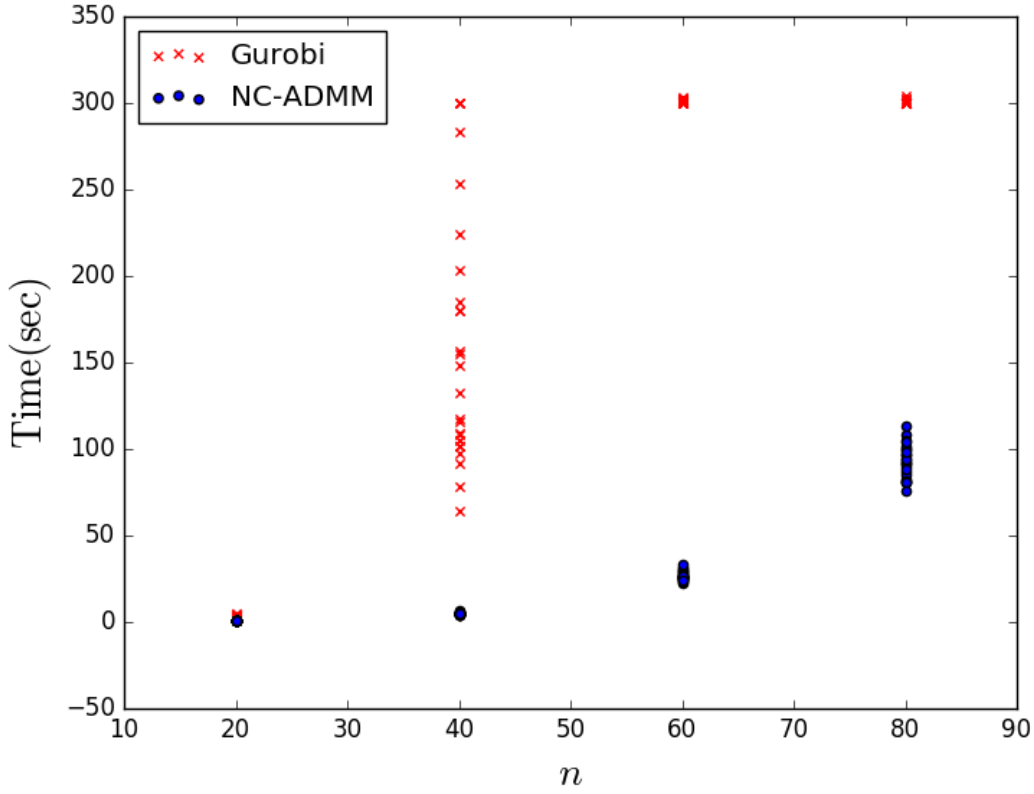

Figure 2.8: Time comparison of Gurobi and NC-ADMM on random graph isomorphism problems. Each point shows how long NC-ADMM or Gurobi ran on a particular problem instance.

Chapter 3

Mixed-Integer Quadratic Programming

3.1 Introduction

3.1.1 The problem

We consider the problem

$$\begin{array}{ll} \text{minimize} & (1/2)x^T P x + q^T x + r \\ \text{subject to} & Ax = b \\ & x \in \mathcal{X} \end{array} \tag{3.1}$$

with decision variable $x \in \mathbf{R}^n$. The problem parameters are the symmetric positive semidefinite matrix $P \in \mathbf{R}^{n\times n}$, the matrix $A \in \mathbf{R}^{m\times n}$, the vectors $b \in \mathbf{R}^m$ and $q \in \mathbf{R}^n$, and the real number $r \in \mathbf{R}$. The constraint set $\mathcal{X}$ is the Cartesian product of (possibly nonconvex) real, closed, nonempty sets, *i.e.*, $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_n$, where $\mathcal{X}_i \subseteq \mathbf{R}$ are closed, nonempty subsets of $\mathbf{R}$ for $i = 1, \ldots, n$. If $\mathcal{X}_i$ is a convex set, we refer to variable x_i as a *convex variable*, and if $\mathcal{X}_i$ is a nonconvex set, we call variable x_i a *nonconvex variable*.

Many problems can be put into the form of problem (3.1). For example, if some of the sets $\mathcal{X}_i$ are subsets of integers, our formulation addresses mixed-integer quadratic

and mixed-integer linear programs. This includes applications such as admission control [187], economic dispatch [192], scheduling [44], hybrid vehicle control [178], thermal unit commitment problems [42], Boolean satisfiability problems [128], and hybrid model predictive control [18]. Another application is embedded signal decoding in communication systems, when the nonconvex sets are signal constellations (*e.g.*, QAM constellations; see [99, pg. 416]).

If $\mathcal{X}$ is a convex set, problem (3.1) is a convex optimization problem and can be readily solved using standard convex optimization techniques. Otherwise, the problem (3.1) can be hard in general. It trivially generalizes mixed-integer quadratic programming, an NP-complete problem, and can therefore be used to encode other NP-complete problems such as the traveling salesman problem [191], Boolean satisfiability [153, 138], set cover [112], and set packing [188]. Hence, any algorithm that guarantees finding the global solution to (3.1) suffers from non-polynomial worst-case time complexity (unless P = NP).

3.1.2 Solve techniques

There are a variety of methods for solving (3.1) exactly. When all of the nonconvex sets $\mathcal{X}_i$ in (3.1) are finite, the simplest method is brute force; enumerating through all possible combinations of discrete variables, solving a convex optimization problem for each possible combination, and finding the point with the smallest objective value. Other methods such as branch-and-bound [146] and branch-and-cut [216] are guaranteed to find the global solution. Cutting plane methods [102, 52] rely on solving the relaxation and adding a linear constraint to drive the solution towards being integer. Special purpose methods have been introduced for some specific subclasses of (3.1). Unfortunately, these methods have non-polynomial worst-case runtime, and are often burdensome to use in practice, especially for embedded optimization, where runtime, memory limits, and code simplicity are prioritized. Also, these methods suffer from a large variance in the algorithm runtime.

On the other hand, many heuristics have been introduced that can deliver a good, but suboptimal (and possibly infeasible) point in a very short amount of time. For

example, the *relax-and-round* heuristic consists of replacing each $\mathcal{X}_i$ by its convex hull, solving the resulting relaxation (a convex quadratic program), and projecting the solution onto the nonconvex constraint sets. Another heuristic is to fix the nonconvex variables to several reasonable guess values and solve the convex optimization problem for convex variables. (Each of these methods may not find a feasible point, even if one exists.) The *feasibility pump* is a heuristic to find a feasible solution to a generic mixed integer program and is discussed in [79, 21, 2]. Such heuristics are often quite effective, and can be implemented on very modest computational hardware, making them very attractive for embedded applications (even without any theoretical guarantees).

3.1.3 Embedded applications

We focus on embedded applications where finding a feasible point with relatively small objective value will often result in performance that is practically indistinguishable from implementing the global solution. In embedded applications, the computational resources are limited and a solution must be found in a small time. Hence, methods to find the global solution are not favorable, because their large variance in runtime cannot be tolerated.

In an embedded application, it is often required to solve several instances of (3.1), with different values of the parameters. Here we distinguish two separate use cases, depending on whether one or both of P or A change. This distinction will play an important role in solution methods. In the first use case, we solve many instances of (3.1) in which any of the parameters may change between instances. In the second use case, we solve instances of (3.1) in which q, b, and $\mathcal{X}$ change between instances, but P and A are constant. Although this is more restrictive than the first use case, many applications can be well modeled using this approach, including linear, time-invariant model predictive control and moving horizon estimation. Indeed, all of the three examples we present in Section 3.3 are of this type.

3.1.4 Contributions

Our proposed algorithm is a simple and computationally efficient heuristic to find approximate solutions to problem (3.1) quickly. It is based on the alternating direction method of multipliers (ADMM), an algorithm for solving convex optimization problems. Because the problem class we address includes nonconvex optimization problems, our method is not guaranteed to find the global solution, or even converge.

Numerical experiments suggest that this heuristic is an effective tool to find the global solution in a variety of problem instances. Even if our method does not find the global solution, it usually finds a feasible point with reasonable objective value. This makes it effective for many embedded optimization applications, where finding a feasible point with relatively small objective value often results in performance that is practically indistinguishable from implementing the global solution. An implementation of our algorithm along with numerical examples is available at `www.github.com/cvxgrp/miqp_admm`.

Comparison of the runtime with commercial solvers such as MOSEK [8] and CPLEX [58] show that our method can be substantially faster than solving a global optimization method, while having a competitive practical performance.

3.1.5 Related work

In recent years, much research has been devoted to solving moderately-sized convex optimization problems quickly (*i.e.*, in milliseconds or microseconds), possibly on embedded platforms. Examples include the SOCP solvers ECOS [71], and FiordOs [224], and the QP solver CVXGEN [166]. Other algorithms have been developed exclusively for convex optimal control problems; see [235, 184, 123, 77, 70]. In addition, recent advances in automatic code generation for convex optimization [167, 50] can significantly reduce the cost and complexity of using an embedded solver. Some recent effort has been devoted to (globally) solving mixed-integer convex programs very quickly; see [17], [84] and references therein.

Even though ADMM was originally introduced as a tool for convex optimization problems, it turns out to be a powerful heuristic method even for NP-hard nonconvex

problems [34, Sections 5, 9]. ADMM has been studied extensively in the 80's [100, 88, 23]. More recently, it has found applications in a variety of distributed settings in machine learning such as model fitting, resource allocation, and classification. (See *e.g.*, [231, 206, 232, 245, 177, 203, 11].) Recently, this tool has been used as a heuristic to find approximate solutions to nonconvex problems [47, 46, 87, 164]. In [66], the authors study the *Divide and Concur* algorithm as a special case of a message-passing version of the ADMM, and introduce a three weight version of this algorithm which greatly improves the performance for some nonconvex problems such as circle packing and the Sudoku puzzle. Consensus ADMM has been used for general quadratically constrained quadratic programming in [120]. In [242], ADMM has been applied to nonnegative matrix factorization with missing values. ADMM also has been used for real and complex polynomial optimization models in [129], for constrained tensor factorization in [155], and for optimal power flow in [76]. There is a long history of using the method of multipliers to (attempt to) solve nonconvex problems [46, 47, 115, 117, 195, 234, 152].

3.2 Our heuristic

3.2.1 Algorithm

Our proposed algorithm is an extension of the alternating direction method of multipliers (ADMM) for constrained optimization to the nonconvex setting [34, Sections 5,9]. ADMM was originally introduced for solving convex problems, but practical evidence suggests that it can be an effective method to approximately solve some nonconvex problems as well. In order to use ADMM, we rewrite problem (3.1) as

$$\begin{array}{ll} \text{minimize} & (1/2)x^T P x + q^T x + r + I_{\mathcal{X}}(z) \\ \text{subject to} & \left[\begin{array}{c} A \\ I \end{array}\right] x - \left[\begin{array}{c} 0 \\ I \end{array}\right] z = \left[\begin{array}{c} b \\ 0 \end{array}\right]. \end{array} \tag{3.2}$$

Here $I_{\mathcal{X}}$ denotes the indicator function of $\mathcal{X}$, so that $I_{\mathcal{X}}(x) = 0$ for $x \in \mathcal{X}$ and $I_{\mathcal{X}}(x) = \infty$ for $x \notin \mathcal{X}$. Each iteration in the algorithm consists of the following three

steps:

$$x^{k+1/2} := \mathrm{argmin}_x \Bigg((1/2)x^T P x + q^T x + r + \tag{3.3}$$

$$(\rho/2) \left\| \begin{bmatrix} A \\ I \end{bmatrix} x - \begin{bmatrix} 0 \\ I \end{bmatrix} x^k - \begin{bmatrix} b \\ 0 \end{bmatrix} + u^k \right\|_2^2 \Bigg) \tag{3.4}$$

$$x^{k+1} := \Pi \left(x^{k+1/2} + \begin{bmatrix} 0 & I \end{bmatrix} u^k \right) \tag{3.5}$$

$$u^{k+1} := u^k + \begin{bmatrix} A \\ I \end{bmatrix} x^{k+1/2} - \begin{bmatrix} 0 \\ I \end{bmatrix} x^k - \begin{bmatrix} b \\ 0 \end{bmatrix}. \tag{3.6}$$

Here, Π denotes the projection onto $\mathcal{X}$, vector $u \in \mathbf{R}^{m+n}$ is the dual variable, and $\rho \in \mathbf{R}$ is a scalar parameter. (We will discuss parameter selection later in this chapter.) Note that if $\mathcal{X}$ is not convex, the projection onto $\mathcal{X}$ may not be unique; for our purposes, we only need that $\Pi(z) \in \mathrm{argmin}_{x \in \mathcal{X}} \|x - z\|_2$ for all $z \in \mathbf{R}^n$. Since $\mathcal{X}$ is the Cartesian product of subsets of the real line, *i.e.*, $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_n$, we can take $\Pi(z) = (\Pi_1(z_1), \cdots, \Pi_n(z_n))$, where Π_i is a projection function onto $\mathcal{X}_i$. Usually evaluating $\Pi_i(z)$ is inexpensive; for example, if $\mathcal{X}_i = [\alpha, \beta]$ is an interval, $\Pi_i(z) = \min\{\max\{z, \alpha\}, \beta\}$. If $\mathcal{X}_i$ is the set of integers, Π_i rounds its argument to the nearest integer. For any finite set $\mathcal{X}_i$ with k elements, $\Pi_i(z)$ is a closest point to z that belongs to $\mathcal{X}_i$, which can be found by $\lceil \log_2 k \rceil$ comparisons.

3.2.2 Convergence

If the set $\mathcal{X}$ is convex and problem (3.1) is feasible, the algorithm is guaranteed to converge to an optimal point [34, §3]. However, for $\mathcal{X}$ nonconvex, there is no such guarantee. Indeed, because problem (3.1) can be NP-hard, any algorithm that finds the global solution suffers from nonpolynomial worst-case runtime. Our approach is to give up the accuracy and use methods that find an approximate solution in a small time. (Note that although the points x^{k+1} are not necessarily feasible, they are always in the set $\mathcal{X}$, which may be sufficient for some applications.)

