Structure Exploitation in Mixed-Integer Optimization

with Applications to Energy Systems

Zur Erlangung des akademischen Grades eines

DOKTOR-INGENIEURS

von der KIT-Fakultät für Informatik,
des Karlsruher Instituts für Technologie (KIT)

genehmigte

DISSERTATION

von

M.Sc. Alexander Murray
geb. in Montreal, Canada

Tag der mündlichen Prüfung: 23.10.2020

Hauptreferent: Prof. Dr.-Ing. Veit Hagenmeyer
Korreferent: Prof. Dr.-Ing. Timm Faulwasser

Abstract

The goal of this thesis is to develop new numerical methods for mixed-integer optimization problems and achieve improved speed and scalability. This is done by exploiting common problem structures, such as separability or turnpike behaviour. Methods capable of exploiting these structures have already been developed in the realms of distributed optimization and optimal control, however they are not directly applicable to the mixed-integer case. To be able to make use of distributed computing resources to solve mixed-integer problems, new methods are required. To this end, several extensions of existing methods as well as novel techniques for mixed-integer optimization are presented. Benchmark problems drawn from power and energy systems are used to demonstrate that the presented methods can lead to faster runtimes and can allow for solution to large problems that are otherwise impossible to solve in a centralized manner. The novelties presented in this thesis are as follows:

- Extension of the Augmented Lagrangian Alternating Direction Inexact Newton distributed optimization algorithm applicable to mixed-integer optimization.
- A new partially distributed mixed-integer optimization algorithm based on outer approximation.
- A new distributed mixed-integer optimization algorithm based on branch and bound.
- A first investigation into turnpike properties in mixed-integer optimal control and a specialized algorithm for solving this class of problems.
- A new branch and bound heuristic that more efficiently exploits *a priori* problem information than current warm-starting techniques.

Finally, it is shown that the results of the presented distributed mixed-integer optimization algorithms can heavily depend on how a given problem is parti-

tioned. To this effect, an investigation into partitioning methods for distributed optimization is also given.

Zusammenfassung

Das Ziel dieser Arbeit ist neue numerische Methoden für gemischt-ganzzahlige Optimierungsprobleme zu entwickeln um eine verbesserte Geschwindigkeit und Skalierbarkeit zu erreichen. Dies erfolgt durch Ausnutzung gängiger Problemstrukturen wie separierbarkeit oder Turnpike-eigenschaften. Methoden, die diese Strukturen ausnutzen können, wurden bereits im Bereich der verteilten Optimierung und optimalen Steuerung entwickelt, sie sind jedoch nicht direkt auf gemischt-ganztägige Probleme anwendbar.

Um verteilte Rechenressourcen zur Lösung von gemischt-ganzzahligen Problemen nutzen zu können, sind neue Methoden erforderlich. Zu diesem Zweck werden verschiedene Erweiterungen bestehender Methoden sowie neuartige Techniken zur gemischt-ganzzahligen Optimierung vorgestellt.

Benchmark-Probleme aus Strom- und Energiesystemen werden verwendet, um zu demonstrieren, dass die vorgestellten Methoden zu schnelleren Laufzeiten führen und die Lösung großer Probleme ermöglichen, die sonst nicht zentral gelöst werden können. Die vorliegende Arbeit enthält die folgenden Beiträge:

- Eine Erweiterung des Augmented Lagrangian Alternating Direction Inexact Newton-Algorithmus zur verteilten Optimierung für gemischt-ganzzahlige Probleme.
- Ein neuer, teilweise-verteilter Optimierungsalgorithmus für die gemischt-ganzzahlige Optimierung basierend auf äußeren Approximationsverfahren.
- Ein neuer Optimierungsalgorithmus für die verteilte gemischt-ganzzahlige Optimierung, der auf branch-and-bound Verfahren basiert.
- Eine erste Untersuchung von Turnpike-Eigenschaften bei Optimalsteuerungsproblemen mit gemischten-Ganzzahligen Entscheidungsgrößen und ein spezieller Algorithmus zur Lösung dieser Probleme.

- Eine neue Branch-and-Bound Heuristik, die a priori Probleminformationen effizienter nutzt als aktuelle Warmstarttechniken.

Schließlich wird gezeigt, dass die Ergebnisse der vorgestellten Optimierungsalgorithmen für verteilte gemischt-ganzzahlige Optimierung stark Partitionierungsabhängig sind. Zu diesem Zweck wird auch eine Untersuchung von Partitionierungsmethoden für die verteilte Optimierung vorgestellt.

Table of Contents

List of Abbreviations

ADMM	Alternating Direction Method of Multipliers
ALADIN	Augmented Lagrangian based Alternating Direction Inexact Newton method
B&B	Branch and Bound
GJK	Gilbert-Johnson-Keerthi
IEEE	Institute of Electrical and Electronics Engineers
IP	Integer Program
IPOPT	Interior Point Optimizer
KaFFPa	Karlsruhe Fast Flow Partitioner
KIT	Karlsruher Institut für Technologie
KKT	Karush-Kuhn-Tucker
LP	Linear Program
MICP	Mixed-Integer Convex Program
MILP	Mixed-Integer Linear Program
MINLP	Mixed-Integer Nonlinear Program
MIOCP	Mixed-Integer Optimal Control Problem
MIP	Mixed-Integer Program
MIQP	Mixed-Integr Quadratic Program

NLP	Nonlinear Program
OA	Outer Approximation
OCP	Optimal Control Problem
OPF	Optimal Power Flow
PaDOA	Partially Distributed Outer Aproximation
PSO	Particle Swarm Optimization
PV	Photovoltaic
QP	Quadratic Program
RPD	Reactive Power Dispatch
RTS	Reactive Tabu Search
SC	Spectral Clustering
SQP	Sequential Quadratic Programming
TCL	Therostatically Controlled Load

1 Introduction

1.1 Motivation

All around us, and at every moment, optimization is taking place. Every living thing must make choices and decide how to behave in any given situation. Our perception of reality forms a kind of model, complete with both consciously and unconsciously constructed constraints. To fulfil whatever objectives we may have, we behave according to how our model predicts success. If an unexpected outcome occurs, then the model is adjusted and life goes on. Mathematical modelling and optimization follows a similar process, albeit in a more precise and rigorous way.

The process of mathematical modelling begins with some phenomenon to be understood. The purpose may be to predict the outcome of a chemical reaction, how many people to interview before hiring, or the most cost-effective way of running a power grid. Once the scope of the phenomenon has been determined, the governing dynamics must be identified. These may be something physical, such as Kirchoff's laws, or a quantization of a qualitative value, such as the suitability of a job applicant. In any case, the key here is to describe some aspect of reality as accurately as possible using mathematics. Once the variables, and the relationships between variables, have been established, then the model can be used to make predictions on the outcome of given decisions. Such predictions often come in the form of achieving some set objectives, along with some constraints on how those objectives may be achieved. Carrying on with the previously used examples, this may be the minimization of generator costs, or to hire the best applicant within a certain time window. Regardless of what is chosen, the end result is an optimization problem. If the model is too large or too detailed, then the resulting optimization problem may be intractable. Indeed, we could never act if we were always conscious of every molecule or bacterium in our surroundings. Rather, we require an approximation. Such approximations and reformulations are sometimes required in modelling as

well, but when constructing a model it is important not to lose too much accuracy and become too detached from reality. In some cases, the phenomena to be modelled are so structurally complex that it becomes too difficult—or outright impossible—to create something simpler and still preserve the necessary structure. In these cases, improved methods for optimization are required which can handle this complexity and return a solution. With such solutions, we may better understand and react to the world around us.

Numerical algorithms can be traced back centuries to the ancient Greeks and Egyptians who sought solutions to algebraic and geometric problems [Cla99a, Hor72]. In the 17^{th} century, the well known "Newton's method" became an early iterative optimization algorithm from which countless other algorithms have been based [Wal85,Gau09,NW06]. With the advent of modern computing, there has been a massive increase in algorithmic power due to both improvements in computer processors and algorithms. As reported in [Bix12], between the years 1988 and 2002 there was a machine improvement of a factor of about 1,600 but an algorithmic improvement[1] of about a factor of 3,300, resulting in a total improvement factor of $1,600 \times 3,300 = 5,280,000$. For an updated example, the "AMD EPYC Rome" has a transistor count of 32,000,000,000 while the best transistor count in 1988 was just 7,500,000. Thus, we can infer a machine speed-up of about 4,266. In that same time, there has been an algorithmic speed-up of over 29,000 [Bix12]. This means that a problem that would have taken 32 years to be solved in 1988 only takes about 8 seconds now! If this process is to continue, more advanced methods will be required to tackle ever more complicated problems.

One of the most complicated class of problems are those that involve multiple discrete decisions, such as switches, gears, or people. These problems are known as Integer Programs (IPs), or Mixed-Integer Programs (MIPs) if they also include real-valued variables, and are known to generally be NP-complete [BL12]. The difficulty in solving MIPs is primarily due to the so-called "curse of dimensionality," where increasing problem size leads to combinatorial explosions in the number of combinations of possible decisions. The enormous number of possible combinations makes verifying the optimal-

[1] Calculated by applying successive versions of CPLEX to a set of benchmarking problems. The largest improvement by far came in CPLEX v6 when many new ideas from academia were implemented in the solver.

ity of any given combination quite difficult. For example, consider the problem of assigning 70 people to 70 jobs where each person has a different aptitude with each of the jobs. As the people and jobs are indivisible, the problem is naturally formulated as an integer program. There are more possible solutions to this problem than there are particles in the universe, and thus testing every possible solution is impractical [LRKS91].

However, all is not lost. Even though the complexity of MIPs increases exponentially with problem size, there are certain techniques which can be used to mitigate this issue. For example, by linearizing the job assignment problem and using the simplex method, the vast majority of potential solutions can be ignored and the optimal solution quickly and efficiently obtained [Hun83]. Techniques that can effectively eliminate large swathes of the solution space are the key to solving MIPs, and can be done in several ways. First, part of the constraint set could quite literally be cut off using so-called "cutting planes." Alternatively, by constructing successively better upper and lower bounds on the optimal objective value, one can establish the optimality of a candidate solution. This process is known as Branch and Bound (B&B). Cutting planes, B&B, and other MIP methods will be discussed in more detail in Section 2.2.

Despite the many advances in mixed-integer programming, there are still many problems which remain intractable and many applications where faster solution methods are needed [Iba94, YKF+00, mip18, BDM03]. Two examples of problems in power systems that are difficult to solve at medium and large scales are the reactive power dispatch problem, and battery scheduling. The former is described in detail in Section 3.1.3, and the latter is described in Section 3.1.2. It is shown that large scale instances of both of these problems are difficult to solve with standard techniques. The poor scalability of problems such as battery scheduling and reactive power dispatch are the main focus and motivation of this thesis. To this end, several new methods and algorithms are developed that allow for faster solution to larger MIPs than was previously possible.

1.2 Contributions

Many of the problems in power and energy involve a modular system of components which can be decomposed into several interconnected subsystems.

Such a decomposition can then be exploited by distribution and parallelization. For problems with a temporal component, there is typically a sparse block-diagonal structure that links the variables of subsequent time steps. In either case, this structure can be exploited in order to gain some speed up and/or to attain tractability through the solution of a collection of smaller subproblems.

Problem decomposition and distribution is not new and there are already a number of well-established methods that are available. However, as shown in Sections 2.1.2 and 4.1, these methods are almost all designed for real-valued problems and can perform quite poorly when applied to an MIP. As such, one of the main contributions of this thesis is the marriage of Mixed-Integer Programming and Distributed Optimization. The result of which are three algorithms for distributed mixed-integer optimization, which are described in Sections 3.1, 3.2, and 3.3. Each of these sections are based on work published in [MEHF18], [MFH18], [MHF18], [MFH+20], [MH20b], and [MH20a]. There is no "silver bullet" to distributed mixed-integer optimization, and each of the presented algorithms have different properties and different performance for different problems.

One significant factor that affects the performance of distributed methods is the partitioning of the given optimization problem. Often, the choice of problem partitioning is fully (or at least partially) open. As such, one key question in distributed optimization is "how ought I decompose the problem at hand for a given distributed optimization algorithm?" Some background on this problem is given in Section 4.1, and some first steps towards problem partitioning for distributed mixed-integer optimization are given in Section 4.1.1. Section 4.1 is primarily based on the work published in [KMS+20] and [MKS+20].

For problems with temporal decomposability, it is tempting to consider distributed optimization as an option. However, Section 4.2 presents two alternatives for this case based on the work published in [MHF20] and [FM20]. One, is an algorithm for mixed-integer optimal control problems with turnpikes. The details of the algorithm are found in Section 4.2.2. The other, is a method which takes advantage of *a priori* information in a new way in order to more efficiently solve MIPs. This method is presented in Section 4.2.3. Both methods bring concepts from existing turnpike theory to mixed-integer programming to more effectively solve certain mixed-integer optimal control problems.

1.3 Outline

This dissertation is organized into five chapters. Chapter 2 focuses on the state of the art in constrained optimization and is structured as follows: in Section 2.1, the core concepts and techniques used in mathematical programming of continuous problems are introduced. These problems often arise as subproblems of MIP algorithms and come in a variety of forms. The various problem subclasses are introduced, and some illustrative examples are given to help distinguish them. The methods used to solve such problems are described in Sections 2.1.1 and 2.1.2.

Chapter 3 begins with a presentation of a distributed algorithm for MIPs. Its optimality and convergence properties are examined and its performance is tested on two different case studies: reactive power dispatch, and battery scheduling. In Section 3.2, a partially distributed optimization algorithm is presented. In contrast to the algorithm of Section 3.1, it has stronger convergence and optimality guarantees but is not suited for certain problem classes or contexts where full distribution is required. Its performance is also benchmarked using two case studies: thermostatically controlled loads, and battery scheduling. In Section 3.3 a fully distributed algorithm is presented which has favourable convergence and optimality guarantees. Two case studies are used to compare its performance against that of alternative methods: a suite of MIQPs, and battery scheduling.

While Chapter 3 is primarily concerned with distributed approaches to mixed integer programming, Chapter 4 consists of two parts. First, in Section 4.1 some insight is given into how problems ought to be partitioned for input into distributed methods. Two optimization problems involving power grids are used as examples, and it is shown that proper partitioning of these problems plays an important role in the performance of the distributed algorithm which is applied. Section 4.2 goes beyond exploitation of modularity in MIPs and presents some methods for taking advantage of certain temporal structures in mixed-integer optimal control problems. These methods are compared and several case studies are used as benchmark examples.

Chapter 5 gives final conclusions on the contributions presented throughout the thesis and lists a number of ways in which the work may be extended. All of the algorithms presented in this thesis are implemented in MATLAB R2019a

relying on CasADi v3.5.0 [AGH+19], and all numerical experiments are run on a 2.9GHz Intel Core i5-4460S CPU with 8GB of RAM.

2 State of the Art

The scope of this chapter is to give an overview of the algorithms and techniques commonly used to numerically solve optimization problems. This begins with methods for continuous optimization in Section 2.1, which are categorized further into centralized (Section 2.1.1) and distributed algorithms (Section 2.1.2). In Section 2.2 the methods most commonly used for mixed problems involving both discrete and continuous variables are given.

2.1 Nonlinear Programming

Typically, the goal of an optimization problem is to minimize an objective function $f : \mathbb{R}^n \to \mathbb{R}$ over a set $\mathcal{X} \subseteq \mathbb{R}^n$. Mathematically, this can be expressed as:

$$\begin{aligned} &\min_{x} \ f(x) \\ s.t. \quad &x \in \mathcal{X} \subseteq \mathbb{R}^n. \end{aligned} \tag{2.1}$$

One can classify optimization problems into two main categories: constrained and unconstrained. An unconstrained optimization problem is simply (2.1) with $\mathcal{X} = \mathbb{R}^n$. If the function f is twice differentiable, then the minimizer of an unconstrained problem can be solved by first identifying the set of points with zero gradient:

$$\Xi = \{x \in \mathbb{R}^n | \nabla f = 0\},$$

along with the points Ω where ∇f is not defined. Together, Ξ and Ω are the *critical points* of f. The *global minimum* of the unconstrained problem can be found by evaluating $\min\{f(x)|x \in \Xi \cup \Omega\}$. It should be noted that while finding the global minimum is often the goal, it is sometimes sufficient to find a *local minimum*.

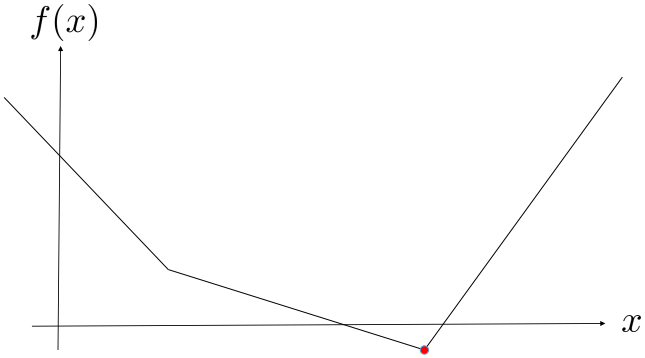

Figure 2.1: An example of a linear program. The red dot indicates the optimal solution.

Definition 1. *A point $x^\star \in \mathcal{X}$ for an optimization problem* (2.1) *with objective function $f : \mathcal{X} \to \mathbb{R}$ is said to be locally optimal if $\exists \epsilon > 0$ such that $f(x^\star) \leq f(x)$ for all $x \in \{y \in \mathcal{X} | \left\| x^\star - y \right\| < \epsilon\}$.*

Now consider the more general case of Problem (2.1) where $\mathcal{X} \neq \mathbb{R}^n$. In this case, the critical points of f are no longer necessarily potential solutions to (2.1). Solving such constrained optimization problems requires specialized techniques, depending on their structure. If f is linear and $\mathcal{X}$ is a polytope, then the problem is said to be a Linear Program (LP), one of the first constrained optimization problems to be studied [Dan48, Dan81]. A simple LP is depicted in Figure 2.1.

Convexity is another very important structural property, and it is defined formally in Definition 2.

Definition 2. *A set C is defined as convex if $\forall x, y \in C$ and $t \in [0, 1]$ it holds that $(1 - t)x + ty \in C$. That is, for all points x and y in C, the line segment connecting x and y is also in C. A function is said to be convex if its epigraph forms a convex set.*

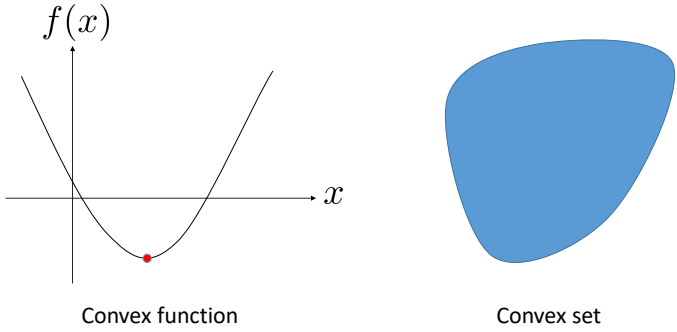

Figure 2.2: An example of a convex function and a convex set. The red dot indicates the optimal solution.

Remark 1. *A related notion is that of strict convexity. A set C is said to be strictly convex if $\forall x, y \in C$ it holds that $x(1-t) + yt$ belongs to the interior of C for all $t \in (0, 1)$.*

A example of a convex set and a convex function are depicted in Figure 2.2, and an important implication of Definition 2 is given in Corollary 1.

Corollary 1. *The intersection of two convex sets is also convex. Likewise, if the functions $f_1 : \mathcal{X} \to \mathbb{R}$ and $f_2 : \mathcal{X} \to \mathbb{R}$ are convex then so too is $f_1 + f_2$.*

Remark 2. *The union of two convex sets is not necessarily convex. A trivial example of this is given by $C_1 \cup C_2$ for convex sets $C_1 \neq \emptyset$ and $C_2 \neq \emptyset$ such that $C_1 \cap C_2 = \emptyset$.*

If both the objective function f and feasible set $\mathcal{X}$ are convex, then the problem is said to be a *convex program*, one of two main categories of Nonlinear Programs (NLPs). Likewise, if f and $\mathcal{X}$ are strictly convex then the problem is said to be a *strictly convex program*. One of the key properties of convex programs is that their local solutions are also global solutions, and thus they

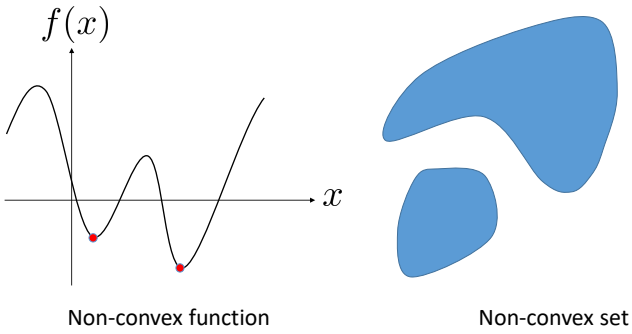

Figure 2.3: An example of a non-convex function and a non-convex set. The red dots indicate some locally optimal solutions.

have a unique minimum [BV04]. The second category of NLPs are non-convex programs, which are optimization problems that have non-convexities in either the objective function f or the feasible set $\mathcal{X}$. They comprise a more challenging class of constrained optimization problems as they can potentially have many local minima and thus local optimization methods are typically insufficient to obtain a globally optimal solution. Figure 2.3 illustrates a non-convex objective function and a non-convex feasible set. The non-convexity of the set in Figure 2.3 stems both from being disconnected and the "bend" in the upper subset.