Our numerical results verify that even for simple examples, the algorithm may fail to converge, converge to a suboptimal point, or fail to find a feasible point, even if

one exists. Since the objective value need not decrease monotonically (or at all), it is critical to keep track of the best point found. That is, for a selected primal feasibility tolerance ϵ^{tol}, we shall reject all points x such that $\|Ax - b\| > \epsilon^{\text{tol}}$, and among those primal feasible points x that $\|Ax - b\| \leq \epsilon^{\text{tol}}$, we choose the point with the smallest objective value. Here, ϵ^{tol} is a tolerance for accepted feasibility. We should remind the reader again, that this point need not be the global minimum.

3.2.3 Initialization

To initialize x^0, one can randomly choose a point in $\mathbf{Co}\,\mathcal{X}$, where $\mathbf{Co}\,\mathcal{X}$ denotes the convex hull of $\mathcal{X}$. More specifically, this means that we need to have access to a subroutine that generates random points in $\mathbf{Co}\,\mathcal{X}$. Our numerical results show that running the algorithm multiple times with different random initializations increases the chance of finding a feasible point with smaller objective value. Hence, we suggest running the algorithm multiple times initialized with random starting points and report the best point as the approximate solution. We always initialize $u^0 = 0$.

3.2.4 Computational cost

In this subsection, we make a few comments about the computational cost of each iteration. The first step involves minimizing a strongly convex quadratic function and is actually a linear operator. The point $x^{k+1/2}$ can be found by solving the following system of equations:

$$\begin{bmatrix} P + \rho I & A^T \\ A & -(1/\rho)I \end{bmatrix} \begin{bmatrix} x^{k+1/2} \\ v \end{bmatrix} = \begin{bmatrix} q' \\ 0 \end{bmatrix},$$

where $q' = -q + \rho\left(x^k + A^T b - \begin{bmatrix} A^T & I \end{bmatrix} u^k\right)$. Since the matrix on the lefthand side remains constant for all iterations, we can precompute the LDL^T factorization of this matrix once and cache the factorization for use in subsequent iterations. When P and A are dense, the factorization cost is $O(n^3)$, yet each subsequent iteration costs only $O(n^2)$. (Both factorization and solve costs can be significantly smaller if P or A is

sparse.) Amortizing the factorization step over all iterations means that the first step is quite efficient. Also notice that the matrix on the lefthand side is quasi-definite and hence suitable for LDL^T factorization.

In many applications, P and A do not change across problem instances. In this case, for different problem instances, we solve (3.1) for the same P and A and varying b and q. This lets us use the same LDL^T factorization, which results in a significant saving in computation.

The second step involves projection onto $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_n$ and can typically be done much more quickly than the first step. It can be done in parallel since the projection onto $\mathcal{X}$ can be found by projections onto $\mathcal{X}_i$ for $i = 1, \ldots, n$. The third step is simply a dual update and is computationally inexpensive.

3.2.5 Preconditioning

Both theoretical analysis and practical evidence suggest that the precision and convergence rate of first-order methods can be significantly improved by preconditioning the problem. Here, we use diagonal scaling as preconditioning as discussed in [15] and [241]. Diagonal scaling can be viewed as applying an appropriate linear transformation before running the algorithm. When the set $\mathcal{X}$ is convex, the preconditioning can substantially affect the speed of convergence, but does not affect the quality of the point returned, (which must be a solution to the convex problem). In other words, for convex problems, preconditioning is simply a tool to help the algorithm converge faster. Optimal choice of preconditioners, even in the convex case, is still an active research area [95, 97, 94, 96, 93, 209, 116, 31, 65]. In the nonconvex case, however, preconditioning can have a critical role in the *quality* of approximate solution, as well as the speed at which this solution is found.

Specifically, let $F \in \mathbf{R}^{n\times n}, E \in \mathbf{R}^{m\times m}$ be diagonal matrices with positive diagonal entries. The goal is to choose F and E to improve the convergence of ADMM on the

preconditioned problem

$$\begin{array}{ll} \text{minimize} & (1/2)x^TPx + q^Tx + I_{\mathcal{X}}(z) \\ \text{subject to} & \left[\begin{array}{c} EA \\ F \end{array}\right] x - \left[\begin{array}{c} 0 \\ F \end{array}\right] z = \left[\begin{array}{c} Eb \\ 0 \end{array}\right]. \end{array} \tag{3.7}$$

We use the choice of E and F recommended in [95] to minimize the effective condition number (the ratio of the largest singular value to the smallest non-zero singular value) of the following matrix

$$\left[\begin{array}{cc} E & 0 \\ 0 & F \end{array}\right] \left[\begin{array}{c} A \\ I \end{array}\right] P^{\dagger} \left[\begin{array}{cc} A^T & I \end{array}\right] \left[\begin{array}{cc} E & 0 \\ 0 & F \end{array}\right],$$

where $P^{\dagger}$ denotes the pseudo-inverse of P. Given matrix $M \in \mathbf{R}^{n\times n}$, minimizing the condition number of DMD for diagonal $D \in \mathbf{R}^{n\times n}$ can be cast as a semidefinite program. However, a heuristic called *matrix equilibration* can be used to avoid the computational cost of solving a semidefinite program. (See [211, 36] and references therein.) Since for embedded applications computational resources are limited, we avoid finding $P^{\dagger}$ or equilibrating completely. We instead find E to normalize the rows of A (usually in ℓ_1 or ℓ_2 norm) and set F to be the identity.

After finding E and F, preconditioned ADMM has the following form:

$$\begin{aligned} x^{k+1/2} &:= \left[\begin{array}{cc} I & 0 \end{array}\right] \left[\begin{array}{cc} P + \rho F^2 & A^TE \\ EA & -(1/\rho)I \end{array}\right]^{-1} \\ &\quad \left[\begin{array}{c} -q + \rho\left(F^2x^k + A^TE^2b - \left[\begin{array}{cc} A^TE & F \end{array}\right] u^k\right) \\ 0 \end{array}\right] \\ x^{k+1} &:= \Pi\left(x^{k+1/2} + \left[\begin{array}{cc} 0 & F^{-1} \end{array}\right] u^k\right) \\ u^{k+1} &:= u^k + \left[\begin{array}{c} EA \\ F \end{array}\right] x^{k+1/2} - \left[\begin{array}{c} 0 \\ F \end{array}\right] x^k - \left[\begin{array}{c} Eb \\ 0 \end{array}\right]. \end{aligned} \tag{3.8}$$

3.2.6 The overall algorithm

We use the update rules (3.8) for $k = 1, \ldots, N$, where N denotes the (fixed) maximum number of iterations. Also, as described above, the algorithm is repeated for M number of random initializations. The computational cost of the algorithm consists of a factorization and MN matrix products and projections. A description of the overall algorithm is given in Algorithm 7, with $f(x) = (1/2)x^T Px + q^T x + r$.

Algorithm 6 Approximately solving nonconvex constrained QP (3.1)

if A or P changed **then**
  find E and F by equilibrating $\begin{bmatrix} A \\ I \end{bmatrix} P^{\dagger} \begin{bmatrix} A^T & I \end{bmatrix}$
  find LDL^T factorization of $\begin{bmatrix} P + \rho F^2 & A^T E \\ EA & -(1/\rho)I \end{bmatrix}$
end if
$x_{\text{best}} := \emptyset$, $f(x_{\text{best}}) := \infty$
for random initialization $1, 2, \ldots, N$ **do**
  for iteration $1, 2, \ldots, M$ **do**
    update x from (3.8)
    if $\|Ax - b\|_2 \leq \epsilon^{\text{tol}}$ and $f(x) < f(x_{\text{best}})$ **then**
      $x_{\text{best}} = x$
    end if
  end for
end for
return x_{best}

We mention a solution refinement technique here that can be used to find a solution with possibly better objective value after the algorithm stops. This technique, sometimes known as *polishing*, consists of fixing the nonconvex variable and solving the resulting convex optimization problem. Using this technique, one may use larger ϵ^{tol} during the N iterations and only reduce ϵ^{tol} at the refinement step. Depending on the application, it might be computationally sensible to solve the resulting convex optimization problem. Another effective technique is to introduce a notion of *no-good cut* during iterations for problems with binary variables. A no-good cut forces the

vector of binary variables to change over iterations, by appending the linear equality constraint $\sum_{i \in T} x_i - \sum_{i \in F} x_i \leq B - 1$, to the minimization in the first step of (3.6), where we have $T = \{i \mid x_{b_i}^k = 1\}$ (*i.e.*, T is the set of binary variables for which the last iterate was 1), $F = \{i \mid x_{b_i}^k = 0\}$, (*i.e.*, F is the set of binary variables for which the last iterate was 0), and B is the number of elements of T. We do not use either of these techniques in the following examples.

3.3 Numerical examples

In this section, we explore the performance of our proposed algorithm on some example problems. For each example, ρ was chosen between 0.1 and 10 to yield good performance; all other algorithm parameters were kept constant. As a benchmark, we compare our results to the commercial solver MOSEK, which can globally solve MIQPs. All experiments were carried out on a system with two 3.06 GHz cores with 4 GB of RAM.

The results suggest that this heuristic is effective in finding approximate solutions for mixed integer quadratic programs.

3.3.1 Randomly generated QP

First we demonstrate the performance of our algorithm qualitatively for a random mixed-Boolean quadratic program. The matrix P in (3.1) was chosen as $P = QQ^T$, where the entries of $Q \in \mathbf{R}^{n \times n}$, as well as those of q and A, were drawn from a standard normal distribution. The constant r was chosen such that the optimal value of the unconstrained quadratic minimization is 0. The vector b was chosen as $b = Ax_0$, where $x_0 \in \mathcal{X}$ was chosen uniformly randomly, thus ensuring that the problem is feasible. We used $n = 200$ and $m = 50$ with $\mathcal{X}_i = \{0, 1\}$ for $i = 1, \ldots, 100$, $\mathcal{X}_i = \mathbf{R}_+$ for $i = 101, \ldots, 150$, and $\mathcal{X}_i = \mathbf{R}$ for the other indices i.

We used MOSEK to find the optimal value for the problem. After more than 16 hours MOSEK certifies that the optimal value is equal to 2040. We ran algorithm 7 for 10 different initializations and 200 iterations for each initialization, with step size

$\rho = 0.5$. For a naive implementation in MATLAB, it took 120 milliseconds to complete all precomputations (preconditioning and factorization), and 800 milliseconds to do all 2000 iterations. The best objective value found for the problem was 2067 (1.3% suboptimal). Our implementation in C enables us to solve sparse problems significantly faster.

One interesting observation is that the parameter ρ tends to trade off feasibility and optimality: with small values of ρ, the algorithm often fails to find a feasible point, but feasible points found tend to have low objective value. On the other hand, with large values of ρ, feasible points are found more quickly, but tend to have higher objective value.

3.3.2 Hybrid vehicle control

We consider a simple hybrid electric vehicle drivetrain (similar to that of [35, Exercise 4.65]), which consists of a battery, an electric motor/generator, and a heat engine, in a parallel configuration. Control of a hybrid vehicle appears as an embedded practice in application [45, 89, 180]. We assume that the demanded power P_t^{des} at the times $t = 0, \ldots, T-1$ is known in advance. Our task is to plan out the battery and engine power outputs P_t^{batt} and P_t^{eng}, for $t = 0, \ldots, T-1$, so that

$$P_t^{\text{batt}} + P_t^{\text{eng}} \geq P_t^{\text{des}}.$$

(Strict inequality above corresponds to braking.)

The battery has stored energy E_t at time t, which evolves according to

$$E_{t+1} = E_t - \tau P_t^{\text{batt}}, \qquad t = 0, \ldots, T-1,$$

where τ is the length of each discretized time interval. The battery capacity is limited, so that $0 \leq E_t \leq E^{\max}$ for all t, and the initial energy E_0 is known. We penalize the terminal energy state of the battery according to $g(E_T)$, where

$$g(E) = \eta(E^{\max} - E)^2,$$

for $\eta \geq 0$.

At time t, the engine may be on or off, which is modeled with binary variable z_t. If the engine is on ($z_t = 1$), then we have $0 \leq P_t^{\text{eng}} \leq P^{\max}$, and $\alpha(P_t^{\text{eng}})^2 + \beta P_t^{\text{eng}} + \gamma$ units of fuel are consumed, for nonnegative constants α, β, and γ. If the engine is off ($z_t = 0$), it consumes no fuel, and $P_t^{\text{eng}} = 0$. Because $z_t \in \{0, 1\}$, the power constraint can be written as $0 \leq P^{\text{eng}} \leq P^{\max} z_t$, and the fuel cost as $f(P_t^{\text{eng}}, z_t)$, where

$$f(P, z) = \alpha P^2 + \beta P + \gamma z.$$

Additionally, we assume that turning the engine on after it has been off incurs a cost $\delta \geq 0$, *i.e.*, at each time t, we pay $\delta(z_t - z_{t-1})_+$, where $(\cdot)_+$ denotes the positive part.

The hybrid vehicle control problem can be formulated as

$$\begin{array}{ll} \text{minimize} & \eta(E_T - E^{\max})^2 + \sum_{t=0}^{T-1} f(P_t^{\text{eng}}, z_t) \\ & +\delta(z_t - z_{t-1})_+ \\ \text{subject to} & E_{t+1} = E_t - \tau P_t^{\text{batt}} \\ & P_t^{\text{batt}} + P_t^{\text{eng}} \geq P_t^{\text{des}} \\ & z_t \in \{0, 1\}, \end{array} \tag{3.9}$$

where all constraints must hold for $t = 0, \ldots, T-1$. The variables are P_t^{batt}, P_t^{eng}, and z_t for $t = 0, \ldots, T-1$, and E_t, for $t = 1, \ldots, T$. In addition to the parameters given above, we take z_{-1} to be a parameter denoting the initial engine state.