Note that if an optimization problem has a disconnected feasible set, such as the one shown in Figure 2.3 then that problem is said to be *disconnected*. It can sometimes be helpful to model such a feasible set using integer-valued variables, which results in a Mixed-Integer Program (MIP). Details on the state of the art and open problems in mixed-integer programming is given in Section 2.2.

A key concept in the numerical optimization of real-valued problems is the concept of duality. As its name suggests, it offers a related problem which can sometimes be easier to solve than the original (primal) problem. The Lagrange

dual problem[1] is constructed by taking problem constraints into account within the objective function. For a problem of the form

$$\begin{aligned} P = \min_{x} \quad & f(x) \\ s.t. \quad & g(x) \leq 0 \\ & h(x) = 0 \end{aligned} \tag{2.2}$$

its Lagrange dual is

$$\begin{aligned} D = \max_{\mu,\lambda} \inf_{x} \quad & f(x) + \mu^\top g(x) + \lambda^\top h(x) \\ s.t. \quad & \mu \geq 0, \end{aligned} \tag{2.3}$$

where μ and λ are the *Lagrange multipliers* for the constraints $g(x) \leq 0$ and $h(x) = 0$, respectively. Intuitively, Lagrange multipliers can be thought of as "shadow prices" or sensitivity with respect to certain constraints [Ber06]. If the primal optimal objective P and dual optimal objective D are equal, then *strong duality* is said to hold. These concepts are crucial for establishing the Karush-Kuhn-Tucker (KKT) optimality conditions.

Before stating the conditions, first recall Problem (2.1) with objective f and feasible set $\mathcal{X} = \{x | g(x) \leq 0, h(x) = 0\}$, where g and h are denoted in vector notation as $g = [g_1, \ldots, g_m]^\top$ and $h = [h_1, \ldots, h_\ell]^\top$, respectively. The KKT conditions are commonly used in the verification of the optimality of candidate solutions to purely real-valued NLPs and the following provides a necessary condition for local optimality of a point $x^\star$.

[1] Which for brevity, is may be referred to as simply the *dual problem*.

Theorem 1. *Let f,g, and h be continuously differentiable at a point $x^\star$ such that $\nabla g_i(x^\star)$ and $\nabla h_j(x^\star)$ are linearly independent for all i and j such that $g_i(x^\star) = 0$ and $h_j(x^\star) = 0$. Then, for all $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, \ell\}$ there exist Lagrange multipliers μ_i and λ_j such that:*

$$\nabla f(x^\star) + \sum_{i=1}^{m} \mu_i \nabla g_i(x^\star) + \sum_{j=1}^{\ell} \lambda_j \nabla h_j(x^\star) = 0 \tag{2.4}$$

$$g_i(x^\star) \leq 0, \forall i \in \{1, \ldots, m\} \tag{2.5}$$

$$h_i(x^\star) = 0, \forall i \in \{1, \ldots, \ell\} \tag{2.6}$$

$$\mu_i \geq 0, \forall i \in \{1, \ldots, m\} \tag{2.7}$$

$$\mu_i g_i(x^\star) = 0, \forall i \in \{1, \ldots, m\} \tag{2.8}$$

Equations (2.4) – (2.8) collectively form the first-order KKT optimality conditions. Equation (2.4) is known as the "stationarity" condition and gives a notion of optimality with respect to constraints. Equations (2.5) and (2.6) ensure primal feasibility of $x^\star$, while Equation (2.7) ensures dual feasibility.[2] Complementary slackness of the Lagrange multipliers μ and the inequality constraints g is upheld by Equation (2.8).

For convex optimization problems, the conditions (2.4) – (2.8), along with some regularity conditions,[3] are sufficient for verifying local optimality [NW06, BV04]. For non-convex programs that is no longer the case and an additional second-order sufficient condition is required, such as:

$$s^\top \nabla^2_{xx} L(x^\star, \lambda^\star, \mu^\star) s \geq 0 \tag{2.9}$$

where $L(x, \lambda, \mu) = f(x) + \sum_{i=1}^{m} \mu_i g_i(x) + \sum_{j=1}^{\ell} \lambda_j h_j(x)$ is the Lagrangian of Problem 2.1, and $s \neq 0$ is a vector which satisfies

$$\left[\nabla_x g_i(x^\star), \nabla_x h_j(x^\star)\right]^\top s = 0$$

for the constraints g_i such that $g_i(x^\star) = 0$.

[2] That is, the feasibility of the dual problem.

[3] Such as Slater's condition: $g(x^\star) < 0, h(x^\star) = 0$, or the linear independence of the gradient of the active constraints at $x^\star$.

The goal of many optimization algorithms is to obtain a point which satisfies the KKT conditions (2.4)-(2.8), and sometimes the second order condition (2.9) as well. We shall categorize these algorithms as 1) centralized, which solve (2.1) with a single shared memory, and 2) distributed, which make use of problem decomposability. Problem decomposition can allow for solution with distributed computing resources, although it is not required. A brief overview of some common centralized optimization methods is given in Section 2.1.1, followed by distributed algorithms in Section 2.1.2.

2.1.1 Centralized Algorithms

For the most basic class of optimization problem, the LP, there is a plethora of available algorithms [Vai89, Kar84, Kha79, Van15]. The simplex method being one of the most common. The main idea behind the simplex method is to represent the problem as a matrix "tableau," which is solved via Gaussian elimination. In effect, each iteration (corresponding to a pivot in Gaussian elimination) navigates the vertices of the feasible set of the LP until no better point is available. This method is known to be polynomial time for *most* problems, although worst case problems can be constructed which have an exponential runtime [KM70].

In addition to the simplex method, the "Interior Point Method" is also commonly used for solving LPs [CLS19, IT02, LS15]. Modern implementations of interior point are generally as fast as those of simplex methods [GT96], with some differences arising depending on the specific structure of the LP. The main idea is to approximate the constraints with a barrier function and then solve the resulting unconstrained optimization problem. This can be done by iteratively solving a system of equations with a variant of Newton's method [PW00]. An advantage of interior point is that it can also efficiently solve convex optimization problems [BV04]. However, interior point is not alone in this regard.

Sequential Quadratic Programming (SQP) is an iterative method for problems of the form (2.1), where f, g, and h are twice differentiable. The main idea behind SQP is to iteratively construct successive quadratic approximations of the

original problem. Each iteration of SQP contains the following optimization subproblem:

$$\begin{aligned} \min_{d} \; & \nabla_x f(x_k)^\top d + 1/2 d^\top \left(\nabla^2_{x,x} f(x_k) - \mu g(x_k) - \lambda h(x_k) \right) d \\ s.t. \; & g(x_k) + \nabla_x g(x_k)^\top d \leq 0 \\ & h(x_k) + \nabla_x h(x_k)^\top d = 0 \end{aligned}$$

where μ and λ are Lagrange multipliers, x_k is the current iterate, and d is the search direction. One can think of SQP as being an extension to Newton's method for inequality constrained problems. Indeed, for purely equality constrained problems, SQP is equivalent to applying Newton's method to the first-order KKT conditions [NW06]. Often, the Hessian matrices in each SQP step are approximated to reduce computational load and improve runtime, however this can affect the convergence of the algorithm [Ulb04].

To solve Problem (2.1) to global optimality, even more advanced methods are needed. In this case, there are two main options: spatial branch and bound or heuristic methods. Spatial branch and bound seeks to solve (2.1) to global optimality by iteratively splitting the feasible set $\mathcal{X}$ into successively smaller pieces and obtaining local solutions on each of the resultant subproblems. This method shares many properties with the B&B method for MIPs, which will be covered in detail in Section 2.2.1. Alternatively, one may attempt to use a heuristic such as genetic algorithms, particle swarm optimization, reactive tabu search, etc. [BNKF98, EK95, BT95]. Each of these techniques attempts to overcome the difficulties of non-convexity and non-differentiability in a different way, by sacrificing optimality and convergence guarantees in exchange for a lighter computational burden.

Although non-convex NLPs remain a very challenging problem class, some large convex NLPs can be solved relatively quickly. For example the "nql180" benchmark problem [Mit03] has 162,001 decision variables and 130,080 constraints but is solved in approximately 3 seconds by the commercial solver MOSEK [ApS19]. However, this is not always the case. When an NLP is too large to be solved on a single machine, or extra speed is required, then distributed optimization may be needed. Background on the state of the art of distributed optimization for NLPs is given in Section 2.1.2.

2.1.2 Distributed Algorithms

It should first be noted that there are significant differences between the terms distributed optimization, decentralized optimization, distributed computing, and parallel computing, and in fact some of these terms are defined differently by different authors [KM08, CYS18]. Herein, these terms shall be defined as follows:

Definition 3 (Decentralized Optimization Algorithm)**.** *A decentralized optimization algorithm is an algorithm which solves an optimization problem and is completely parallelizable.*

Definition 4 (Distributed Optimization Algorithm)**.** *A distributed optimization algorithm is an algorithm which alternates between solving decoupled subproblems and a centralized coupling problem.*

Definition 5 (Parallel Computing)**.** *Parallel computing consists of a collection of processors working in parallel which all have access to a single shared memory.*

Definition 6 (Distributed Computing)**.** *Distributed computing consists of a collection of processors and a communication network between them, where each processor only has access to its own memory resources.*

Each of these concepts is depicted visually in Figure 2.4.

While the subproblems solved by both distributed and decentralized algorithms can make use of distributed computing resources, it is not necessary. For example, if the problem at hand is too large to solve centrally but no other computing resources are available, then distributed optimization can still be used. In this case each of the subproblems are solved sequentially rather than in parallel.

A lot of research on NLPs has gone into how to exploit full or partial separability in problem structure [Ers14, HBO14, KAGB04, BFG+18, GHT16a, MDS+17]. A problem is said to be fully separable if its objective function f and feasible set $\mathcal{X}$ can be decomposed into

$$f = \sum_{i=1}^{N} f_i(x_i)$$

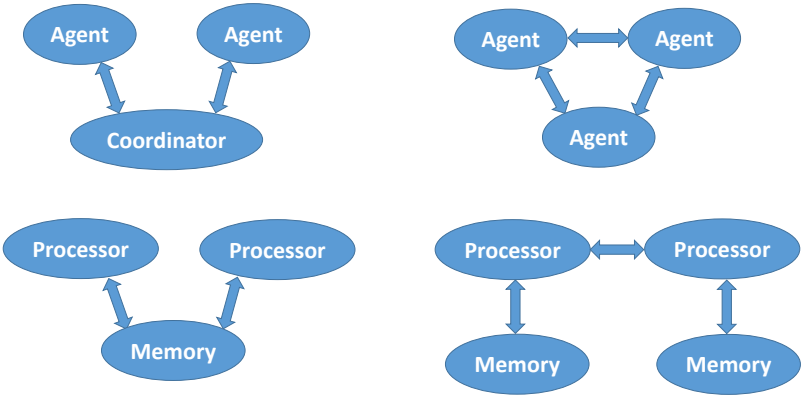

Figure 2.4: (Top left) Distributed optimization. This schematic takes the perspective of multi-agent systems, although other scenarios abound. More on Distributed optimization is given in Section 2.1.2. (Top right) Decentralized optimization. (Bottom left) Parallel computing. (Bottom right) Distributed computing.

such that $f_i : \mathcal{X}_i \to \mathbb{R}$ for all $i \in \{1, \ldots, N\}$ and

$$\mathcal{X} = \mathcal{X}_1 \cup \cdots \cup \mathcal{X}_N,$$

such that $\mathcal{X}_i \cap \mathcal{X}_j = \emptyset$ for all $i \neq j$. Fully separable problems can be decomposed into N subproblems, each of which can be solved independently. Unsurprisingly, most problems do not come in this form (for $N > 1$) and instead have some kind of coupling between the subproblems. If a problem is separable save for some affine equality constraints, then the is problem is said to be partially separable.

The following offers an example of a partially separable problem that has been decomposed into N parts:

$$\begin{aligned} &\min_{x,z} \sum_{i=1}^{N} f_i(x_i) \\ s.t. \quad & x_i \in \mathcal{X}_i \subseteq \mathbb{R}^{n_i} \ \forall i \in \{1, \ldots, N\}, \\ &\sum_{i=1}^{N} A_i x_i = b, \end{aligned} \tag{2.10}$$

where $A_i \in \mathbb{R}^{\ell \times n_i}$ for all i, and $b \in \mathbb{R}^{\ell}$. Typically, algorithms for solving (2.10) involve a decomposition of the KKT criteria in order to guarantee local optimality upon termination. Such methods are known as distributed or decentralized optimization algorithms, depending on how the algorithm handles the coupling between each subproblem, if there is any. Where decomposition is concerned, the focus of this thesis will be on distributed optimization algorithms.

By decomposing a given problem and applying a distributed optimization method, it is sometimes possible to obtain solutions to large problems that would otherwise be intractable in a centralized setting. Consider Problem (2.10) and suppose that it is not solvable by any single available processor. However, suppose that

$$\begin{aligned} &\min_{x_1} \ f_1'(x_1) \\ s.t. \quad & x_1 \in \mathcal{X}_1' \end{aligned} \tag{2.11}$$

and

$$\begin{aligned} &\min_{x_2} \ f_2'(x_2) \\ s.t. \quad & x_2 \in \mathcal{X}_2' \end{aligned} \tag{2.12}$$

are both individually solvable, where $f_1', f_2', \mathcal{X}_1'$, and $\mathcal{X}_2'$ are some modification of $f_1, f_2, \mathcal{X}_1$, and $\mathcal{X}_2$, respectively. In this case, problem decomposition can be employed to solve the overall problem via a collection of subproblems. This now begs the question of how to construct $f_1', f_2', \mathcal{X}_1'$, and $\mathcal{X}_2'$, and how

to perform the coupling step so that $\sum_{i=1}^{N} A_i x_i = b$ is satisfied, however for now let us first consider other scenarios where distributed optimization may be beneficial.

As the subproblems in distributed optimization algorithms are independent of one another, there is also the possibility for speed-up via parallelization by solving each subproblem with a different processor [Pel09, TMW09, LBR15, MEHF18, VNR+06, LU05]. It is generally difficult to predict *a priori* how difficult each subproblem will be, and thus balancing the work done by each processor is non-trivial. As shall be shown in Section 4.1, the results can heavily depend on how the centralized problem is partitioned.

Additional benefits can be found where communication bandwidth is an issue. Namely, optimization becomes possible without the full problem description being stored in any one place. This feature enables multiple agents[4] to cooperate without the exchange of confidential data. Doing so can also mean that the amount of data transferred within the system can be reduced, minimizing expensive communication hardware requirements.

The inherent modularity of distributed and decentralized frameworks can also be beneficial in a multi-agent setting when the number of agents in the system is dynamic. For example, when a faulty component needs to be isolated and shut down for maintenance, the temporarily reduced system can be readily re-optimized within a distributed optimization framework. It is worth noting that agents can also be unresponsive or acting atypically, which are known as Byzantine faults. Detection and resolution of Byzantine faults are not considered within this thesis, but we refer the interested reader to an early paper by Lamport et al. ([LSP82]) and the overview by Driscoll ([DHSZ03]).

For distributed continuous convex optimization, there are already a number of available algorithms such as dual decomposition [Eve63, NS08], Alternating Direction Method of Multipliers (ADMM) [BPC+11, EB92, GM76], or Augmented Lagrangian based Alternating Direction Inexact Newton (ALADIN) methods [HFD16], which can all be used to solve large-scale strictly convex programs to global optimality by alternating between solving small-scale convex optimization problems and sparse linear algebra operations.

[4] In a distributed computing system, these would be the individual processors.

The main idea in dual decomposition is to take advantage of separable problem structure by iteratively solving a collection of dual subproblems. These dual subproblems are constructed by placing the constraints and appropriate Lagrange multipliers into the objective function. As an example, consider the dual problem of (2.10):

$$\begin{aligned} \max_{\lambda} \quad & \min_{x,z} \sum_{i=1}^{N} f_i(x_i) + \lambda^\top \left(\sum_{i=1}^{N} A_i x_i - b \right) \\ s.t. \quad & x_i \in \mathcal{X}_i \subseteq \mathbb{R}^{n_i} \ \forall i \in \{1, \ldots, N\}, \\ & \lambda \in \mathbb{R}^\ell \end{aligned} \tag{2.13}$$

where λ is the Lagrange multiplier of the consensus constraint $\sum_{i=1}^{N} A_i x_i = b$. As λ is also an optimization variable in dual problem (2.13), it is also referred to as a dual variable. Dual decomposition solves (2.10) by alternating between the dual subproblems for each $i \in \{1, \ldots, N\}$:

$$x_i^+ = \arg\min_{x_i} \ f_i(x_i) + (\lambda^+)^\top A_i x_i$$

and updating the dual variables:

$$\lambda^+ \leftarrow \lambda^+ + \alpha \left(\sum_{i=1}^{N} A_i x_i^+ - b \right).$$

where α is the chosen step size parameter. If the set of optimal Lagrange multipliers for Problem (2.10) is non-empty and $f = \sum_{i=1}^{N} f_i$ is strictly convex, then dual decomposition is known to converge to a unique minimizer. If f is not strictly convex, then convergence is no longer guaranteed [Han88, BPC+11].

For general convex problems, ADMM offers a good alternative to dual decomposition. Originally introduced in [GM75] and [GM76], ADMM has seen wide use in the realm of convex optimization, although it may be used as a heuristic for non-convex problems as well. In comparison with other methods ADMM is known to converge quite slowly, however solutions with minor suboptimality are often achievable in just tens of iterations. For a detailed overview of ADMM see [BPC+11]. A form of ADMM for solving (2.10) is shown in Algorithm 1.

Algorithm 1: ADMM

Input: $x_i^0 \in \mathcal{X}_i$, $\lambda_i^0 \in \mathbb{R}^\ell$ for $i \in \{1, \ldots, N\}$, $\rho > 0$, and a numerical tolerance $\varepsilon > 0$.

Initialization: Set $k = 0$.

1. Solve for all $i \in \{1, \ldots, N\}$:
$$y_i^{k+1} := \underset{y_i \in \mathcal{X}_i}{\arg\min}\, f_i(y_i) + (\lambda_i^k)^\top A_i y_i + (\rho/2)||A_i(y_i - x_i^k)||_2^2$$
2. Termination check:
If $||\sum_{i=1}^N A_i y_i^{k+1} - b||_1 < \varepsilon$ then terminate, else $k \leftarrow k + 1$ and go to Step 1.
3. Update the dual variable:
$$\lambda_i^{k+1} := \lambda_i^k + \rho A_i \left(y_i^{k+1} - x_i^k\right)$$
4. Solve the coupling QP:
$$x^{k+1} := \underset{x}{\arg\min} \sum_{i=1}^N (\rho/2)||A_i(y_i^{k+1} - x_i)||_2^2 - (\lambda_i^{k+1})^\top A_i x_i \tag{2.14}$$
$$\text{s.t.} \sum_{i=1}^N A_i x_i = b. \tag{2.15}$$

Output: x, λ.

It is proven that for a convex problem (2.10), the primal and dual iterates both approach optimality and $||\sum_{i=1}^N A_i y_i^{k+1} - b||_1 \to 0$ as $k \to \infty$ [BPC+11]. However, in practice this can take many iterations and it may be the case that the condition in Step 2 is satisfied before the primal and/or dual iterates reach optimality. Although ADMM has been applied to non-convex NLPs—notably the optimal power flow problem [Ers14, EMJ+17]—it is important to note if f_i or $\mathcal{X}_i$ is non-convex for some i, then ADMM may not converge [HFD16]. To solve such problems, the ALADIN algorithm was developed.

ALADIN builds on ADMM and sequential quadratic programming methods to attain convergence guarantees for general NLPs, and has been shown to converge significantly faster than ADMM [EMJ+17]. The main steps of ALADIN are outlined in Algorithm 2.

Algorithm 2: ALADIN

Input: Initial guess $x_0 \in X$, $\lambda > 0$, $\rho > 0$, $\mu > 0$, $\Sigma \succeq 0$ and a numerical tolerance $\varepsilon > 0$.

Initialization: Set $y = x_0$.

1. Solve for all $i \in \{1, \dots, N\}$ the decoupled NLPs

$$\begin{aligned} x_i^\star = \quad & \min_{x_i} \quad f_i(x_i) + \lambda^\top A_i x_i + \frac{\rho}{2} ||x_i - y_i||_{\Sigma_i}^2 \\ & \text{s.t.} \quad g_i(x_i) \leq 0 \mid \kappa_i \end{aligned}$$

2. Compute local gradients, Hessians, and active sets for QP

$$\begin{aligned} G_i = \quad & \nabla_{x_i} f_i(x_i)|_{x_i = x_i^\star} \\ H_i = \quad & \nabla_{x_i}^2 (f_i(x_i) + \kappa_i^\top g_i(x_i))|_{x_i = x_i^\star} \\ C_{i,j}^\star = \quad & \begin{cases} \nabla_{x_i} (g_i(x_i))_j |_{x_i = x_i^\star} & \text{if} \quad (g_i(x_i^\star))_j = 0 \\ 0 \quad \text{otherwise} \end{cases} \end{aligned}$$

3. Solve coupling QP

$$\begin{aligned} \Delta x \quad = \quad & \operatorname*{argmin}_{\Delta x, s} \frac{1}{2} \Delta x^\top H \Delta x + G^\top \Delta x + \lambda^\top s + \frac{\mu}{2} \|s\|_2^2 \\ \text{s.t.} \quad & A(x + \Delta x) - s = b \quad \mid \lambda_{QP} \\ & C^\star \Delta x = 0 \end{aligned}$$

4. Termination check
   If $\|Ax^\star - b\|_1 < \varepsilon$ **and** $\rho \, \|\Sigma_i (x_i^\star - y_i)\|_1 < \varepsilon$, terminate.
   Otherwise, update
   $y \leftarrow x^\star + \Delta x, \quad \lambda \leftarrow \lambda_{QP}$,
   and go to Step 1.