We used the parameter values $\alpha = 1$, $\beta = 10$, $\gamma = 1.5$, $\delta = 10$, $\eta = 0.1$, $\tau = 5$, $P^{\max} = 1$, $E^{\max} = 200$, $E_0 = 200$, and $z_{-1} = 0$. The demanded power trajectory P_t^{des} is not shown, but can be obtained by summing the engine power and battery power in Figure 3.1. We ran the algorithm with $\rho = 0.4$ for 900 iterations, with primal optimality threshold $\epsilon^{\text{tol}} = 10^{-4}$. The global solution found by MOSEK generates an objective value of 139.52 and the best objective value with our algorithm was 140.07. In Figure 3.1, we see that qualitatively, the optimal trajectory and the trajectory generated by ADMM are very similar. Our implementation in C carries out precomputations in 27 milliseconds. The total time for all 900 iterations is 63 milliseconds, which gives each iteration an average time of 70 microseconds. MOSEK finds the

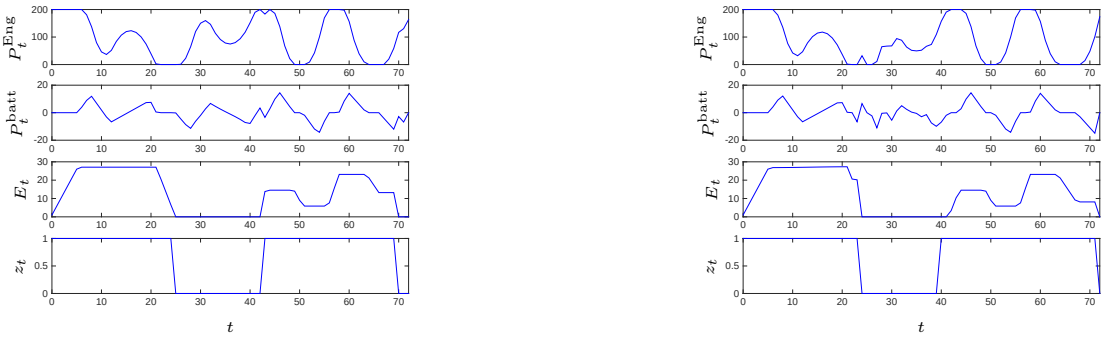

Figure 3.1: Engine power, battery power, battery energy, and engine on/off signals versus time. Left: the global solution. Right: the solution found using ADMM (Algorithm 1).

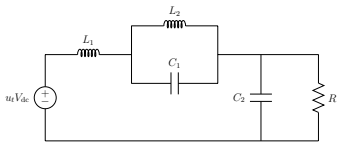

Figure 3.2: Converter circuit model.

first feasible point after 1 second, and it takes about 15 seconds to find a point with the same quality as found with our heuristic.

3.3.3 Power converter control

We discuss the control of an embedded switched-mode power converter control [212, 38, 40]. We consider control of the switched-mode power converter shown in Figure 3.2. The circuit dynamics are

$$\xi_{t+1} = G\xi_t + Hu_t, \qquad t = 0, 1, \ldots, T-1,$$

where $\xi_t = (i_{1,t}, v_{1,t}, i_{2,t}, v_{2,t})$ is the system state at epoch t, consisting of all inductor currents and capacitor voltages, and $u_t \in \{-1, 0, 1\}$ is the control input. The dynamics matrices $G \in \mathbf{R}^{4\times 4}$ and $H \in \mathbf{R}^{4\times 1}$ are obtained by discretizing the dynamics of the circuit in Figure 3.2.

We would like to control the switch configurations so that v_2 tracks a desired sinusoidal waveform. This can be done by solving

$$\begin{array}{ll} \text{minimize} & \sum_{t=0}^{T}(v_{2,t}-v_{\text{des}})^2+\lambda|u_t-u_{t-1}| \\ \text{subject to} & \xi_{t+1}=G\xi_t+Hu_t \\ & \xi_0=\xi_T \\ & u_0=u_T \\ & u_t\in\{-1,0,1\}, \end{array} \tag{3.10}$$

where $\lambda \geq 0$ is a tradeoff parameter between output voltage regulation and switching frequency. The variables are ξ_t for $t = 0, \ldots, T$ and u_t for $t = 0, \ldots, T-1$.

Note that if we take $\lambda = 0$, and take the input voltage u_t to be unconstrained (*i.e.*, allow u_t to take any values in $\mathbf{R}$), (3.10) can be solved as a convex quadratic minimization problem, with solution ξ_t^{ls}. Returning to our original problem, we can penalize deviation from this ideal waveform by including a regularization term $\mu\|\xi - \xi_t^{\text{ls}}\|^2$ to (3.10), where $\mu > 0$ is a positive weighting parameter. We solved this regularized version of (3.10), with $L_1 = 10\,\mu\text{H}$, $C_1 = 1\,\mu\text{F}$, $L_2 = 10\,\mu\text{H}$, $C_2 = 10\,\mu\text{F}$, $R = 1\,\Omega$, $V_{\text{dc}} = 10\,\text{V}$, $T = 100$ (with a discretization interval of $0.5\,\mu\text{s}$), $\lambda = 1.5\,\text{V}^2$, and $\mu = 0.1$. We run algorithm 7 with $\rho = 2.7$ and 500 iterations for three different initializations. It takes less than 20 milliseconds for our implementation to carry out all precomputations, and it takes about 150 milliseconds for all iterations (with an average time of 100 microseconds per iteration). An approximate solution is found via our heuristic in less than 170 milliseconds, whereas it takes MOSEK more than 4 hours to find the global solution. Figure 3.3 compares the approximate solution derived by the heuristic with the global solution.

3.3.4 Signal decoding

We consider maximum-likelihood decoding of a message passed through a linear multiple-input and multiple-output (MIMO) channel [210, 228, 61]. In particular, we have

$$y = Hx + v,$$

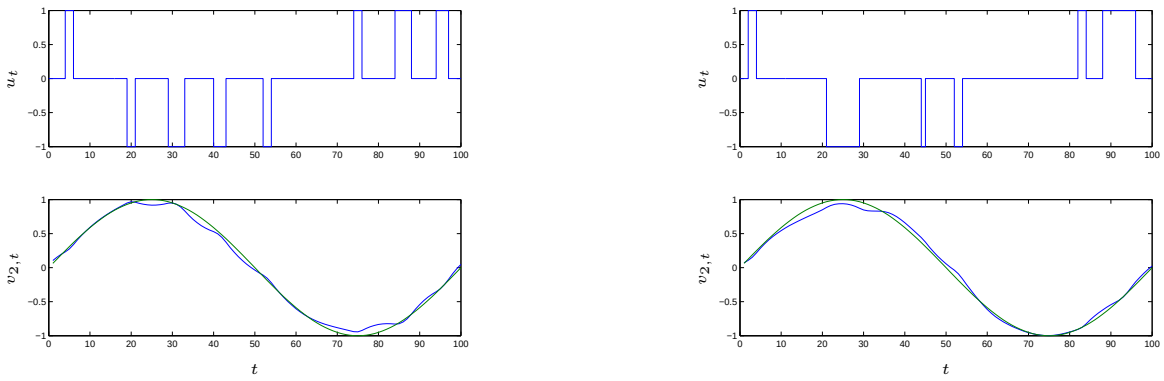

Figure 3.3: The switch configuration and the output voltage. Left: the global solution. Right: the solution using ADMM (Algorithm 1).

where $y \in \mathbf{R}^p$ is the message received, $H \in \mathbf{R}^{p\times n}$ is the channel matrix, $x \in \mathbf{R}^n$ is the message sent, and the elements of the noise vector $v \in \mathbf{R}^p$ are independent, identically distributed Gaussian random variables. We further assume that the elements of x belong to the *signal constellation* $\{-3,-1,1,3\}$. The maximum likelihood estimate of x is given by the solution to the problem

$$\begin{array}{ll} \text{minimize} & \|H\hat{x} - y\|^2 \\ \text{subject to} & \hat{x}_i \in \{-3,-1,1,3\}, \quad i = 1,\ldots,n, \end{array} \tag{3.11}$$

where $\hat{x} \in \mathbf{R}^n$ is the variable.

We generate 1000 random problem instances with $H \in \mathbf{R}^{2000\times 400}$ chosen from a standard normal distribution. The uncorrupted signal x is chosen uniformly randomly and the additive noise is Gaussian such that the signal-to-noise ratio (SNR) is 8 dB. For such a problem in embedded application, branch-and-bound methods are not desirable due to their worst-case time complexity. We run the heuristic with only one initialization, with 10 iterations to find x^{admm}. The average runtime for each problem (including preprocessing) is 80 milliseconds, which is substantially faster than branch-and-bound based methods. We compare the performance of the points x^{admm} with the points found by relax-and-round technique x^{rlx}. In Figure 3.4 we have plotted the histogram of the difference between the objective values evaluated at x^{admm} and x^{rlx}.

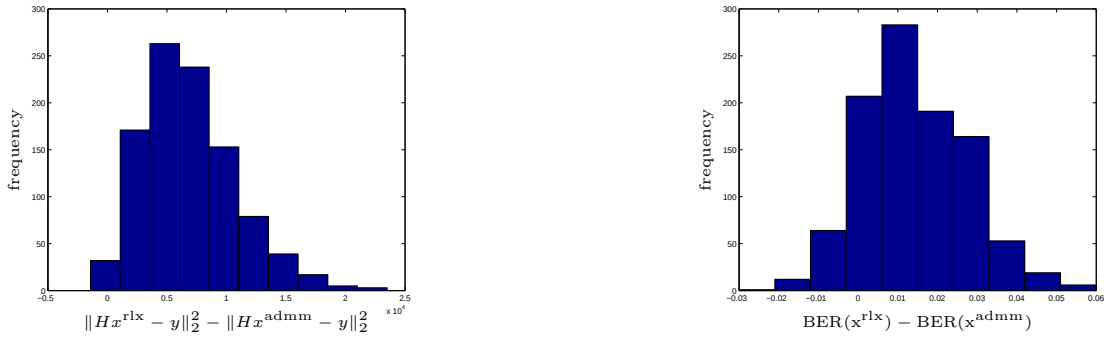

Figure 3.4: Comparison of ADMM heuristic and relax-and-round. Left: The difference in objective values. Right: The difference in bit error rates.

Depicted in Figure 3.4, we see that in 95% of the cases, the bit error rate using our heuristic was at least as good as the bit error rate (BER) using relax and round.

3.4 Conclusions

In this chapter, we introduced an effective heuristic for finding approximate solutions to convex quadratic minimization problems over the intersection of affine and non-convex sets. Our heuristic is significantly faster than branch-and-bound algorithms and has shown effective in a variety of embedded problems including hybrid vehicle control, power converter control, and signal decoding.

Chapter 4

Linear Programming for Graph Isomorphism

4.1 Graph isomorphism problem

4.1.1 Problem statement

Consider two weighted undirected graphs, each with n vertices labeled $1, \ldots, n$, described by their adjacency matrices $A, \tilde{A} \in \mathbf{R}^{n \times n}$, where A_{ij} is the weight on the edge in the first graph between vertices i and j, and zero if there is no edge between vertices i and j (and similarly for $\tilde{A}$). Since the graphs are undirected, the adjacency matrices are symmetric. The two graphs are isomorphic if there is a permutation of the vertices of the first graph that makes the first graph the same as the second. This occurs if and only if there is a permutation matrix $P \in \mathbf{R}^{n \times n}$ (*i.e.*, a matrix with exactly one entry in each row and column that is one, with the other zero) that satisfies $PAP^T = \tilde{A}$. We will say that the permutation matrix P transforms A to $\tilde{A}$ if $PAP^T = \tilde{A}$, or equivalently, $PA = \tilde{A}P$.

The graph isomorphism problem (GIP) is to determine whether such a permutation matrix exists, and to find one if so. GIP can be formulated as the (feasibility)

optimization problem

$$\begin{array}{ll} \text{find} & P \\ \text{subject to} & PA = \tilde{A}P \\ & P\mathbf{1} = \mathbf{1}, \quad P^T\mathbf{1} = \mathbf{1} \\ & P_{ij} \in \{0,1\}, \quad i,j = 1,\ldots,n, \end{array} \tag{4.1}$$

with variable $P \in \mathbf{R}^{n\times n}$, where $\mathbf{1}$ is the vector with all entries one. The data for this problem are the adjacency matrices of the two graphs, A and $\tilde{A}$. The constraints on the last two lines enforce that P is a permutation matrix. This problem has n^2 scalar variables (*i.e.*, P_{ij}), and all constraints except the last one (which requires the variables to take on Boolean values 0 or 1) are linear in P. The problem (4.1) is an integer (or Boolean) linear program (LP).

If P transforms A to $\tilde{A}$, the two matrices are similar, and therefore have the same spectrum, *i.e.*, the same eigenvalues including multiplicity. This observation gives a very simple spectral condition for isomorphism: The spectra of A and $\tilde{A}$ must be the same. Since this condition is readily checked, we will henceforth assume that this is the case.

Assumption 1. *Adjacency matrices A and $\tilde{A}$ have the same eigenvalues including multiplicity.*

Assumption 1 does not imply that the graphs are isomorphic. But if assumption 1 does not hold, the graphs are surely not isomorphic.

GIP arises in several applications including chem-informatics, mathematical chemistry, electronic design automation, and network theory. For example it can be used in determining if two descriptions of a molecule are the same [122], or whether the physical layout of an electronic circuit correctly reflects the given circuit schematic diagram [54].

4.1.2 Computational complexity

GIP is one of the few problems in NP that has so far resisted all attempts to be classified as NP-complete, or within P. In 1979, Garey and Johnson [90] listed 12

such problems and the GIP is one of only two on that list whose complexity remains unresolved. For many restricted graph classes, polynomial time algorithms are known. This is, for example, the case for trees [140], planar graphs [119], interval graphs [159], partial k-trees [30], graphs of bounded degree [160], or graphs with bounded eigenvalue multiplicity [13]. Recently, Babai announced a quasipolynomial time algorithm for all graphs, *i.e.*, one with running time $2^{O((\log n)^c)}$ for some fixed $c > 0$ [12].

This chapter does not advance the discussion of computational complexity of GIP; instead we describe effective heuristics for it. However, many of the ideas we will encounter (*e.g.*, compact graphs, graph spectrum) are closely connected to those that arise in research on its computational complexity.

4.1.3 A simple relaxation

One relaxation of problem (4.1) can be obtained by replacing constraints $P_{ij} \in \{0, 1\}$ with interval relaxation $0 \leq P_{ij} \leq 1$. The relaxed problem can be written as

$$\begin{array}{ll} \text{find} & P \\ \text{subject to} & PA = \tilde{A}P \\ & P\mathbf{1} = \mathbf{1}, \quad P^T\mathbf{1} = \mathbf{1} \\ & P \geq 0, \end{array} \tag{4.2}$$

where the last constraint is entry-wise. The constraint $P \leq 1$ is removed because it is implied by $P\mathbf{1} = P^T\mathbf{1} = \mathbf{1}$ and $P \geq 0$. This problem has n^2 scalar variables, $n^2 + 2n$ scalar linear equality constraints (some of which are redundant), and n^2 scalar linear inequality constraints. While (4.1) is a Boolean LP, the relaxation (4.2) is an LP, and readily solved in polynomial time using different methods such as interior-point methods or ellipsoid method [35].