Output: $x^\star, \lambda$.

As in ADMM, the main idea is to decompose the problem into a collection of decoupled subproblems and a coupling QP.[5] Note that one may classify ALADIN as a two-level hierarchical optimization algorithm as the QP in Step 3 takes the role of a coordinating entity [Sca09]. The dual variable κ_i is associated with the i^{th} inequality constraint $h_i(x_i)$ and is obtained during the resolution of Step 1. Likewise, the dual variable λ_{QP} is associated with the equality consensus constraint in Step 3. In Step 2, the Jacobians of the active inequality constraint are obtained as they are required for the computation of Step 3.

While ALADIN has some favourable convergence properties, it can still only guarantee convergence to a locally optimal solution. Thus, if for some i, $\mathcal{X}_i$ is disconnected in (2.10), then the final solution from Algorithm 2 will greatly depend on its initial guess. Having a good initial guess is typically quite rare and thus for such disconnected problems it can be advised to use more specialized methods. The integrality constraints of MIPs are one such example of a disconnected feasible set in an NLP, and the methods used to solve them are the focus of Section 2.2.

2.2 Mixed-Integer Programming

Mixed-Integer Programs are non-convex problems that have the following form:

$$\min_{x,z} f(x,z) \tag{2.16a}$$

$$\text{s.t. } x \in \mathcal{X} \subseteq \mathbb{R}^n, \tag{2.16b}$$

$$z \in \mathcal{Z} \subseteq \mathbb{Z}^m. \tag{2.16c}$$

One can think of (2.16) as being a special case of (2.1), when there are polynomial equality constraints present. Such constraints are mathematically equivalent to the integer variables z of Problem (2.16), and thus such problems

[5] As this QP subproblem is equality constrained, it may be solved as a matrix inversion problem via solution of the relevant Karush-Kuhn-Tucker conditions.

are most commonly represented as MIPs. The discrete variables z often arise as "switches" between different values or dynamics in a system. Some examples include on/off decisions, selling/buying, or perhaps a choice of gear.

In terms of complexity, MIPs are known to be NP-complete [PS82]; i.e. both global and local MIP algorithms often do not scale well in terms of computation time. This is mainly due to fact that integer decisions in MIPs generally lead to disconnected, and thus non-convex, feasible sets. The standard approach utilized by numerical optimization algorithms to deal with this is to use B&B methods [BKL+13, BBC+08, IBM19, Gur19]. These methods seek to establish optimality without having to explore the entire integer-feasible decision space. This is done by iteratively "branching" integer decisions and then solving relaxed subproblems in order to establish upper and lower bounds on the optimal objective value. The details on B&B, along with other techniques for mixed-integer programming are given in this section.

2.2.1 Branch and Bound

Branch and Bound originates with [LD60] and [MM57] and is the most common method of solving mixed-integer programs. The algorithm consists of iteratively splitting the feasible set and obtaining a solution to the subproblems resulting from each new subset. The decision of how to split the feasible set is known as "branching" and more details on that are given in Section 2.2.2. If it is established that an optimal solution cannot exist in one of the subsets then it is "pruned" and not explored further.

For mixed-integer programs, the integer decisions are typically chosen for branching.[6] That is, some integer variables are fixed to certain values while others are relaxed. These subproblems are typically arranged in a tree structure such that the root node of the tree corresponds to the convex relaxation of the MIP, and each child node corresponds to a different feasible decision for a particular integer variable.

[6] For highly non-convex NLPs with many local optima, global optimality can be established by partitioning the feasible set and branching on each partition.

As an example, consider the following integer program:

$$\begin{aligned} \min_{z} \quad & \sum_{i=1}^{4} z_i \\ \text{s.t.} \quad & 2z_1 + z_2 \leq 1, \\ & 2z_2 + z_3 \leq 1, \\ & 2z_3 + z_4 \leq 1, \\ & z_i \in \{0,1\} \quad \forall i \in \{1,\ldots,4\}. \end{aligned} \tag{2.17}$$

Problem (2.17) can be solved in a B&B framework by first solving its continuous relaxation, followed by (2.17) with $z_1 = 0$ and $z_2, z_3, z_4 \in [0,1]$, etc. This process is depicted in Figure 2.5 by the B&B decision tree for (2.17) with a "breadth-first" branching strategy.

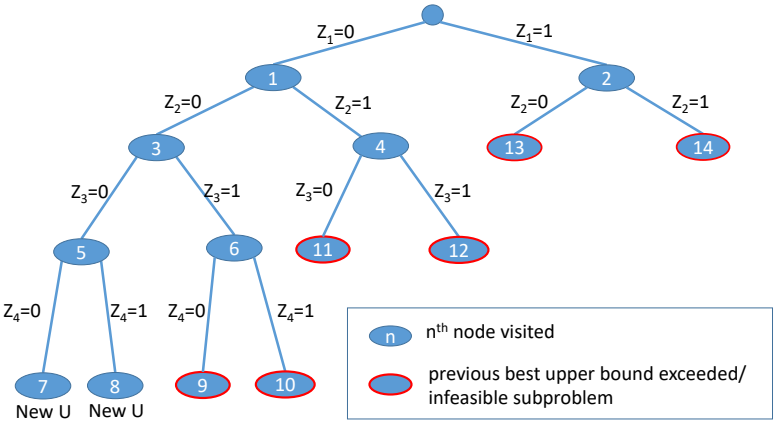

Figure 2.5: An example of a B&B decision tree for Problem (2.17).

Pseudocode for the B&B method is shown in Algorithm 3. The problem relaxations are used to establish lower bounds on the optimal objective value. This is done by taking the smallest solution from leaves of the explored decision subtree $\mathcal{T}$. That is, the lowest value among nodes which do not have explored

children.[7] On the other hand, leaf nodes of the B&B tree correspond to subproblems with all integers fixed, and thus feasible solutions to the given problem, which in turn provide upper bounds on the optimal objective value. If a node in the candidate node set $\mathcal{S}$ is found to have a value above that of the current best upper bound U, then clearly its corresponding child nodes do not need to be searched since they contain a subset of the feasible set of current subproblem and can thus only have an equal or greater solution. This pruning technique can be used to eliminate large swathes of the feasible set and better focus the search for the optimal value. If $U - L \leq \epsilon$ for a sufficiently small threshold $\epsilon > 0$, then the algorithm will terminate without searching any other nodes with the upper bound value as the solution.

Algorithm 3: Branch and Bound

Input: Termination tolerance $\epsilon > 0$.
Preparation: Set $U = \infty$, $L = -\infty$, $\mathcal{T} = \emptyset$. Candidate node set $\mathcal{S} = \{0\}$.
While $\mathcal{S} \neq \emptyset$**:**

1. Choose $n \in \mathcal{S}$, and update $\mathcal{S} \leftarrow S\{n\}$
2. Obtain an upper bound $J^\star(n)$ on the value of the child nodes of n.
3. If $J^\star(n) < U$, then $U \leftarrow J^\star(n)$ and proceed to Step 4.
   If $J^\star(n) > U$ proceed to Step 1.
   Else add the child nodes of n to $\mathcal{S}$ and proceed to Step 4.
4. $L \leftarrow \min_{n \in \mathcal{P}(\mathcal{S})} \{J^\star(n)\}$

   If $U - L \leq \epsilon$ terminate.
   Else proceed to Step 1.

Output: $J^\star(n) = U$.

The upper bounding function $J^\star(n)$ can be constructed in a number of ways. For example, suppose that each node n corresponds to a vector $\mathbf{Z}(n) \in \mathbb{R}^m$ such that $\mathbf{Z}(n)_i \in \mathbb{Z}$ for $i \in \{1, \ldots, m\}$. Furthermore, suppose that for every child node $\tilde{n}$ of n, it holds that $|\{i|\mathbf{Z}(n)_i \in \mathbb{Z}\}| < |\{i|\mathbf{Z}(\tilde{n})_i \in \mathbb{Z}\}|$. That is, the set of fixed integers of the child nodes contains the set of fixed integers of their

[7] This statement assumes a typical strategy—such as depth-first or breadth first—for exploring the decision tree. The fully general statement would additionally require a path to the root node through the subtree of explored nodes.

parent nodes. The value $J^\star(n)$ for node n can then be obtained by solving (2.16) such that

$$z_i = \begin{cases} \mathbf{Z}(n)_i, & \text{if } \mathbf{Z}(n)_i \in \mathbb{Z} \\ z_i \in conv(\mathbb{Z}_i), & \text{otherwise} \end{cases}$$

where $conv(Z)$ denotes the convex hull of a set Z.[8]

It should be noted that the efficiency of Algorithm 3 is highly dependent on preprocessing of the problem and the heuristics used in Step 1. Such preprocessing can take many forms, such as the additional or removal of constraints and variables, or solution of a subproblem to establish good optimality bounds. Some background on preprocessing methods for MIPs can be found in [Sav94], and some other heuristics can be found in [RP08] and [FL10]. More information on how to perform Step 1 is given in Section 2.2.2.

2.2.2 Branching Heuristics

One major question when using the B&B method is which values to branch on. Indeed, this is very difficult (if not impossible) to answer *a priori* and only heuristics with good, but not provably optimal, performance can be applied. There are a number of branching heuristics which have been developed, a few of which shall be decribed here:

- Breadth-first: Explore all sibling nodes (all nodes of the same depth) before proceeding to their child nodes. Good for improving the lower bound.
- Depth-first: Explore child nodes before proceeding to sibling nodes. Quickly obtains upper bounds.
- Best-first: Explore the node with the lowest associated objective value. Similar to a greedy algorithm.

[8] The convex hull of Z is the smallest superset of Z which is convex. For example, $conv(\{0, 2, 3\}) = [0, 3]$.

More advanced branching rules also exist, such as reliability branching [AKM05], strong branching, and pseudocost branching [LS99]. Which branching method is best to use is difficult to answer since different branching rules are better suited to different contexts. For example, if no upper bound U is available then a branching rule which quickly obtains one, such as depth-first, is often preferrable since it can then be used to begin pruning the search tree. On the other hand, a strategy that quickly improves the lower bound L, such as breadth-first, is better in the opposite case since it can then lead to faster termination by satisfying $U - L \leq \varepsilon$.