If the relaxed problem (4.2) is infeasible, then A and $\tilde{A}$ are obviously not isomorphic. Therefore, from this point of the chapter on, we assume that this problem is feasible.

Assumption 2. *We assume the relaxed problem (4.2) is feasible.*

Assumption 2 does not imply that the graphs are isomorphic. But if assumption 2 does not hold, the graphs are surely not isomorphic. The necessary and sufficient conditions for feasibility of this LP has been studied in [201] and is closely related to common equitable partitions of two graphs.

The set of feasible points for problem (4.2) is a polytope in $\mathbf{R}^{n\times n}$. Permutation matrices transforming A to $\tilde{A}$ are extreme points or vertices of this polytope. (However, not every extreme point of this polytope is necessarily a permutation matrix.) Given assumption 2, GIP comes down to finding a permutation matrix among extreme points of the feasible set of problem (4.2). Our algorithms are all based on variations and extensions of this observation.

Surprisingly, under some conditions on the graphs (stated in §4.2.4), this relaxation is tight; that is, the set of feasible point for problems (4.1) and (4.2) are both singletons. Hence, it is sufficient to solve problem (4.2) to find a permutation matrix transforming A to $\tilde{A}$, or certify that A and $\tilde{A}$ are not isomorphic if problem (4.2) is infeasible.

4.1.4 Outline

We will describe the basic version of the algorithm in §4.2, then describe sparsity constraints in §4.3 to tighten the relaxation and increase the probability of success for the heuristic. An ADMM-based algorithm to solve the problem is described in §4.5, and finally in §4.6 we study some examples and numerical experiments.

4.2 Algorithm

4.2.1 Basic randomized subroutine

As we discussed in §4.1.3, the set of feasible points for problem (4.2) form a polytope and permutation matrices transforming A to $\tilde{A}$ are extreme points or vertices of this polytope. In order to encourage finding an extreme point of this polytope, we minimize a random linear objective over the set of feasible points. Hence our basic randomized subroutine is to generate a random matrix $W \in \mathbf{R}^{n\times n}$ with i.i.d. Gaussian

entries and solve the LP

$$\begin{array}{ll} \text{minimize} & \mathbf{Tr}(W^T P) \\ \text{subject to} & PA = \tilde{A}P \\ & P\mathbf{1} = \mathbf{1}, \quad P^T\mathbf{1} = \mathbf{1} \\ & P \geq 0. \end{array} \tag{4.3}$$

If the solution to problem (4.3) found happens to be a permutation matrix we conclude that we solved this instance of GIP, since A and $\tilde{A}$ are isomorphic and the solution $P^\star$ is a permutation matrix that transforms A to $\tilde{A}$. On the other hand, if the solution found is not a permutation matrix, we cannot say anything about A and $\tilde{A}$.

Notice that with probability 1 the solution of problem (4.3) is unique. Let p^{succ} denote the probability under the distribution of W that the randomized subroutine finds a permutation matrix when A and $\tilde{A}$ are isomorphic. It is easy to see that p^{succ} is only a function of A (and does not depend on $\tilde{A}$). We briefly comment here that $p^{\text{succ}} > 0$.

Since scaling W does not change the optimization problem, without loss of generality we can assume that W is chosen uniformly from an (n^2-1)-sphere in an n^2 dimensional space. Consider the set of permutation matrices in $\mathbf{R}^{n\times n}$ that transform A to $\tilde{A}$. Centered around each such permutation matrix, there is a cone with nonzero solid angle such that if $-W$ is in that cone, the solution of problem (4.3) will be a permutation matrix. Probability of success p^{succ} is the probability of W being in one of these cones, and hence the probability that this algorithm successfully finds an acceptable permutation matrix.

Even though solving just one instance of problem (4.3) might look unsophisticated, it turns out to be an effective heuristic to solve the GIP in some cases. It is guaranteed to find a solution of GIP (with probability one) under some conditions which are discussed in next subsection.

Standard solvers based on the simplex method and interior-point methods can be used to solve problem (4.3), but can be inefficient in practice. However, we will show in §4.5 that the special structure of this problem enables us to use a custom method

to solve this optimization problem more efficiently.

4.2.2 Polishing

After the basic randomized subroutine converges to a doubly stochastic matrix $P^* \in \mathbf{R}^{n\times n}$, we use can the Hungarian algorithm [143] to project P^* on the set of permutation matrices, which we denote by $\tilde{P}^*$. Finding the closest permutation matrix to P^* (in Frobenius norm) can be done in $O(n^3)$ time. If $\tilde{P}^* A = \tilde{A}\tilde{P}^*$, then a permutation mapping A to $\tilde{A}$ is found. This step, as will see in §4.6, can increase the probability of success of the subroutine.

4.2.3 Repeated randomized algorithm

As we discussed in §4.2.1, choosing a random instance of W and running the basic randomized algorithm will find a permutation that relates A and $\tilde{A}$ with probability p^{succ}. We also discussed in §4.2.2 that we can use Hungarian algorithm to (potentially) increase the probability of success. If we repeat this basic randomized algorithm for N independently chosen random instances of W the probability of success is $1-(1-p^{\text{succ}})^N$. For example if the probability of success for the basic randomized subroutine is 80%, after repeating the basic subroutine 4 times the probability of success will be 99.9%.

We evidently have $p^{\text{succ}} > 0$, meaning that by solving problem (4.3) there is a positive probability of finding a permutation matrix that that transforms A to $\tilde{A}$. In particular, by repeatedly solving problem (4.3) with independent random choices of W, we will solve GIP with probability one. This probability can be extremely small, however, so the expected number of LPs we must solve to solve GIP can be extremely large. The benefit of this probabilistic method over a deterministic spectral-based heuristic is that when a deterministic heuristic fails, there is no hope to recover the permutation that relates A and $\tilde{A}$, but this probabilistic method can be used repeatedly to find the underlying permutation. This randomized algorithm can potentially have false negatives, *i.e.*, when A and $\tilde{A}$ are isomorphic, it might fail to find a permutation relating them.

4.2.4 Theoretical guarantees

In this subsection we show that under some conditions, solving problem (4.3) is guaranteed to find a permutation solution if A and $\tilde{A}$ are isomorphic, in other words, $p^{\text{succ}} = 1$. Specifically if A has distinct eigenvalues, and for every eigenvector v of A we have $\mathbf{1}^T v \neq 0$, the relaxation in problem (4.2) is tight.

In order to show this, assume $QA = \tilde{A}Q$, where Q is a permutation matrix and let P be a doubly stochastic matrix such that $PA = \tilde{A}P$. Also let $A = V\Lambda V^T$ be an eigendecompostion of A. Defining $R = V^T Q^T P V$, we have

$$R\Lambda = V^T Q^T P V \Lambda = V^T Q^T P A V = V^T Q^T \tilde{A} P V = V^T A Q^T P V = \Lambda V^T Q^T P V = \Lambda R.$$

Hence for every i, j we have $R_{ij}\Lambda_{ii} = \Lambda_{jj} R_{ij}$, or equivalently $R_{ij}(\Lambda_{ii} - \Lambda_{jj}) = 0$. Since $\Lambda_{ii} - \Lambda_{jj} \neq 0$ for $i \neq j$, all off-diagonal entries of R must be zero and the matrix R must be diagonal. Also $RV^T\mathbf{1} = V^T Q^T P \mathbf{1} = V^T Q^T \mathbf{1} = V^T \mathbf{1}$, which implies that $R_{ii}(V^T\mathbf{1})_i = (V^T\mathbf{1})_i$. Our second assumption on V enforces that $(V^T\mathbf{1})_i \neq 0$ and $R_{ii} = 1$ for all i and hence $P = QVRV^T = QVV^T = Q$.

For graphs specified above, relaxing the set of permutation matrices to the set of doubly stochastic matrices does not extend the set of feasible points of problem (4.1), which is a singleton.

A graph with adjacency matrix A is said to be a *compact graph* if the set of feasible points to problem (4.3) is the convex hull of the set of feasible points to problem (4.1). For example, all graphs with two aforementioned properties are compact. The concept of *compact graphs* was introduced by Tinhofer [223], who proved that trees and cycles (which violate the two aforementioned assumptions) and the disjoint union of isomorphic copies of a compact graph are compact. If A is the adjacency matrix of a compact graph and A and $\tilde{A}$ are isomorphic, then $p^{\text{succ}} = 1$. It is not a hard problem to solve the graph isomorphism problem between two trees or two cycles (or some other compact graphs), but it is interesting to see that this problem can be solved (with probability 1) by only solving one LP.

4.3 Sparsity constraints

We discussed one possible relaxation of problem (4.1) in §4.1.3. However, we can find tighter relaxations by adding convex constraints about P that we know must be true. This will create a tighter relaxation of the original problem which can potentially increase p^{succ} for the randomized algorithm. Specifically, we consider the following extension

$$\begin{array}{ll} \text{minimize} & \mathbf{Tr}(W^T P) \\ \text{subject to} & PA = \tilde{A}P \\ & P\mathbf{1} = \mathbf{1}, \quad P^T\mathbf{1} = \mathbf{1} \\ & P \geq 0 \\ & P_{ij} = 0 \quad (i,j) \in \mathcal{K}, \end{array} \tag{4.4}$$

where $\mathcal{K}$ is a set of pairs of indices. The difference between this extension and problem (4.3) is the last line of the constraints, which requires some entries in P to be zero. If $\mathcal{K} = \emptyset$, this problem reduces to problem (4.3). In general, problem (4.4) can be considered as a problem with $n^2 - \|\mathcal{K}\|$ scalar variables. Our efficient method for solving the LP relaxation (described in §4.5) can handle these constraints efficiently.

There are different ways to find a proper set $\mathcal{K}$. The simplest and least useful choice of $\mathcal{K}$ is the empty set. The maximal $\mathcal{K}$, denoted by $\mathcal{K}^{\max}$ is the set of pairs of indices i, j such that $P_{ij} = 0$ for every permutation matrix P transforming A to $\tilde{A}$. Any valid set $\mathcal{K}$ will satisfy $\emptyset \subseteq \mathcal{K} \subseteq \mathcal{K}^{\max}$.

We will show below how to find pairs of indices i, j such that for any permutation matrix P satisfying $PA = \tilde{A}P$, we must have $P_{ij} = 0$.

Lemma 1. *Let P be a permutation matrix. If $Pa = b$ for two given vectors $a, b \in \mathbf{R}^n$ and $P_{ij} = 1$, then we must have $a_j = b_i$. This implies that if $a_j \neq b_i$ then $P_{ij} = 0$.*

This simple lemma can be used to construct $\mathcal{K}$. If we know that $Pa = b$ for any permutation matrix that transforms A to $\tilde{A}$, we can include the pair (i, j) in $\mathcal{K}$. If all entries of vector a are distinct, then $\mathcal{K}$ contains $n^2 - n$ entries and permutation P is uniquely determined. At the other extreme, if all entries of vector a are equal, this lemma adds no new information.

The idea here is to use graph invariants to find equalities of the form $Pa = b$ and then construct a set $\mathcal{K}$ from these equalities. By graph invariant, we mean any function of nodes that is independent of labeling the nodes. For instance, the number of paths with length k starting from a node is an invariant, where k is a positive integer. (We will discuss this in next subsection.) In principle, the method can be used with any invariant. But we are interested in invariants that are easy to compute, *e.g.*, degree invariants and spectral invariants.

4.3.1 Degree invariants

A simple linear equality that holds is $P(\mathbf{diag}\, A) = \mathbf{diag}\, \tilde{A}$. This means that vertices that are mapped to each other must have the same self-edge weights. If A and $\tilde{A}$ have no self-edges (*i.e.*, diagonal entries of A and $\tilde{A}$ are zero), this equality will add no information.

A more interesting equality that holds is $P(A^k\mathbf{1}) = \tilde{A}^k\mathbf{1}$ for every positive integer k. In other words, the number of paths of length k starting from a vertex must be the same for two vertices that map to each other.

For example, when the graphs are unweighted (*i.e.*, the edges have weight one) this equality with $k = 1$ means that the degrees of mapped vertices must be the same. In other words, all nodes of degree i must be mapped to each other. Hence, if the number of nodes with degree i is denoted by n_i, the original problem has $(n_1 + n_2 + \cdots)^2$ variables, but we know only $n_1^2 + n_2^2 + \cdots$ of them can potentially be nonzero. In the extreme case that all degrees are equal (or equivalently the graph is regular), this method does not eliminate any variables from P.

This equality is valid for every positive integer k. However, according to the Cayley-Hamilton theorem, A^k can be written as a linear combination of $A, A^2, \ldots, A^n$ for every positive k. Hence for a given pair of indices i, j if $(A^k)_i = (A^k)_j$ for $k = 1, \ldots, n$, we will have $(A^k)_i = (A^k)_j$ for every positive k. Therefore it is only enough to consider this equality for $k = 1, \ldots, n$.

In summary, we have two sets of equalities:

- $P(\mathbf{diag}\, A) = \mathbf{diag}\, \tilde{A}$.

- $P(A^k\mathbf{1}) = \tilde{A}^k\mathbf{1}$, for $k = 1, \ldots, n$.

We denote the set of pairs of indices constructed this way by $\mathcal{K}^{\text{degree}}$. Clearly, we have $\mathcal{K}^{\text{degree}} \subseteq \mathcal{K}^{\max}$.

4.3.2 Spectral invariants

As mentioned earlier, we assume that A and $\tilde{A}$ share the same spectrum; otherwise the graphs are evidently non-isomorphic. Let λ be an eigenvalue of A with multiplicity k and columns of $V \in \mathbf{R}^{n\times k}$ be an orthonormal basis for eigenspace associated with λ, hence we have $AV = \lambda V$ and $V^TV = I_k$, where I_k denotes the identity matrix in $\mathbf{R}^{k\times k}$. Similarly, assume that columns of $\tilde{V} \in \mathbf{R}^{n\times k}$ are an orthonormal basis of eigenspace associated with λ, hence we have $\tilde{A}\tilde{V} = \lambda\tilde{V}$ and $\tilde{V}^T\tilde{V} = I_k$. We have

$$\tilde{A}(PV) = (\tilde{A}P)V = (PA)V = P(AV) = P(\lambda V) = \lambda(PV).$$

In other words columns of PV are eigenvectors of $\tilde{A}$ associated with λ. Therefore we have $PV = \tilde{V}Q$ where $Q \in \mathbf{R}^{k\times k}$. We observe that

$$Q^TQ = Q^T\tilde{V}^T\tilde{V}Q = V^TP^TPV = V^TV = I_k.$$

Hence Q is an orthogonal in $\mathbf{R}^{k\times k}$. Therefore $PV = \tilde{V}Q$ implies that $P(VV^T)P^T = \tilde{V}\tilde{V}^T$, and we have the following equalities:

- $P(\mathbf{diag}\, VV^T) = \mathbf{diag}\, \tilde{V}\tilde{V}^T$.
- $P(VV^T\mathbf{1}) = \tilde{V}\tilde{V}^T\mathbf{1}$.

Notice that $P\left(\left(VV^T\right)^k\mathbf{1}\right) = (\tilde{V}\tilde{V}^T)^k\mathbf{1}$ adds no more information for $k > 1$, since $(VV^T)^k = VV^T$ and $(\tilde{V}\tilde{V}^T)^k = \tilde{V}\tilde{V}^T$.

These equalities hold for any eigenvalue λ. We denote the set of pairs of indices constructed this way by $\mathcal{K}^{\text{spectral}}$. Clearly, we have $\mathcal{K}^{\text{spectral}} \subseteq \mathcal{K}^{\max}$.

4.3.3 Constructing $\mathcal{K}$

In summary, here is how we construct $\mathcal{K} = \mathcal{K}^{\text{degree}} \cup \mathcal{K}^{\text{spectral}}$. We start with $\mathcal{K} = \emptyset$, and we sequentially consider the degree invariant equalities for $k = 1, \ldots, n$ and spectral invariant equalities for every eigenvalue λ. For each equality, we add the new pairs of disallowed i, j to the set $\mathcal{K}$.

When $\mathcal{K}$ is constructed, the number of variables that could possibly be nonzero is $n^2 - |\mathcal{K}|$. In §4.6 we will report the number of elements in $\mathcal{K}$ for our experiments, and we see how the sparsity constraints can increase the probability of success for our heuristic, for each LP solved.

Simple examples show that $\mathcal{K}^{\text{degree}}$ and $\mathcal{K}^{\text{spectral}}$ are not necessarily subsets of each other. Therefore in general it is a good idea to include both of these constraints in the basic subroutine. Also, our examples show that *pruning* can be a helpful technique for making $\mathcal{K}$ larger. In pruning, we disallow pair i, j if no neighbor of i can be mapped to a neighbor of j.

One reasonable conjecture could be that choosing the maximal $\mathcal{K}$ would result in solving the problem with probability 1. In other words if $\mathcal{K} = \mathcal{K}^{\max}$ then $p^{\text{succ}} = 1$. This conjecture is wrong, however. It can be shown that for the Petersen graph $\mathcal{K}^{\max} = \emptyset$. Also $(1/3)A$ is a doubly stochastic matrix that commutes with A. If this conjecture were true, then A could be written as the convex combination of automorphisms of the Petersen graph. Each such automorphism Π has the property that $\Pi(v)$ is connected to v for all vertices v. But the only automorphism of the Petersen graph that has this property is the identity. Hence the Petersen graph disproves this conjecture.

4.4 Algorithm summary

A summary of the algorithm is presented below.

Algorithm 7 Randomized LP heuristic

1: Check whether assumptions 1, 2 hold. If not, declare A and $\tilde{A}$ as non-isomorphic.
2: Construct $\mathcal{K}$ as described in §4.3.3
3: **for** random instance $W_1, \ldots, W_N$ **do**
4: Solve problem (4.4) to find a solution P^*
5: Use Hungarian algorithm to find the closest permutation matrix $\tilde{P}^*$ to P^*
6: **if** $\tilde{P}^* A = \tilde{A}\tilde{P}^*$ **then**
7: Declare A and $\tilde{A}$ as isomorphic and return $\tilde{P}^*$
8: **end if**
9: **end for**

Here N denotes the number of times problem (4.4) is solved and can be chosen beforehand based on an approximation of p^{succ}. As mentioned before, the probability of the success of this algorithm is $1 - (1 - p^{\text{succ}})^N$.

4.5 Solution method

Any LP solver such as ones base on the simplex method or interior-point methods can be used to solve problem (4.4). With n^2 variables and n^2 equality constraints, using a general purpose interior-point solver would involve $O(n^6)$ flops. However, the special structure of the problem enables us to use a custom method which is more efficient. We use the alternating direction method of multipliers (ADMM) in consensus form [34] for solving problem (4.4) as follows. We write the problem as

$$\begin{array}{ll} \text{minimize} & f_1(P_1) + f_2(P_2) + f_3(P_3) \\ \text{subject to} & Z = P_i, \quad i = 1, 2, 3 \\ & ZA = \tilde{A}Z, \end{array} \tag{4.5}$$

where

$$\begin{aligned} f_1(P_1) &= (1/2)\,\mathbf{Tr}(W^T P_1) + \mathbf{I}(P_1\mathbf{1} = \mathbf{1}), \\ f_2(P_2) &= (1/2)\,\mathbf{Tr}(W^T P_2) + \mathbf{I}(P_2^T\mathbf{1} = \mathbf{1}), \\ f_3(P_3) &= \mathbf{I}(P_3 \geq 0) + \mathbf{I}\left((P_3)_{i,j} = 0, \quad (i,j) \in \mathcal{K}\right). \end{aligned}$$

Here **I** is the indicator function, and takes the value 0 if its argument is true, the value ∞ if its argument is false. Assuming that $A = V\Lambda V^T$ and $\tilde{A} = \tilde{V}\Lambda\tilde{V}^T$ are eigenvalue decompositions of A and $\tilde{A}$, the ADMM updates will be

$$\begin{aligned}
P_1^{k+1} &= \text{argmin}_{P_1\mathbf{1}=\mathbf{1}} \left(\mathbf{Tr}\left(\left(W/2 + Y_1^k \right)^T W \right) + (1/2)\|P_1 - Z^k\|_F^2 \right) \\
P_2^{k+1} &= \text{argmin}_{P_2^T\mathbf{1}=\mathbf{1}} \left(\mathbf{Tr}\left(\left(W/2 + Y_2^k \right)^T W \right) + (1/2)\|P_2 - Z^k\|_F^2 \right) \\
P_3^{k+1} &= \Pi_{\{P|f_3(P)=0\}}(Z^k - Y_3^k) \\
Z^{k+1} &= \Pi_{\{Z|ZA=\tilde{A}Z\}} \left(\frac{P_1^{k+1} + P_2^{k+1} + P_3^{k+1}}{3} \right) \\
Y_1^{k+1} &= Y_1^k + P_1^{k+1} - Z^{k+1} \\
Y_2^{k+1} &= Y_2^k + P_2^{k+1} - Z^{k+1} \\
Y_3^{k+1} &= Y_3^k + P_3^{k+1} - Z^{k+1},
\end{aligned}$$

where $\Pi_{\mathcal{C}}$ denotes the projection operator onto $\mathcal{C}$. Without loss of generality, we chose $\rho = 1$ in ADMM, because scaling ρ is equivalent to scaling W. After generating a random direction W, we scale W such that the absolute value of elements of W averages to 1. This is only for the ADMM algorithm to converge faster, and it does not affect the solution found by the ADMM. (Remember that the solution to problem (4.5) is unique with probability 1.)

In order to find the projection onto $\{Z|ZA = \tilde{A}Z\}$ we notice that $ZA = \tilde{A}Z$ is equivalent to $ZV\Lambda V^T = \tilde{V}\Lambda\tilde{V}^T Z$, which is equivalent to $\tilde{V}^T ZV\Lambda = \Lambda\tilde{V}^T ZV^T$, or equivalently $\tilde{V}^T ZV$ commutes with Λ. Hence, $ZA = \tilde{A}Z$ if and only if $(\tilde{V}^T ZV)_{ij} = 0$ for any i, j that $\Lambda_{ii} \neq \Lambda_{jj}$. After simplifying the steps above, the ADMM update

steps will reduce to

$$\begin{aligned}
P_1^{k+1} &= Z^k - W/2 - Y_1^k - (1/n)\left(\left(Z^k - W/2 - Y_1^k\right)\mathbf{1} - \mathbf{1}\right)\mathbf{1}^T \\
P_2^{k+1} &= Z^k - W/2 - Y_2^k - (1/n)\mathbf{1}\left(\mathbf{1}^T\left(Z^k - W/2 - Y_2^k\right) - \mathbf{1}^T\right) \\
P_3^{k+1} &= \max\left(Z^k - Y_3^k, 0\right) \circ S \\
Z^{k+1} &= \tilde{V}\left(\left(\tilde{V}^T\frac{P_1^{k+1} + P_2^{k+1} + P_3^{k+1}}{3}V\right) \circ R\right)V^T \\
Y_1^{k+1} &= Y_1^k + P_1^{k+1} - Z^{k+1} \\
Y_2^{k+1} &= Y_2^k + P_2^{k+1} - Z^{k+1} \\
Y_3^{k+1} &= Y_3^k + P_3^{k+1} - Z^{k+1},
\end{aligned}$$

where $\circ$ denotes entry wise product, and R_{ij} is 1 if $\lambda_i = \lambda_j$ and 0 otherwise, and $S_{ij} = 1$ if $(i, j) \in K$ and 0 otherwise.

The first three update rules are the first proximal operator in ADMM algorithm and are, in fact, minimizing a quadratic function over an affine subspace. Hence, they are linear operators in P, and are $O(n^2)$ flops. The fourth equality is the projection of $(P_1^{k+1} + P_2^{k+1} + P_3^{k+1})/3$ onto $\{Z|ZA = \tilde{A}Z\}$, and is of $O(n^3)$ complexity. Finally, the last three equalities are dual updates, and can be done in $O(n^2)$ flops. Therefore each iteration of the ADMM algorithm is done in $O(n^3)$ flops.

The practical complexity of this heuristic is $O(n^3)$, which is basically the square root of the one obtained with interior-point methods. We observe that this algorithm usually converges within tens to hundreds of ADMM iterations.

4.6 Examples

We use several test cases to examine our heuristic. In the first subsection we discuss a specific example, the Frucht Graph, in order to give a qualitative description of the heuristic. Through this example, we will describe how this heuristic can benefit from using graph invariant constraints. Then we use several standard classes of graphs for benchmarking algorithms. We will report our experiment results for undirected random graphs, two dimensional grids, cubic Hype-Hamiltonian graphs, Kronecker

Eye Filp graphs, Miyazaky Augmented graphs, and affine geometries graphs [64, 135, 133, 158, 157, 134]. These graphs are described in §4.6.2. We observe that this heuristic is more effective for some classes of graphs. We also notice that the time complexity of this heuristic is $O(n^3)$, and it is more effective when graph invariant constraints are taken into account. The code for our solver and our examples can be found at `https://github.com/cvxgrp/graph_isom`.

4.6.1 Frucht graph

The Frucht graph is an example of a graph with no nontrivial automorphisms, that violates the assumptions in §4.2.4. As shown in Figure 4.1, the Frucht graph is a 3-regular graph with 12 vertices and 18 edges. It was first described by Robert Frucht in 1939.

We permute nodes of the Frucht graph randomly and run the algorithm for $100,000$ times with and without sparsity mask. Each run uses a random instance of linear objectives (W). We observe that without using sparsity extension, in 25.2% of random instances the algorithm finds the permutation matrix. When we used sparsity mask (graph invariant constraints, or $\mathcal{K}$), which contains only 14 nonzeros, the algorithm found the correct permutation in every single instance of $100,000$ runs, so our guess is that $p^{\text{succ}} = 1$ when sparsity constraints are taken into account.

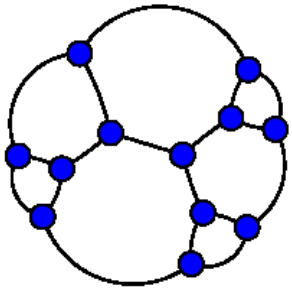

Figure 4.1: The Frucht graph.

4.6.2 Database of graphs for benchmarking algorithms

We use several classes of graphs in [64, 135, 133, 158, 157] to evaluate the effectiveness of this heuristic on a variety of graphs. In particular, our experiments were on the following classes of graphs.

- Undirected random graphs (R1N family) is taken from [64]. In these undirected Erdos-Renyi graphs, there is an edge between two vertices with probability 0.1.
- Cubic Hypo-Hamiltonian graphs clique-connected (CCH family) is proposed in [158], and is built using as basic block two non-isomorphic HypoHamiltonian graphs with 22 vertices.
- Affine geometries graphs (AG2 family) is a part of the benchmark of bliss [134]. It contains point-line graphs of affine geometries $AG_2(q)$.
- Two dimensional grids (G2N family) also comes from [64] benchmark and is discussed in [137]. They are introduced for simulating applications dealing with regular structures as those operating at the lower levels of an image processing task.
- Kronecker Eye Flip graphs (KEF family) is a part of the benchmark of bliss [134].
- Miyazaky Augmented graphs (MZA family) are also taken from the benchmark of bliss [134].

For each graph, we ran the algorithm for 50 random instances of W and have summarized the results in Table 4.1 below. The first column states the graph type, and the second column states the ratio of possibly nonzero entries in P. In other words the number in the second column is $1 - \frac{|\mathcal{K}|}{n^2}$. The third and fourth column represent the percentage of runs that a permutation was successfully found with and without sparsity constraints ($\mathcal{K}$), respectively.