Section 4.2.3 presents a method for generating a custom branching heuristic based on *a priori* information. For example, online optimal control often involves the solution of a sequence of related problems. Information about not only their final results, but also their decision tree subproblems can be carried over between iterations for use in the next problem. In effect, this means that different areas of the feasible set can be prioritized and potentially yield faster solutions.

Cutting Planes

While B&B is an effective tool for organizing a search of the feasible set, there are other tools which can assist in the solution-finding process. Cutting planes, also known as cuts, are added constraints which are meant to allow for faster optimization [DFJ54]. For Mixed Integer Linear Programs (MILPs), a solution can be obtained through convex relaxation if the convex hull of the MILP is equivalent to its LP relaxation. As an example, consider the problem shown in Figure 2.6. Here, the cutting planes are linear inequality constraints. With these additional constraints, a continuous relaxation of the problem results in an optimal solution to the original MIP.

In general, constructing these cuts can be quite difficult and a number of methods have been developed. Some examples of cutting plane algorithms are:

- Gomory cut [Gom58]: Solve the continuously relaxed problem and obtain a solution. Introduce a cut such that the relaxed solution is on one side of the cutting plane and all feasible integer points are on the other. Many such cuts may be required before an integer point is obtained.

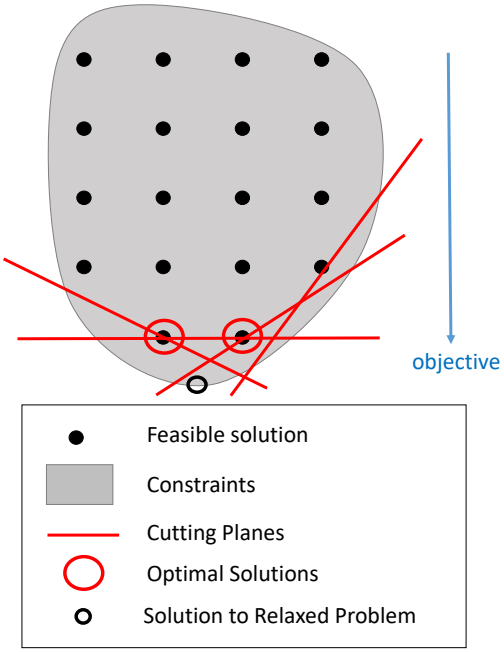

Figure 2.6: The gray area is region bounded by the inequality constraints, the black dots are the integer-feasible points and the red lines are the added constraints.

- Chvatal-Gomory cut [Chv73]: Given the constraints $Az \leq b$ on the integer valued variable z, then it holds that for any $\lambda > 0$ one may use the cut $\lfloor \lambda Az \rfloor \leq \lfloor \lambda b \rfloor$.
- Split cut [Bal79]: For some integer variable z, obtain the sets D_1, D_2 constructed by $z \leq y$ and $z \geq y + 1$, respectively. For a feasible set G, the split cut is obtained by computing $(D_1 \cup D_2) \cap G$.

While cutting plane methods alone can take many iterations to obtain an integer solution, they can be very effective when used in conjuction with a B&B method. This is known as "branch and cut" [PR91]. This method can yield

faster convergence by returning better solutions to the relaxed subproblems and thus better lower bounds.

2.2.3 Outer Approximation

Outer Approximation (OA) goes back to Duran and Grossmann [DG86] and is a commonly used algorithm for solving mixed-integer programs which are convex under continuous relaxation. Problems belonging to this class are known as Mixed Integer Convex Programs (MICPs), and differ from general MINLPs in that they can be solved to global optimality without resorting to spatial B&B or heuristic methods [LYBV18, KAGB04]. A notable extension of OA has been developed by Fletcher and Leyffer [FL94], who suggest to include curvature information in the relaxed integer program leading to a quadratic outer approximation method.

The main steps of OA consist of an NLP subproblem with fixed integer values and an MILP. A construction of such an NLP subproblem for (2.16) is shown below:

$$\begin{aligned} f^\star(z) \quad &= \min_{x,y} \; f(x,z) \\ &\quad s.t.\; Ax + Bz = c, \\ &\quad\quad x \in \mathcal{X}, \end{aligned} \tag{2.18}$$

for a fixed integer parameter z.

The second step of OA involves constructing a set of supporting hyperplanes at the solution to (2.18). Let $x^\star(z)$ denote a solution of (2.18) in dependence on z. If f is not differentiable, we have to use a particular choice of the subgradient, that is:

$$\lambda^\star(z) \in \partial_x f(x^\star(z), z).$$

Likewise, one can construct subgradients of f with respect to z at the optimal solution of (2.18):

$$\mu^\star(z) \in \partial_z f(x^\star(z), z).$$

Both $\lambda^\star(z)$ and $\mu^\star(z)$ can be use to construct supporting hyperplanes of the convex function f at the point z. A collection of such hyperplanes can then be

used to construct a piecewise affine approximation of f. Let H denote a set of hyperplane coefficients associated with a finite set of points $\Pi \subseteq \mathcal{Z}$:

$$\mathcal{H}(\Pi) = \left\{ (\alpha, \beta, \gamma) \;\middle|\; \begin{array}{l} z \in \Pi \\ \alpha = \lambda^\star(z) \\ \beta = \mu^\star(z) \\ \gamma = f^\star(z) - \alpha^\mathsf{T} x^\star(z) - \beta^\mathsf{T} z \end{array} \right\} . \tag{2.19}$$

The set H can be used to construct a piecewise affine lower bound on f:

$$\Phi(x, z, \Pi) = \max_{(\alpha,\beta,\gamma)\in H(\Pi)} \left\{ \alpha^\mathsf{T} x + \beta^\mathsf{T} z + \gamma \right\} \leq f(x, z) \; . \tag{2.20}$$

The particular construction of Φ on f allows us to establish for the following tightness property:

Lemma 1. *Let $\mathcal{X}$ and* $\mathrm{conv}(\mathcal{Z})$ *be compact convex polytopes and let $\Pi \subseteq \mathcal{Z}$ be any finite set of points. It holds that for all $z \in \Pi$,*

$$f^\star(z) = \min_{x\in\mathcal{X}} \; \Phi(x, z, \Pi) \quad \text{s.t.} \quad Ax + Bz = c \; . \tag{2.21}$$

Proof. If there is no $x \in \mathcal{X}$ and $z \in \mathcal{Z}$ such that $Ax + Bz = c$, then both sides of (2.21) are equal to infinity and the statement of the lemma holds in the extended value sense. Thus, we may assume that the equation $Ax + Bz = c$ has a solution in $\mathcal{X}$ for a fixed z. By the definition of Φ in (2.19) and (2.20) we have that

$$\Phi(x, z, \Pi) = f^\star(z) + \left[\lambda^\star(z)\right]^\mathsf{T} (x - x^\star(z)) + \underbrace{\left[\mu^\star(z)\right]^\mathsf{T} (z - z)}_{=0}$$

Thus, we have

$$\begin{aligned} & \min_{x \in X, Ax+Bz=c} \Phi(x, z, \Pi) \\ = & \min_{x \in \mathcal{X}, Ax+Bz=c} f^\star(z) + \left[\lambda^\star(z)\right]^\mathsf{T} (x - x^\star(z)) \\ = & f^\star(z) - \left[\lambda^\star(z)\right]^\mathsf{T} x^\star(z) + \underbrace{\min_{x \in \mathcal{X}, Ax+Bz=c} \left[\lambda^\star(z)\right]^\mathsf{T} x}_{LP1} \end{aligned}$$

Let $\mathcal{X} := \{x | Gx \leq 0\}$ where $G \in \mathbb{R}^{n \times m}$. Writing out the KKT conditions of LP1 and (2.18) yields:

$$\begin{aligned} \left[\lambda^\star(z)\right]^\top + \nu^\top A + \mu^\top G = 0 \\ Ax + Bz = c \\ Gx \leq 0 \\ \mu \geq 0 \\ \mu^\top G x = 0 \end{aligned} \qquad \begin{aligned} \partial_x f(x, z) + \nu^\top A + \mu^\top G = 0 \\ Ax + Bz = c \\ Gx \leq 0 \\ \mu \geq 0 \\ \mu^\top G x = 0 \end{aligned}$$

which in turn reduce to

$$\begin{aligned} \left[\lambda^\star(z)\right]^\top x + \nu^\top (c - Bz) = 0 \\ Ax + Bz = c \\ Gx \leq 0 \end{aligned} \qquad \begin{aligned} \partial_x f(x, z) x + \nu^\top (c - Bz) = 0 \\ Ax + Bz = c \\ Gx \leq 0 \end{aligned}$$

As $x^\star$ is a solution to (2.18) and $\lambda^\star(z) = \partial_x f(x, z)|_{x=x^\star}$ by construction, it follows that $x^\star$ is a solution to LP1. This proves the assertion

$$f^\star(z) = \min_{x \in \mathcal{X}} \Phi(x, z, \Pi) \quad \text{s.t.} \quad Ax + Bz = c$$

for all $z \in \Pi$. □

Lemma 1 states that the minimum of (2.18) with an objective function f and compact convex polytopic feasible set $\mathcal{X}$ is equivalent to the minimum of the linearization of f at a globally optimal point $x^\star \in \mathcal{X}$. Note that the proof of Lemma 1 does not rely on the convexity of f, however if f is non-convex

then it no longer holds that $\Phi(x, z, \Pi) \leq f(x, z)$ for all $(x, z) \in \mathcal{X} \times \mathcal{Z}$. Thus, the convexity of f is crucial for the third step of OA. Herein, the supporting hyperplane set $H(\Pi)$ are used to solve the MILP coupling subproblem given by:

$$\min_{x \in \mathcal{X}, z \in \mathcal{Z}} \Phi(x, z, \Pi) \quad \text{s.t.} \quad Ax + Bz = c \,, \tag{2.22}$$

which gives a lower bound, L, on (2.16). As the solutions $(x^\star, z^\star)$ of subproblem (2.18) are feasible points of Problem (2.16), the value of $f(x^\star, z^\star)$ provides an upper bound, U, on (2.16). Thus, optimality of a point $(x^\star, z^\star)$ can be established by satisfaction of $U - L \leq \varepsilon$ for a given termination tolerance ε.

The main steps of OA are summarized in Algorithm 4, and an illustration of a possible iteration of the algorithm is provided in Figure 2.7.