As we discussed in §4.5, each iteration of the ADMM algorithm is $O(n^3)$, hence we expect that the runtime of the heuristic be a cubic function of the number of vertices

Graph type	n	sparsity ratio	success rate without $\mathcal{K}$	success rate with $\mathcal{K}$
R1N	20	0.050	1.0	1.0
R1N	40	0.025	1.0	1.0
R1N	60	0.017	1.0	1.0
R1N	80	0.012	1.0	1.0
R1N	100	0.010	1.0	1.0
R1N	200	0.005	1.0	1.0
R1N	400	0.003	1.0	1.0
R1N	600	0.002	1.0	1.0
R1N	800	0.001	1.0	1.0
R1N	1000	0.001	1.0	1.0
CHH	22	0.393	0.74	0.88
CHH	44	0.205	0.86	0.96
CHH	88	0.204	0.72	0.96
CHH	132	0.226	0.78	0.92
CHH	198	0.225	0.66	0.96
CHH	264	0.204	0.54	0.90
CHH	352	0.205	0.48	0.86
CHH	440	0.212	0.32	0.92
CHH	550	0.212	0.30	0.82
CHH	660	0.204	0.38	0.82
CHH	792	0.205	0.34	0.82
CHH	924	0.208	0.26	0.82
AG2	10	0.520	0.34	1.0
AG2	21	0.510	0.20	1.0
AG2	35	0.506	0.12	0.72
AG2	55	0.504	0	0
AG2	105	0.502	0	0
AG2	136	0.501	0	0
G2N	16	0.375	0.54	0.98
G2N	36	0.186	0.52	0.98
G2N	64	0.110	0.46	1.0
G2N	81	0.078	0.44	0.96
G2N	100	0.072	0.42	1.0
G2N	196	0.038	0.58	0.98
G2N	400	0.019	0.52	0.98
G2N	576	0.013	0.54	1.0
G2N	784	0.010	0.48	0.90
KEF	32	0.375	0.68	0.80
KEF	50	0.270	0.82	0.86
KEF	72	0.167	0.78	0.88
KEF	98	0.103	0.62	0.92
KEF	128	0.062	1.0	1.0
KEF	242	0.046	1.0	1.0
KEF	392	0.036	1.0	1.0
KEF	578	0.029	1.0	1.0
KEF	800	0.025	1.0	1.0
KEF	1058	0.021	1.0	1.0
MZA	40	0.159	1.0	1.0
MZA	80	0.085	0.62	0.76
MZA	120	0.057	0.22	0.28
MZA	160	0.044	0.24	0.32
MZA	200	0.035	0.24	0.32
MZA	240	0.029	0.08	0.16
MZA	280	0.025	0	0.08
MZA	320	0.022	0	0.08
MZA	360	0.019	0	0

Table 4.1: The outcome for 50 instances of our heuristic on different problems.

of the graphs. This is because we observe that the number of iterations that ADMM requires before it convergence does not change dramatically as the number of vertices changes. Figure 4.2 shows the scatter plot of runtime of the algorithm (denoted by t) versus the number of vertices (denoted by n) in log-scale for 50 runs over random undirected graphs (R1N family). This runtime excludes the polishing step, which is done using Hungarian algorithm. We observe that the average runtime is linear in the size of the graph when plotted in log-scale. For comparison we also used ECOS [71], an interior-point method for second-order cone programming. We see that our ADMM solver is far more scalable than interior-point methods.

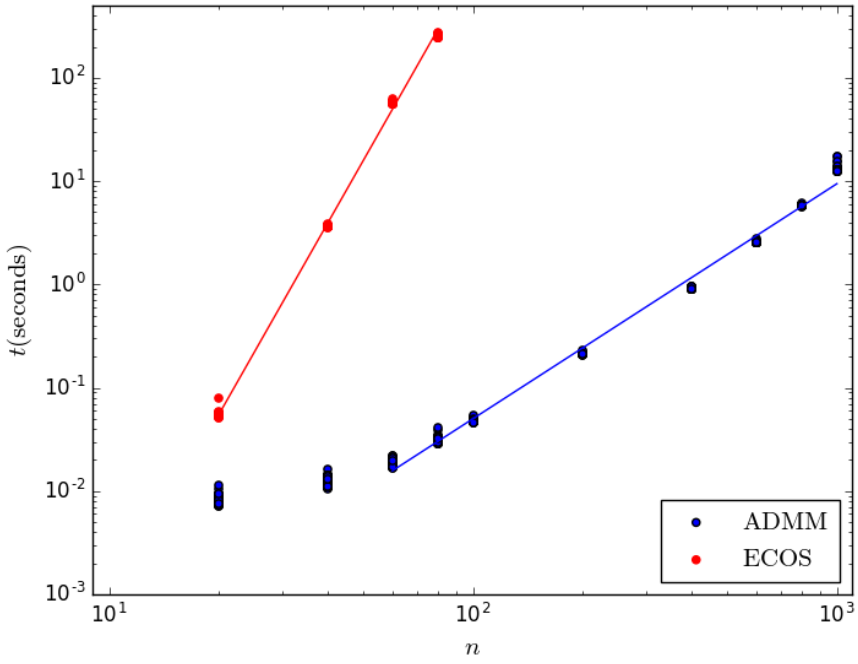

Figure 4.2: Runtime versus graph size in log-scale for 50 random runs.

Bibliography

[1] T. Achterberg. SCIP: Solving Constraint Integer Programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

[2] T. Achterberg and T. Berthold. Improving the Feasibility Pump. *Discrete Optimization*, 4(1):77–86, 2007.

[3] R. Adams and M. Laughton. Optimal planning of power networks using mixed-integer programming. Part 1: Static and time-phased network synthesis. *Proceedings of the Institution of Electrical Engineers*, 121(2):139–147, 1974.

[4] Y. Aflalo, A. Bronstein, and R. Kimmel. Graph matching: Relax or not? *arXiv preprint arXiv:1401.7623*, 2014.

[5] Y. Aflalo, A. Bronstein, and R. Kimmel. On convex relaxation of graph isomorphism. *Proceedings of the National Academy of Sciences*, 112(10):2942–2947, 2015.

[6] W. Anderson and T. Morley. Eigenvalues of the Laplacian of a graph. *Linear and Multilinear Algebra*, 18(2):141–145, 1985.

[7] D. Applegate, R. Bixby, V. Chvatal, and W. Cook. Concorde TSP solver, 2006.

[8] MOSEK ApS. *The MOSEK optimization toolbox for MATLAB manual. Version 7.1 (Revision 28)*, 2015.

[9] A. Atserias and E. Maneva. Sherali–Adams relaxations and indistinguishability in counting logics. *SIAM Journal on Computing*, 42(1):112–137, 2013.

[10] D. Axehill and A. Hansson. A preprocessing algorithm for MIQP solvers with applications to MPC. In *Proceedings of the 43rd IEEE Conference on Decision and Control*, volume 3, pages 2497–2502, 2004.

[11] N. Aybat, S. Zarmehri, and S. Kumara. An ADMM algorithm for clustering partially observed networks. In *Proceedings of the SIAM International Conference on Data Mining*. SIAM, 2015.

[12] L. Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, pages 684–697. ACM, 2016.

[13] L. Babai, D. Grigorye, and D. Mount. Isomorphism of graphs with bounded eigenvalue multiplicity. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 310–324. ACM, 1982.

[14] J. Beasley. Heuristic algorithms for the unconstrained binary quadratic programming problem. *Management School, Imperial College, London, UK*, 1998.

[15] A. Beck. *Introduction to Nonlinear Optimization: Theory, Algorithms, and Applications with MATLAB*, volume 19. SIAM, 2014.

[16] P. Belotti. Couenne: A user's manual. *Technical Report*, 2009.

[17] A. Bemporad. Solving mixed-integer quadratic programs via nonnegative least squares. *5th IFAC Conference on Nonlinear Model Predictive Control*, pages 73–79, 2015.

[18] A. Bemporad and M. Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3):407–427, 1999.

[19] A. Bemporad and P. Patrinos. Simple and certifiable quadratic programming algorithms for embedded linear model predictive control. In *Nonlinear Model Predictive Control*, volume 4, pages 14–20, 2012.

[20] A. Bemporad, J. Roll, and L. Ljung. Identification of hybrid systems via mixed-integer programming. In *IEEE Conference on Decision and Control*, volume 1, pages 786–792, 2001.

[21] L. Bertacco, M. Fischetti, and A. Lodi. A feasibility pump heuristic for general mixed-integer problems. *Discrete Optimization*, 4(1):63–76, 2007.

[22] D. Bertsekas. *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.

[23] D. Bertsekas and J. Eckstein. Dual coordinate step methods for linear network flow problems. *Mathematical Programming*, 42(1-3):203–243, 1988.

[24] D. Bertsimas and R. Weismantel. *Optimization Over Integers*, volume 13. Dynamic Ideas, Belmont, Massachusetts, 2005.

[25] D. Bienstock. Computational study of a family of mixed-integer quadratic programming problems. *Mathematical Programming*, 74(2):121–140, 1996.

[26] M. Bierlaire, P. Toint, and D. Tuyttens. On iterative algorithms for linear least squares problems with bound constraints. *Linear Algebra and its Applications*, 143:111–143, 1991.

[27] E. Bixby, M. Fenelon, Z. Gu, E. Rothberg, and R. Wunderling. MIP: Theory and practice, closing the gap. In *System modeling and optimization*, pages 19–49. Springer, 2000.

[28] R. Bixby, M. Fenelon, Z. Gu, E. Rothberg, and R. Wunderling. Mixed-integer programming: A progress report. *The Sharpest Cut: The Impact of Manfred Padberg and his work, MPS-SIAM Series on Optimization*, 4:309–326, 2004.

[29] R. Bixby and E. Rothberg. Progress in computational mixed-integer programming, a look back from the other side of the tipping point. *Annals of Operations Research*, 149(1):37–41, 2007.

[30] H. Bodlaender. Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *Journal of Algorithms*, 11(4):631–643, 1990.

[31] D. Boley. Local linear convergence of the alternating direction method of multipliers on quadratic or linear programs. *SIAM Journal on Optimization*, 23(4):2183–2207, 2013.

[32] P. Bonami, L. Biegler, A. Conn, G. Cornuéjols, I. Grossmann, C. Laird, J. Lee, A. Lodi, F. Margot, and N. Sawaya. An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optimization*, 5(2):186–204, 2008.

[33] S. Boyd, M. Hast, and K. Astrom. MIMO PID tuning via iterated LMI restriction. *International Journal of Robust and Nonlinear Control*, 2015.

[34] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[35] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[36] A. Bradley. *Algorithms for the Equilibration of Matrices and their Application to Limited-Memory Quasi-Newton Methods*. PhD thesis, Stanford University, 2010.

[37] P. Brucker, B. Jurisch, and B. Sievers. A branch and bound algorithm for the job-shop scheduling problem. *Discrete applied mathematics*, 49(1):107–127, 1994.

[38] S. Buso and P. Mattavelli. Digital control in power electronics. *Lectures on Power Electronics*, 1(1):1–158, 2006.

[39] R. Byrd, J. Nocedal, and R. Waltz. KNITRO: An integrated package for nonlinear optimization. In *Large-scale nonlinear optimization*, pages 35–59. Springer, 2006.

[40] M. Camara, H. Gualous, F. Gustin, A. Berthon, and B. Dakyo. DC/DC converter design for supercapacitor and battery power management in hybrid vehicle applications-polynomial control strategy. *IEEE Transactions on Industrial Electronics*, 57(2):587–597, 2010.

[41] J. Campbell. Integer programming formulations of discrete hub location problems. *European Journal of Operational Research*, 72(2):387–405, 1994.

[42] M. Carrión and J. Arroyo. A computationally efficient mixed-integer linear formulation for the thermal unit commitment problem. *IEEE Transactions on Power Systems*, 21(3):1371–1378, 2006.

[43] I. Castillo, F. Kampas, and J. Pintér. Solving circle packing problems by global optimization: Numerical results and industrial applications. *European Journal of Operational Research*, 191(3):786–802, 2008.

[44] J. Catalão, H. Pousinho, and V. Mendes. Scheduling of head-dependent cascaded hydro systems: Mixed-integer quadratic programming approach. *Energy Conversion and Management*, 51(3):524–530, 2010.

[45] S. Chakraborty, M. Lukasiewycz, C. Buckl, S. Fahmy, N. Chang, S. Park, Y. Kim, P. Leteinturier, and H. Adlkofer. Embedded systems and software challenges in electric vehicles. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 424–429. EDA Consortium, 2012.

[46] R. Chartrand. Nonconvex splitting for regularized low-rank + sparse decomposition. *IEEE Transactions on Signal Processing*, 60(11):5810–5819, 2012.

[47] R. Chartrand and B. Wohlberg. A nonconvex ADMM algorithm for group sparsity with sparse groups. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6009–6013. IEEE, 2013.

[48] C. Chekuri and S. Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM Journal on Computing*, 35(3):713–728, 2005.

[49] Y. Chen and M. Wainwright. Fast low-rank estimation by projected gradient descent: General statistical and algorithmic guarantees. *arXiv preprint arXiv:1509.03025*, 2015.

[50] E. Chu, N. Parikh, A. Domahidi, and S. Boyd. Code generation for embedded second-order cone programming. In *Proceedings of the 2013 European Control Conference*, pages 1547–1552, 2013.

[51] P. Chu and J. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of heuristics*, 4(1):63–86, 1998.

[52] V. Chvátal, W. Cook, and M. Hartmann. On cutting-plane proofs in combinatorial optimization. *Linear Algebra and its Applications*, 114:455–499, 1989.

[53] R. Cohen and L. Katzir. The generalized maximum coverage problem. *Information Processing Letters*, 108(1):15–22, 2008.

[54] C. Colbourn. On testing isomorphism of permutation graphs. *Networks*, 11(1):13–21, 1981.

[55] C. Collins and K. Stephenson. A circle packing algorithm. *Computational Geometry*, 25(3):233–256, 2003.

[56] M. Conforti, G. Cornuejols, and G. Zambelli. *Integer Programming*. Graduate Texts in Mathematics. Springer International Publishing, 2014.

[57] D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *International journal of pattern recognition and artificial intelligence*, 18(03):265–298, 2004.

[58] IBM ILOG CPLEX. User's manual for CPLEX, version 12.1. *International Business Machines Corporation*, 46(53):157, 2009.

[59] J. Crawford and L. Auton. Experimental results on the crossover point in random 3-SAT. *Artificial intelligence*, 81(1):31–57, 1996.

[60] A. Cross, R. Wilson, and E. Hancock. Inexact graph matching using genetic search. *Pattern Recognition*, 30(6):953–970, 1997.

[61] O. Damen, A. Chkeif, and J. Belfiore. Lattice code decoder for space-time codes. *IEEE Communications letters*, 4(5):161–163, 2000.

[62] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410, 1954.

[63] J. David and B. De Moor. The opposite of analytic centering for solving minimum rank problems in control and identification. In *Proceedings of the 32nd IEEE Conference on Decision and Control*, pages 2901–2902. IEEE, 1993.

[64] M. De Santo, P. Foggia, C. Sansone, and M. Vento. A large database of graphs and its use for benchmarking graph isomorphism algorithms. *Pattern Recognition Letters*, 24(8):1067–1079, 2003.

[65] W. Deng and W. Yin. On the global and linear convergence of the generalized alternating direction method of multipliers. *Journal of Scientific Computing*, pages 1–28, 2012.

[66] N. Derbinsky, J. Bento, V. Elser, and J. Yedidia. An improved three-weight message-passing algorithm. *arXiv preprint arXiv:1305.1961*, 2013.

[67] G. Di Pillo and L. Grippo. Exact penalty functions in constrained optimization. *SIAM Journal on control and optimization*, 27(6):1333–1360, 1989.

[68] S. Diamond and S. Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[69] S. Diamond, R. Takapoui, and S. Boyd. A general system for heuristic minimization of convex functions over non-convex sets. *Optimization Methods and Software*, pages 1–29, 2017.

[70] M. Diehl, H. Bock, and J. Schlöder. A real-time iteration scheme for nonlinear optimization in optimal feedback control. *SIAM Journal on control and optimization*, 43(5):1714–1736, 2005.

[71] A. Domahidi, E. Chu, and S. Boyd. ECOS: An SOCP solver for embedded systems. In *Proceedings of the 12th European Control Conference*, pages 3071–3076. IEEE, 2013.

[72] J. Eckstein and D. Bertsekas. On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators. *Mathematical Programming*, 55(1-3):293–318, 1992.

[73] J. Eckstein and W. Yao. Augmented Lagrangian and alternating direction methods for convex optimization: A tutorial and some illustrative computational results. *RUTCOR Research Reports*, 32, 2012.

[74] J. Eckstein and W. Yao. Understanding the convergence of the alternating direction method of multipliers: Theoretical and computational perspectives. *Pacific Journal of Optimization*, 11(4):619–644, 2015.

[75] R. Erdős and A. Rényi. Asymmetric graphs. *Acta Mathematica Hungarica*, 14(3):295–315, 1963.

[76] T. Erseghe. Distributed optimal power flow using ADMM. *IEEE Transactions on Power Systems*, 29(5):2370–2380, 2014.

[77] H. Ferreau, C. Kirches, A. Potschka, H. Bock, and M. Diehl. qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363, 2014.

[78] M. Fiori and G. Sapiro. On spectral properties for graph matching and graph isomorphism problems. *Information and Inference*, 4(1):63–76, 2015.

[79] M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Mathematical Programming*, 104(1):91–104, 2005.

[80] R. Fletcher. An exact penalty function for nonlinear programming with inequalities. *Mathematical Programming*, 5(1):129–150, 1973.

[81] R. Fletcherand and S. Leyffer. Solving mixed-integer nonlinear programs by outer approximation. *Mathematical Programming*, 66(1-3):327–349, 1994.

[82] C. Floudas. *Nonlinear and Mixed-Integer Optimization: Fundamentals and Applications*. Oxford University Press, 1995.

[83] A. Frank and A. Asuncion. University of California, Irvine Machine Learning Repository, 2010.

[84] D. Frick, A. Domahidi, and M. Morari. Embedded optimization for mixed logical dynamical systems. *Computers and Chemical Engineering*, 72:21–33, 2015.

[85] J. Friedman, T. Hastie, and R. Tibshirani. *The Elements of Statistical Learning*, volume 1. Springer series in statistics Springer, Berlin, 2001.

[86] R. Frucht. Graphs of degree three with a given abstract group. *Canadian Journal of Mathematics*, 1:365–378, 1949.

[87] M. Flt and L. Jimbergsson. Using ADMM for hybrid system MPC, 2015. Student Paper.

[88] 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):17–40, 1976.

[89] D. Gao, C. Mi, and A. Emadi. Modeling and simulation of electric and hybrid vehicles. *Proceedings of the IEEE*, 95(4):729–745, 2007.

[90] M. Garey and D. Johnson. Computers and intractability: A guide to NP-completeness, 1979.

[91] M. Garey and D. Johnson. *Computers and Intractability*, volume 29. Freeman, 2002.

[92] A. Geoffrion and R. Marsten. Integer programming algorithms: A framework and state-of-the-art survey. *Management Science*, 18(9):465–491, 1972.

[93] E. Ghadimi, A. Teixeira, I. Shames, and M. Johansson. Optimal parameter selection for the alternating direction method of multipliers (ADMM): Quadratic problems. *IEEE Transactions on Automatic Control*, 60(3):644–658, 2015.

[94] P. Giselsson. Improved fast dual gradient methods for embedded model predictive control. In *International Federation of Automatic Control (IFAC)*, pages 2303–2309, 2014.

[95] P. Giselsson and S. Boyd. Diagonal scaling in Douglas-Rachford splitting and ADMM. In *53rd Annual IEEE Conference on Decision and Control (CDC)*, pages 5033–5039, 2014.

[96] P. Giselsson and S. Boyd. Monotonicity and restart in fast gradient methods. In *53rd Annual IEEE Conference on Decision and Control (CDC)*, pages 5058–5063, 2014.

[97] P. Giselsson and S. Boyd. Preconditioning in fast dual gradient methods. In *53rd Annual IEEE Conference on Decision and Control (CDC)*, pages 5040–5045, 2014.

[98] F. Glover, B. Alidaee, C. Rego, and G. Kochenberger. One-pass heuristics for large-scale unconstrained binary quadratic problems. *European Journal of Operational Research*, 137(2):272–287, 2002.

[99] I. Glover and P. Grant. *Digital Communications*. Pearson Education, 2010.

[100] R. Glowinski and A. Marroco. Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de dirichlet non linéaires. *Revue française d'automatique, informatique, recherche opérationnelle. Analyse numérique*, 9(2):41–76, 1975.

[101] M. Goldberg. The packing of equal circles in a square. *Mathematics Magazine*, pages 24–30, 1970.

[102] R. Gomory et al. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical society*, 64(5):275–278, 1958.

[103] J. Gross. *Combinatorial methods with computer applications*. CRC Press, 2007.

[104] S. Gualandi and F. Malucelli. Exact solution of graph coloring problems via constraint programming and column generation. *INFORMS Journal on Computing*, 24(1):81–100, 2012.

[105] M. Guignard-Spielberg and K. Spielberg. *Integer Programming: State of the Art and Recent Advances*, volume 140. Springer, 2005.

[106] Inc. Gurobi Optimization. Gurobi Optimizer Reference Manual, 2015.

[107] S. Han and O. Mangasarian. Exact penalty functions in nonlinear programming. *Mathematical programming*, 17(1):251–269, 1979.

[108] L. He, C. Han, and W. Wee. Object recognition and recovery by skeleton graph matching. In *IEEE International Conference on Multimedia and Expo*, pages 993–996. IEEE, 2006.

[109] M. Hestenes. Multiplier and gradient methods. *Journal of optimization theory and applications*, 4(5):303–320, 1969.

[110] M. Hifi and R. M'Hallah. A literature review on circle and sphere packing problems: Models and methodologies. *Advances in Operations Research*, 2009, 2009.

[111] H. Hmam. Quadratic optimization with one quadratic equality constraint. Technical report, Electronic Warfare and Radar Division, Defence Science and Technology Organisation (DSTO), Australia, 2010.

[112] D. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, 11(3):555–556, 1982.

[113] D. Hochbaum. Approximating covering and packing problems: Set cover, vertex cover, independent set, and related problems. In *Approximation algorithms for NP-hard problems*, pages 94–143. PWS Publishing Co., 1996.

[114] K. Hoffman, M. Padberg, and G. Rinaldi. Traveling salesman problem. In *Encyclopedia of Operations Research and Management Science*, pages 1573–1578. Springer, 2013.

[115] M. Hong. A distributed, asynchronous and incremental algorithm for nonconvex optimization: An ADMM approach. *IEEE Transactions on Control of Network Systems*, 2017.

[116] M. Hong and Z. Luo. On the linear convergence of the alternating direction method of multipliers. *Mathematical Programming*, pages 1–35, 2012.

[117] M. Hong, Z. Luo, and M. Razaviyayn. Convergence analysis of alternating direction method of multipliers for a family of nonconvex problems. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3836–3840. IEEE, 2015.

[118] H. Hoos and T. Stützle. SATLIB: An online resource for research on SAT. *Sat2000: Highlights of Satisfiability Research in the Year 2000*, pages 283–292, 2000.

[119] J. Hopcroft and J. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 172–184. ACM, 1974.

[120] K. Huang and N. Sidiropoulos. Consensus-ADMM for general quadratically constrained quadratic programming. *IEEE Transactions on Signal Processing*, 64(20):5297–5310, 2016.

[121] T. Ibaraki. Integer programming formulation of combinatorial optimization problems. *Discrete Mathematics*, 16(1):39–52, 1976.

[122] C. Irniger. Graph matching – filtering databases of graphs using Machine Learning techniques, 2005.

[123] J. Jerez, P. Goulart, S. Richter, G. Constantinides, E. Kerrigan, M. Morari, et al. Embedded online optimization for model predictive control at megahertz rates. *IEEE Transactions on Automatic Control*, 59(12):3238–3251, 2014.

[124] R. Jeroslow. Representations of unbounded optimization problems as integer programs. *Journal of Optimization Theory and Applications*, 30(3):339–351, 1980.

[125] R. Jeroslow. Representability in mixed-integer programming: Characterization results. *Discrete Applied Mathematics*, 17(3):223–243, 1987.

[126] R. Jeroslow. Representability of functions. *Discrete Applied Mathematics*, 23(2):125–137, 1989.

[127] R. Jeroslow and J. Lowe. Experimental results on the new techniques for integer programming formulations. *Journal of the Operational Research Society*, pages 393–403, 1985.

[128] R. Jeroslow and J. Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 1(1-4):167–187, 1990.

[129] B. Jiang, S. Ma, and S. Zhang. Alternating direction method of multipliers for real and complex polynomial optimization models. *Optimization*, 63(6):883–898, 2014.

[130] E. Johnson, G. Nemhauser, and M. Savelsbergh. Progress in linear programming-based algorithms for integer programming: An exposition. *INFORMS Journal on Computing*, 12(1):2–23, 2000.

[131] S. Johnson. The NLopt nonlinear-optimization package. `http://ab-initio.mit.edu/nlopt`, 2014.

[132] M. Jünger, T. Lieblingand, D. Naddef, G. Nemhauser, W. Pulleyblank, G. Reinelt, G. Rinaldi, and L. Wolsey. *50 Years of Integer Programming 1958-2008: From the Early Years to the State of the Art*. Springer Science and Business Media, 2009.

[133] T. Junttila and P. Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX)*, 12:135–149, 2007.

[134] T. Junttila and P. Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. http://www.tcs.hut.fi/Software/bliss/benchmarks/index.shtml, April 2009.

[135] T. Junttila and P. Kaski. Conflict propagation and component recursion for canonical labeling. In *Theory and Practice of Algorithms in (Computer) Systems*, pages 151–162. Springer, 2011.

[136] R. Kalman. Identification of noisy systems. *Russian Mathematical Surveys*, 40(4):25–42, 1985.

[137] A. Kandel, H. Bunke, and M. Last. *Applied graph theory in computer vision and pattern recognition*, volume 52. Springer, 2007.

[138] R. Karp. *Reducibility Among Combinatorial Problems*. Springer, 1972.

[139] K. Katayama and H. Narihisa. Performance of simulated annealing-based heuristic for the unconstrained binary quadratic programming problem. *European Journal of Operational Research*, 134(1):103–119, 2001.

[140] P. Kelly et al. A congruence theorem for trees. *Pacific J. Math*, 7(0957):961–968, 1957.

[141] S. Khuller, A. Moss, and J. Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999.

[142] J. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

[143] H. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics (NRL)*, 52(1):7–21, 2005.

[144] A. Land and A. Doig. An automatic method for solving discrete programming problems. In *50 Years of Integer Programming 1958-2008*, pages 105–132. Springer, 2010.

[145] E. Lawler. The traveling salesman problem: A guided tour of combinatorial optimization. *Wiley Series in Discrete Mathematics*, 1985.

[146] E. Lawler and D. Wood. Branch-and-bound methods: A survey. *Operations research*, 14(4):699–719, 1966.

[147] R. Lazimy. Mixed-integer quadratic programming. *Mathematical Programming*, 22(1):332–349, 1982.

[148] J. Lee. A graph-based approach for modeling and indexing video data. In *IEEE International Symposium on Multimedia*, pages 348–355. IEEE, 2006.

[149] S. Leyffer. Integrating SQP and branch-and-bound for mixed-integer nonlinear programming. *Computational Optimization and Applications*, 18(3):295–309, 2001.

[150] D. Li and X. Sun. *Nonlinear Integer Programming*, volume 84. Springer Science and Business Media, 2006.

[151] G. Li and T. Pong. Splitting methods for nonconvex composite optimization. *arXiv preprint arXiv:1407.0753*, 2014.

[152] G. Li and T. Pong. Global convergence of splitting methods for nonconvex composite optimization. *SIAM Journal on Optimization*, 25(4):2434–2460, 2015.

[153] R. Li, D. Zhou, and D. Du. Satisfiability and integer programming as complementary tools. In *Proceedings of the 2004 Asia and South Pacific Design Automation Conference*, pages 879–882, 2004.

[154] Z. Li, S. Zhang, Y. Wang, X. Zhang, and L. Chen. Alignment of molecular networks by integer quadratic programming. *Bioinformatics*, 23(13):1631–1639, 2007.

[155] A. Liavas and N. Sidiropoulos. Parallel algorithms for constrained tensor factorization via alternating direction method of multipliers. *IEEE Transactions on Signal Processing*, 63(20):5450–5463, 2015.

[156] T. Lipp and S. Boyd. Variations and extension of the convex–concave procedure. *Optimization and Engineering*, pages 1–25, 2014.

[157] J. López-Presa and A. Anta. Fast algorithm for graph isomorphism testing. In *International Symposium on Experimental Algorithms*, pages 221–232. Springer, 2009.

[158] J. López-Presa, A. Anta, and L. Chiroque. Conauto-2.0: Fast isomorphism testing and automorphism group computation. *arXiv preprint arXiv:1108.1060*, 2011.