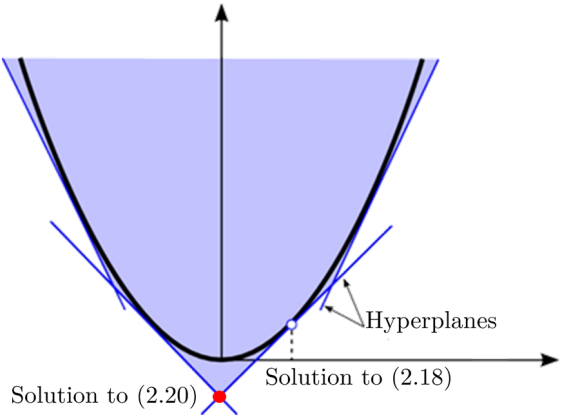

Figure 2.7: Outer Approximation Example.

Algorithm 4: Outer Approximation for MICP

Input: Initial guess $z \in \mathcal{Z}$ and a numerical tolerance $\varepsilon > 0$.

Initialization: Set $\Pi = \emptyset$ and $U = \infty$.

Repeat:

1. Solve the convex optimization problem

$$f^\star(z) = \min_x \; f(x,z) \quad \text{s.t.} \quad \begin{cases} Ax + Bz = c \\ x \in \mathcal{X} \,. \end{cases} \tag{2.23}$$

2. If (2.23) has no feasible solution, return a certificate of infeasibility. Otherwise, update

$$U \leftarrow \min\left\{U, f^\star(z)\right\} \quad \text{and} \quad \Pi \leftarrow \Pi \cup \{z\} \,.$$

3. Solve the (extended) MILP

$$(x^+, y^+, z^+) \in \underset{x\in\mathcal{X}, y, z\in\mathcal{Z}}{\operatorname{argmin}} \; \sum_{i=1}^N y_i \quad \text{s.t.} \quad \begin{cases} \forall i \in \{1,\dots,N\}, \\ \forall (\alpha_i, \beta_i, \gamma_i) \in \mathcal{H}_i \; (\Pi) \\ \alpha_i^\top x_i + \beta_i^\top z_i + \gamma_i \le y_i \\ Ax + Bz = c \end{cases} \tag{2.24}$$

4. If $U - \sum_{i=1}^N y_i^+ \le \varepsilon$, terminate.

5. Update $z \leftarrow z^+$ and go to Step 1.

Output: (x^+, z^+).

The following finite termination result for outer approximation is (at least in very similar versions) well-known in the literature [DG86, LYBV18].

Theorem 2. *For MICPs, Algorithm 4 terminates after a finite number of iterations.*

Proof. Notice that if the equation $Ax + Bz = c$ has no solution in $\mathcal{X}$, this will be detected immediately by Step 2 of Algorithm 4, which causes termination. Thus, we may assume that all optimization problems are feasible.

The main idea of the proof is to assume that the termination criteria are never satisfied and proceed by contradiction. First, note that any solution (x^+, y^+, z^+) of the MILP (2.24) satisfies the equation

$$\Phi(x^+, z^+, \Pi) = \sum_{i=1}^{N} y_i^+ \tag{2.25}$$

by construction. Moreover, because we have $Ax^+ + Bz^+ = c$, the inequality

$$\min_{x, Ax+Bz^+=c} \Phi(x, z^+, \Pi) \leq \Phi(x^+, z^+, \Pi) \tag{2.26}$$

holds. If we assume that the termination criterion is not satisfied, we must have

$$\sum_{i=1}^{N} y_i^+ < U - \epsilon \,. \tag{2.27}$$

If $z^+ \in \Pi$, then the result of Lemma 1 implies that

$$f^\star(z^+) \overset{(2.21)}{=} \min_{x, Ax+Bz^+=c} \Phi(x, z^+, \Pi) \,, \tag{2.28}$$

as well as $U \leq f(z^+)$. By substituting all the above relations we find that

$$f^\star(z^+) \overset{(2.28),(2.26)}{\leq} \Phi(x^+, z^+, \Pi) \overset{(2.25),(2.27)}{<} U - \epsilon \leq f(z^+) - \epsilon \,,$$

which is a contradiction.

If $z^+ \notin \Pi$, then $\Pi \leftarrow \Pi \cup \{z^+\}$ in Step 2 and the algorithm continues. This implies z^+ cannot be chosen twice, and as $\mathcal{Z}$ is finite, Algorithm 4 must terminate after a finite number of iterations. □

2.2.4 Mixed-Integer Optimal Control

The methods described in Sections 2.2.1-2.2.3 are applicable to a wide class of problems, however there are more efficient, specialized methods available for MIPs with specific structure. A prime example of which are Mixed-Integer

Optimal Control Problems (MIOCPs). First, consider the general form of an Optimal Control Problem (OCP) with a discretized time horizon of length T:

$$\min_{x(\cdot),u(\cdot)} \sum_{t=0}^{T-1} f(x(t), u(t)) \tag{2.29a}$$

$$\text{subject to } \forall t \in \{0, \dots, T-1\},$$

$$x(t+1) = h(x(t), u(t)), \quad x(0) = x_0, \tag{2.29b}$$

$$x(t) \in \mathcal{X} \subseteq \mathbb{R}^n, \quad u(t) \in \mathcal{U} \subseteq \mathbb{R}^\ell. \tag{2.29c}$$

where $f : \mathbb{R}^n \times \mathbb{R}^\ell \to \mathbb{R}$ is the stage cost and $h : \mathbb{R}^n \times \mathbb{R}^\ell \to \mathbb{R}^n$ is the state transition map.

One can think of OCPs as being a specific type of optimization problem which includes time-varying state dynamics upon which an objective function must be minimized. The solution of (2.29) is a control trajectory

$$\{u^\star(t) | t \in \{0, \dots, T-1\}\} .$$

The underlying dynamics present in OCPs add structure which can be exploited in terms of numerical solutions. For example, a discretization of an OCP leads to an optimization problem where each time step is dependent on the previous one. This naturally leads to a KKT-system with a block-diagonal structure which can be exploited [KBSS11]. Depending on the exact nature of the problem, more specialized techniques can be used. Some examples include detection of control switching, symmetry reduction, or reformulation [BKMP18, FOBK12, SM06]. In the latter, it is common practice to reformulate hybrid systems as MIOCPs [BM99, BN18, ABRM14]. Model Predictive Control is another area which could benefit from some of the methods described in the following sections, although it will not be extensively discussed in this thesis. For hybrid model predictive control, some work has already been done and we refer the interested reader to the following: [BDCHJ10, BHDS02, BN18, LH09].

MIOCPs are a subclass of (2.29) which include discrete decision variables. Some examples include the Lotka Volterra fishing problem [Sag05] or the

classic "Fuller's problem" [Ful63].[9] Hybrid systems are often formulated as MIOCPs, and have recently seen significant interest [BN18, ABRM14, Gey09, CL18]. These problems involve either switching between different dynamics or jumps in the state trajectory. For completeness, the general form of an MIOCP is given below:

$$\min_{x,u,z} \sum_{t=0}^{T-1} f(x(t), u(t), z(t)) \tag{2.30a}$$

subject to $\forall t \in \{0, \dots, T-1\}$,

$$x(t+1) = h(x(t), u(t), z(t)), \quad x(0) = x_0, \tag{2.30b}$$

$$x(t) \in \mathcal{X} \subseteq \mathbb{R}^n, \quad u(t) \in \mathcal{U} \subseteq \mathbb{R}^\ell \tag{2.30c}$$

$$z(t) \in \mathcal{Z} \subseteq \mathbb{Z}^m, \tag{2.30d}$$

As (2.30) is an MIP, it is nonconvex and thus generally NP-complete. Nonetheless, as with (2.29), there is a particular structure to the problem which can be exploited. This is discussed in depth within Section 4.2, where several techniques for efficient exploitation of MIOCPs are given.

[9] Note that, although the original formulation of Fuller's problem is not an MIOCP it can - and often is - formulated as one.

3 Algorithms for Distributed Mixed-Integer Optimization

When designing distributed optimization algorithms there are a number of things that one must keep in mind:

- Is the algorithm guaranteed to converge?
- Will the returned solution be guaranteed to be optimal?
- How should the agents/processors coordinate with each other?

Each of these points is prioritized and addressed differently among each of the algorithms presented in this section, and there is rarely a perfect answer to these questions. Indeed, as is shown in Section 3.3, an algorithm which has favourable properties on paper is not necessarily one which has good performance.

In Section 2.1.2, some distributed algorithms for continuous problems were described, however as we shall see they are not effective methods for solving MIPs. Overall there are relatively few distributed methods for solving MIPs. For MILPs, there is Benders Decomposition [Ben62] and Dantzig-Wolfe Decomposition [DW60], however both are limited in their scope of applicability. While extensions to higher order problems have been developed [Geo72], these lack optimality, and in some cases, convergence guarantees [SG91]. Consider the following MILP:

$$\min_{x,z} \; c_1^\top x + c_2^\top z \tag{3.1a}$$

$$s.t. \; Ax + Bz \geq b \tag{3.1b}$$

$$z \in \mathcal{Z} \subset \mathbb{Z}^m \tag{3.1c}$$

$$x \geq 0 \tag{3.1d}$$

The main steps of Benders Decomposition for Problem (3.1) are as follows [Kal02]:

1. Initialize $z^\star$ with a feasible integer solution, and set upper bound $U = \infty$ and lower bound $L = -\infty$.

2. Solve the subproblem:

$$u^\star = \arg\min_{u} \; c_2^\top z^\star + (b - Bz^\star)^\top u$$
$$s.t. \; A^\top u \leq c_1$$
$$u \geq 0$$

3. If the solution to Step 2 is unbounded then get the unbounded ray $u^\star$ and add the cut $(b - Bz)^\top u^\star$ to the master problem.

   Otherwise, get the extreme point $u^\star$ and add the cut $y \geq c_2^\top z + (b - Bz)^\top u^\star$ to the master problem. Set $U = \min\{U, c_2^\top z^\star + (b - Bz^\star)^\top u^\star\}$.

4. Solve the master problem:

$$z^\star, y^\star = \arg\min_{z,y} \{y | cuts, z \in \mathcal{Z}\}.$$

   and set $L = y^\star$.

5. If $U - L < \epsilon$ then terminate. Else go to Step 2.

The main idea of Benders Decomposition is to use the continuous subproblems solved in Step 2 to obtain good solutions in the master problem solved in Step 4. The solution to the integer master problem is then used to inform the continuous subproblem. This process continues until the termination threshold between the upper and lower bounds has been satisfied.

Apart from Benders Decomposition there are other alternatives to solving MIPs in a distributed framework. For example, an extension of ADMM for MIPs was presented in [TMBB20] but also lacks any convergence or optimality guarantees. Thus, one of the main contributions of this thesis is to address this gap in the literature. To this end, three different algorithms are presented. The first is a mixed-integer extension of the ALADIN algorithm described in Section 2.1.2, and is shown to outperform competing methods in Section 3.1.