[159] G. Lueker and K. Booth. A linear time algorithm for deciding interval graph isomorphism. *Journal of the ACM (JACM)*, 26(2):183–195, 1979.

[160] E. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. In *21st Annual Symposium on Foundations of Computer Science*, pages 42–49. IEEE, 1980.

[161] V. Lyzinski, D. Fishkind, M. Fiori, J. Vogelstein, C. Priebe, and G. Sapiro. Graph matching: Relax at your own risk. *IEEE transactions on pattern analysis and machine intelligence*, 38(1):60–73, 2016.

[162] S. Magnússon, P. Weeraddana, and C. Fischione. A distributed approach for the optimal power-flow problem based on ADMM and sequential convex approximations. *IEEE Transactions on Control of Network Systems*, 2(3):238–253, 2015.

[163] S. Magnússon, P. Weeraddana, M. Rabbat, and C. Fischione. On the convergence of alternating direction lagrangian methods for nonconvex structured optimization problems. *IEEE Transactions on Control of Network Systems*, 3(3):296–309, 2016.

[164] O. Makela, J. Warrington, M. Morari, and G. Andersson. Optimal transmission line switching for large-scale power systems using the alternating direction method of multipliers. In *Power Systems Computation Conference (PSCC)*, pages 1–6. IEEE, 2014.

[165] J. Mattingley and S. Boyd. Automatic code generation for real-time convex optimization. *Convex Optimization in Signal Processing and Communications*, pages 1–41, 2010.

[166] J. Mattingley and S. Boyd. CVXGEN: A code generator for embedded convex optimization. *Optimization and Engineering*, 13(1):1–27, 2012.

[167] J. Mattingley, Y. Wang, and S. Boyd. Receding horizon control: Automatic generation of high-speed solvers. *IEEE Control Systems Magazine*, 31(3):52–65, 2011.

[168] B. McKay and A. Piperno. Practical graph isomorphism II. *Journal of Symbolic Computation*, 60:94–112, 2014.

[169] R. Merris. Laplacian matrices of graphs: A survey. *Linear Algebra and its applications*, 197:143–176, 1994.

[170] P. Merz and B. Freisleben. Greedy and local search heuristics for unconstrained binary quadratic programming. *Journal of Heuristics*, 8(2):197–213, 2002.

[171] R. Meyer. Integer and mixed-integer programming models: General properties. *Journal of Optimization Theory and Applications*, 16(3-4):191–206, 1975.

[172] R. Meyer. Mixed-integer minimization models for piecewise-linear functions of a single variable. *Discrete Mathematics*, 16(2):163–171, 1976.

[173] R. Meyer. A theoretical and computational comparison of equivalent mixed-integer formulations. *Naval Research Logistics Quarterly*, 28(1):115–131, 1981.

[174] R. Meyer, M. Thakkar, and W. Hallman. Rational mixed-integer and polyhedral union minimization models. *Mathematics of Operations Research*, 5(1):135–146, 1980.

[175] R. Misener and C. Floudas. GloMIQO: Global mixed-integer quadratic optimizer. *Journal of Global Optimization*, 57(1):3–50, 2013.

[176] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of SAT problems. In *Association for the Advancement of Artificial Intelligence (AAAI)*, volume 92, pages 459–465, 1992.

[177] J. Mota, J. Xavier, P. Aguiar, and M. Püschel. Basis pursuit in sensor networks. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 2916–2919. IEEE, 2011.

[178] N. Murgovski, L. Johannesson, J. Sjöberg, and B. Egardt. Component sizing of a plug-in hybrid electric powertrain via convex optimization. *Mechatronics*, 22(1):106–120, 2012.

[179] K. Murty and S. Kabadi. Some NP-complete problems in quadratic and nonlinear programming. *Mathematical Programming*, 39(2):117–129, 1987.

[180] K. Muta, M. Yamazaki, and J. Tokieda. Development of new-generation hybrid system THS II-Drastic improvement of power performance and fuel economy. *SAE transactions*, 113(3):182–192, 2004.

[181] P. Narendra and K. Fukunaga. A branch and bound algorithm for feature subset selection. *IEEE Transactions on Computers*, 100(9):917–922, 1977.

[182] J. Von Neumann. Some matrix inequalities and metrization of metric space. *Tomsk University Review*, 1:286–296, 1937.

[183] L. Ning, T. Georgiou, A. Tannenbaum, and S. Boyd. Linear models based on noisy data and the frisch scheme. *SIAM Review*, 57(2):167–197, 2015.

[184] B. O'Donoghue, G. Stathopoulos, and S. Boyd. A splitting method for optimal control. *IEEE Transactions on Control Systems Technology*, 21(6):2432–2442, 2013.

[185] M. O'kelly. A quadratic integer program for the location of interacting hub facilities. *European Journal of Operational Research*, 32(3):393–404, 1987.

[186] Gurobi Optimization et al. Gurobi optimizer reference manual. *URL: http://www.gurobi.com*, 2012.

[187] D. Oulai, S. Chamberland, and S. Pierre. A new routing-based admission control for MPLS networks. *IEEE Communications Letters*, 11(2):216–218, 2007.

[188] M. Padberg. On the facial structure of set packing polyhedra. *Mathematical Programming*, 5(1):199–215, 1973.

[189] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991.

[190] W. Pan, A. Sootla, and G. Stan. Distributed reconstruction of nonlinear networks: An ADMM approach. *IFAC Proceedings Volumes*, 47(3):3208–3213, 2014.

[191] C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Courier Corporation, 1998.

[192] L. Papageorgiou and E. Fraga. A mixed integer quadratic programming formulation for the economic dispatch of generators with prohibited operating zones. *Electric power systems research*, 77(10):1292–1296, 2007.

[193] N. Parikh and S. Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1(3):123–231, 2013.

[194] N. Parikh and S. Boyd. Block splitting for distributed optimization. *Mathematical Programming Computation*, 6(1):77–102, 2014.

[195] Z. Peng, J. Chen, and W. Zhu. A proximal alternating direction method of multipliers for a minimization problem with nonconvex constraints. *Journal of Global Optimization*, pages 1–18, 2015.

[196] A. Perold. Large-scale portfolio optimization. *Management Science*, 30(10):1143–1160, 1984.

[197] M. Powell. Algorithms for nonlinear constraints that use Lagrangian functions. *Mathematical programming*, 14(1):224–248, 1978.

[198] M. Propato and J. Uber. Booster system design using mixed-integer quadratic programming. *Journal of Water Resources Planning and Management*, 130(4):348–352, 2004.

[199] J. Rocha and T. Pavlidis. A shape analysis model with applications to a character recognition system. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(4):393–404, 1994.

[200] J. Saunderson, V. Chandrasekaran, P. Parrilo, and A. Willsky. Diagonal and low-rank matrix decompositions, correlation matrices, and ellipsoid fitting. *SIAM Journal on Matrix Analysis and Applications*, 33(4):1395–1416, 2012.

[201] E. Scheinerman and D. Ullman. *Fractional Graph Theory: A Rational Approach to the Theory of Graphs*. Courier Corporation, 2011.

[202] C. Schellewald, S. Roth, and C. Schnörr. Evaluation of convex optimization techniques for the weighted graph-matching problem in computer vision. In *Pattern Recognition*, pages 361–368. Springer, 2001.

[203] L. Schizas, A. Ribeiro, and G. Giannakis. Consensus in ad hoc WSNs with noisy links — part I: Distributed estimation of deterministic signals. *IEEE Transactions Signal Processing*, 56(1):350–364, 2008.

[204] T. Schouwenaars, B. De Moor, E. Feron, and J. How. Mixed-integer programming for multi-vehicle path planning. In *European Control Conference*, volume 1, pages 2603–2608. Citeseer, 2001.

[205] T. Sebastian, P. Klein, and B. Kimia. Recognition of shapes by editing their shock graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(5):550–571, 2004.

[206] H. Sedghi, A. Anandkumar, and E. Jonckheere. Multi-step stochastic ADMM in high dimensions: Applications to sparse optimization and matrix decomposition. In *Advances in Neural Information Processing Systems*, pages 2771–2779, 2014.

[207] W. Sharpe. *Portfolio Theory and Capital Markets*. McGraw-Hill, New York, 1970.

[208] H. Sherali and W. Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal on Discrete Mathematics*, 3(3):411–430, 1990.

[209] W. Shi, Q. Ling, K. Yuan, G. Wu, and W. Yin. On the linear convergence of the ADMM in decentralized consensus optimization. *IEEE Transactions on Signal Processing*, 62(7):1750–1761, 2014.

[210] M. Sinnokrot, J. Barry, and V. Madisetti. Embedded Alamouti space-time codes for high rate and low decoding complexity. In *2008 42nd Asilomar Conference on Signals, Systems and Computers*, pages 1749–1753. IEEE, 2008.

[211] A. Sluis. Condition numbers and equilibration of matrices. *Numerische Mathematik*, 14(1):14–23, 1969.

[212] K. Smedley and S. Cuk. One-cycle control of switching converters. *IEEE transactions on Power Electronics*, 10(6):625–633, 1995.

[213] A. Snook, G. Schoenebeck, and P. Codenotti. Graph isomorphism and the lasserre hierarchy. *arXiv preprint arXiv:1401.0758*, 2014.

[214] J. Spingarn. Applications of the method of partial inverses to convex programming: Decomposition. *Mathematical Programming*, 32(2):199–223, 1985.

[215] K. Stephenson. *Introduction to circle packing: The theory of discrete analytic functions*. Cambridge University Press, 2005.

[216] R. Stubbs and S. Mehrotra. A branch-and-cut method for 0-1 mixed convex programming. *Mathematical Programming*, 86(3):515–532, 1999.

[217] R. Takapoui and S. Boyd. Linear programming heuristics for the graph isomorphism problem. *arXiv preprint arXiv:1611.00711*, 2016.

[218] R. Takapoui and H. Javadi. Preconditioning via diagonal scaling. *arXiv preprint arXiv:1610.03871*, 2016.

[219] R. Takapoui, N. Moehle, S. Boyd, and A. Bemporad. A simple effective heuristic for embedded mixed-integer quadratic programming. *International Journal of Control*, pages 1–23, 2017.

[220] M. Tawarmalani and N. Sahinidis. Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Mathematical Programming*, 99(3):563–591, 2004.

[221] M. Tawarmalani and N. Sahinidis. A polyhedral branch-and-cut approach to global optimization. *Mathematical Programming*, 103(2):225–249, 2005.

[222] R. Tibshirani. Lecture notes in modern regression, March 2013.

[223] G. Tinhofer. A note on compact graphs. *Discrete Applied Mathematics*, 30(2):253–264, 1991.

[224] F. Ullmann. FiOrdOs: A Matlab toolbox for C-code generation for first order methods. Master's thesis, ETH Zurich, 2011.

[225] S. Umeyama. An eigendecomposition approach to weighted graph matching problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(5):695–703, 1988.

[226] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.

[227] J. Vielma, S. Ahmed, and G. Nemhauser. A lifted linear programming branch-and-bound algorithm for mixed-integer conic quadratic programs. *INFORMS Journal on Computing*, 20(3):438–450, 2008.

[228] E. Viterbo and J. Boutros. A universal lattice code decoder for fading channels. *IEEE Transactions on Information Theory*, 45(5):1639–1642, 1999.

[229] J. Vogelstein, J. Conroy, L. Podrazik, S. Kratzer, E. Harley, D. Fishkind, R. Vogelstein, and C. Priebe. Large (brain) graph matching via fast approximate quadratic programming. *arXiv preprint arXiv:1112.5507*, 2011.

[230] A. Waechter, C. Lairdl, F. Margot, and Y. Kawajir. Introduction to IPOPT: A tutorial for downloading, installing, and using IPOPT, 2009.

[231] B. Wahlberg, S. Boyd, M. Annergren, and Y. Wang. An ADMM algorithm for a class of total variation regularized estimation problems. *IFAC Proceedings Volumes*, 45(16):83–88, 2012.

[232] D. Wang, H. Lu, and M. Yang. Online object tracking with sparse prototypes. *IEEE Transactions on Image Processing*, 22(1):314–325, 2013.

[233] F. Wang, W. Cao, and Z. Xu. Convergence of multi-block Bregman ADMM for nonconvex composite problems. *arXiv preprint arXiv:1505.03063*, 2015.

[234] F. Wang, Z. Xu, and H. Xu. Convergence of alternating direction method with multipliers for non-convex composite problems. *arXiv preprint arXiv:1410.8625*, 2014.

[235] Y. Wang and S. Boyd. Fast model predictive control using online optimization. *IEEE Transactions on Control Systems Technology*, 18(2):267–278, 2010.

[236] Y. Wang, W. Yin, and J. Zeng. Global convergence of ADMM in nonconvex nonsmooth optimization. *arXiv preprint arXiv:1511.06324*, 2015.

[237] Z. Wang, S. Fang, D. Gao, and W. Xing. Global extremal conditions for multi-integer quadratic programming. *Journal of Industrial and Management Optimization*, 4(2):213–225, 2008.

[238] F. Wen, Y. Yang, P. Liu, R. Ying, and Y. Liu. Efficient ℓ_q minimization algorithms for compressive sensing based on proximity operator. *arXiv preprint arXiv:1506.05374*, 2015.

[239] L . Wolsey. *Integer Programming*, volume 42. Wiley New York, 1998.

[240] L. Wolsey and G. Nemhauser. *Integer and Combinatorial Optimization*. John Wiley & Sons, 2014.

[241] S. Wright and J. Nocedal. *Numerical Optimization*, volume 2. Springer New York, 1999.

[242] Y. Xu, W. Yin, Z. Wen, and Y. Zhang. An alternating direction algorithm for matrix completion with nonnegative factors. *Frontiers of Mathematics in China*, 7(2):365–384, 2012.

[243] L. Yang, T. Pong, and X. Chen. Alternating direction method of multipliers for nonconvex background/foreground extraction. *arXiv preprint arXiv:1506.07029*, 2015.

[244] S. You and Q. Peng. A non-convex alternating direction method of multipliers heuristic for optimal power flow. In *2014 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pages 788–793. IEEE, 2014.

[245] R. Zhang and J. Kwok. Asynchronous distributed ADMM for consensus optimization. In *Proceedings of the International Conference on Machine Learning*, pages 1701–1709, 2014.

[246] Y. Zhang. Restricted low-rank approximation via ADMM. *arXiv preprint arXiv:1512.01748*, 2015.