The second, Partially Distributed Outer Approximation (PaDOA), is based on the outer approximation algorithm described in Section 2.2.3. PaDOA was originally developed as an attempt to create a distributed algorithm for MIPs with strong convergence and optimality guarantees, and as seen in Section 3.2, it can outperform some commercial solvers on several problem classes. The third is a distributed branch and bound algorithm which aims to maintain the optimality guarantees of PaDOA, while remaining applicable to a wide range of problems. Distributed branch and bound is discussed in Section 3.3 along with two case studies that demonstrate its numerical performance.

3.1 Mixed-Integer ALADIN

Both ADMM and ALADIN were originally designed with purely real-valued problems in mind and thus are not immediately applicable to MIPs. It is always possible to reformulate integer constraints as polynomial equality constraints [SOM04]:

$$z \in \{\zeta_1, \dots, \zeta_n\} \iff (z - \zeta_1) \dots (z - \zeta_n) = 0, \quad z \in \mathbb{R}.$$

However, doing so is typically not helpful if the problem is to be solved via a local optimization method. This is due to the fact that each integer point is itself locally optimal and thus the solution of a local algorithm will remain in the neighbourhood of the initial point. Thus, using a local algorithm to solve an NLP with polynomial equality constraints essentially amounts to an expensive means of fixing the integer values.

The following presents an example problem to illustrate the difficulty with which MIPs are solved using continuous local optimization methods:

$$\min_{x,z \in \mathbb{R}} x_1 + z_1 + x_2 + z_2 \tag{3.2a}$$
$$\text{s.t. } -2 \leq x_1, x_2 \leq 2 \tag{3.2b}$$
$$(x_1 + z_1)^2 \leq 1 \tag{3.2c}$$
$$(x_2 + z_2)^2 \leq 1 \tag{3.2d}$$
$$z_1(z_1 - 1) = 0 \tag{3.2e}$$
$$z_2(z_2 - 1) = 0 \tag{3.2f}$$
$$x_1 + x_2 = 0.5. \tag{3.2g}$$

An optimal solution to Problem (3.2) is clearly $x_1 = x_2 = 0.25$ and $z_1 = z_2 = 0$, however if ALADIN (without line search) is initialized closer to $(z_1, z_2) = (1, 1)$ then the algorithm can fail to converge.

This can be seen by first assuming that Step 1 of ALADIN is solved with a local optimization method. In this case, the solution will be either $x_i^\star \geq -1, z_i^\star = 0$ or $x_i^\star \leq 0, z_i^\star = 1$ for $i = 1, 2$, depending on how the local solvers are initialized and the value of ρ. Given that the initial guess for the problem includes $z_1 = z_2 = 1$, we will proceed with the second possibility as it is the most likely outcome.

In Step 2, the active constraint matrix $C^\star$ includes values for the active quadratic constraints (3.2e) and (3.2f). These constraints are always active, and thus Step 3 will always include

$$\Delta z_1 = 0, \quad \Delta z_2 = 0.$$

The solutions $x_1^\star$ and $x_2^\star$ will both continue to be less than or equal to zero and thus cannot satisfy the termination criteria $x_1^\star + x_2^\star = 0.5$. The result is that ALADIN returns to Step 1 with the primal iterates of z_1 and z_2 unchanged. Thus the previous steps are repeated indefinitely, not matter how many iterations are allowed.

However, Problem (3.2) is easily formulatable as an MIP. Namely, by replacing (3.2e) and (3.2f) with $z_1, z_2 \in \{0, 1\}$. As an MIP, the problem is efficiently solvable in a centralized manner using current MIP methods such as outer approximation or B&B. Problem (3.2) can also be solved in a distributed framework through an extension of ALADIN to mixed-integer programming.[1] The structure of one such extension as originally published in [MFH18] is shown in Algorithm 5.

As with the original ALADIN algorithm, Algorithm 5 seeks to determine the solutions to a set of MINLPs in the first step and then keep the integer variables fixed throughout the remaining steps until the consensus step among the real-valued variables is completed and the next iteration begins. Note that the coupling step only involves the real-valued variables x and thus Algorithm 5 is only applicable to problems of the form:

$$\begin{aligned} \min_{x\in\mathbb{R}^n, z\in\mathbb{Z}^m} \quad & \sum_{i=1}^{N} f_i(x_i, z_i) \\ \text{s.t.} \quad & g_i(x_i, z_i) \leq 0, \quad \forall i \in 1, \ldots, N, \\ & \sum_{i=1}^{N} A_i x_i = b. \end{aligned} \tag{3.3}$$

It is easy to verify that Algorithm 5 applied to (3.3) for a fixed value of z reduces to the standard ALADIN algorithm, however in general Algorithm 5 does not inherit the convergence properties of ALADIN. The convergence properties of Algorithm 5 are the subject of Section 3.1.1.

3.1.1 Convergence Properties

As shown by the example of Problem (3.2), ALADIN may return a poor solution if the feasible set is disconnected. Reformulation of the problem to an MIP and application of MI-ALADIN can return a better solution as the integer variables are directly taken into account. For Problem (3.2), MI-ALADIN converges to the optimal solution almost immediately. Thus, it is unclear to

[1] For Problem (3.2), Algorithm 5 converges to the global solution in just 3 iterations.

Algorithm 5: Mixed-Integer ALADIN

Input: Initial guess $x_0 \in X$, $\lambda > 0$, $\rho > 0$, $\mu > 0$, $\Sigma \geq 0$ and a numerical tolerance $\varepsilon > 0$.

Initialization: Set $y = x_0$.

1. Solve for all $i \in \{1, \ldots, N\}$ the decoupled NLPs

$$(x_i^\star, z_i^\star) = \min_{x_i, z_i} \quad f_i(x_i, z_i) + \lambda^\top A_i x_i + \frac{\rho}{2} ||x_i - y_i||_{\Sigma_i}^2$$
$$\text{s.t.} \quad g_i(x_i, z_i) \leq 0 \mid \kappa_i$$

2. Compute local gradients, Hessians, and active sets for QP

$$G_i = \nabla_{x_i} f_i(x_i, z_i^\star)|_{x_i = x_i^\star}$$
$$H_i = \nabla_{x_i}^2 (f_i(x_i, z_i^\star) + \kappa_i^\top g_i(x_i, z_i^\star))|_{x_i = x_i^\star}$$
$$C_{i,j}^\star = \begin{cases} \nabla_{x_i} (g_i(x_i, z_i^\star))_j |_{x_i = x_i^\star} & \text{if} \quad (g_i(x_i^\star, z_i^\star))_j = 0 \\ 0 \quad \text{otherwise} \end{cases}$$

3. Solve coupling QP

$$\Delta x^\star = \operatorname*{argmin}_{\Delta x, s} \frac{1}{2} \Delta x^\top H \Delta x + G^\top \Delta x + \lambda^\top s + \frac{\mu}{2} \|s\|_2^2$$
$$\text{s.t.} \quad A(x + \Delta x) - s = b \quad \mid \lambda_{QP}$$
$$C^\star \Delta x = 0$$

4. Termination check
   If $\|Ax^\star - b\|_1 < \varepsilon$ **and** $\rho \, \|\Sigma_i (x_i^\star - y_i)\|_1 < \varepsilon$, terminate.
   Otherwise, update $y \leftarrow x^\star + \Delta x^\star, \quad \lambda \leftarrow \lambda_{QP}$,
   and go to Step 1.

Output: $(x^\star, z^\star)$, λ.

what extent Algorithm 5 inherits the properties of ALADIN. To this end, this section investigates the convergence properties of Algorithm 5.

Although Algorithm 5 can be applied to problems of the form shown in Problem 3.3, convergence cannot be guaranteed for every problem within this class. However, guarantees can be made for the following subclass of problems:

$$\begin{aligned} \min_{x,z} \quad & f(x) + g(z) \\ \text{s.t.} \quad & h(x) \leq 0, \\ & x \in \mathbb{R}^n, \quad z \in \mathbb{Z}^m. \end{aligned} \tag{3.4}$$

As the real-valued and integer parts of Problem (3.4) are completely separated, it is clear that application of Algorithm 5 to Problem (3.4) will result in the same primal (real-valued) iterates as would be observed when applying the original ALADIN algorithm to Problem 3.4 with a fixed value of z. Thus, by the convergence properties of ALADIN, Algorithm 5 will converge for Problem (3.4) in a finite number of iterations.

3.1.2 Case Study – Battery Scheduling

To test the performance Algorithm 5 we require a problem which is easily scalable both in terms of the number of subproblems N and the dimension of each subproblem $n_i + m_i$. One problem which fits these requirements and is in the form (3.3) is the battery scheduling problem. The battery scheduling problem is depicted in Figure 3.1 for N batteries, with a central connection to the main grid.

At each time step t in $\mathbb{T} = \{1, \ldots, T\}$, battery i has a state of charge $E_i(t)$ along with local power generation $\mathcal{L}_i(t)$ and load profiles $\mathcal{P}_i(t)$. These are taken together and parametrized as $L_i(t) = \mathcal{P}_i(t) - \mathcal{L}_i(t)$. If insufficient power is generated or stored in the battery at time t then power $P_i^+(t)$ can be drawn from the grid to satisfy the demand at node i. Likewise, if there is a surplus then power $P_i^-(t)$ can be sold to the grid. The total power flow between node i and the grid is thus $P_i^+(t) + P_i^-(t)$, where flow towards the battery is taken to be the negative direction and thus $P_i^-(t) \leq 0$, and $P_i^+(t) \geq 0$.

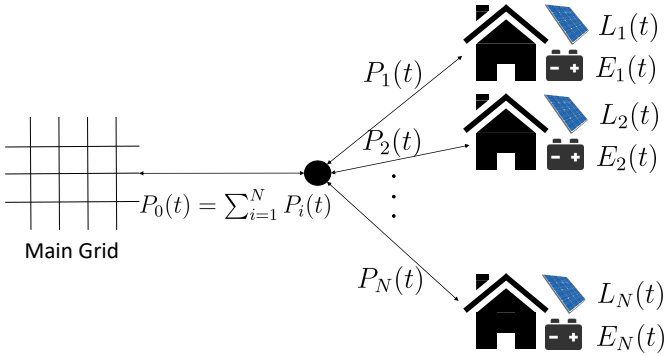

Figure 3.1: Schematic diagram of the battery scheduling problem.

The goal of battery scheduling is to use the projected load forecast and power generation information to minimize the cost of operating the system. This problem can be thought of as an OCP where the charge of a battery E is the state variable and the power provided to the battery, P^+ and taken from the battery, P^- are the controls. Using the discrete-time formulation of this problem as defined in [MFH18] and [MHF18], the cost function is composed of three parts: minimize the cost of energy drawn from the grid/maximize revenue from energy sold to the grid, minimize emptiness of the battery at the end of the time horizon, and minimize large changes in power flow from time step to time step.

For each time step of length τ, the primary objective is to minimize the cost of the power flowing to/from the grid at time t, denoted by $P_0^-(t)$ and $P_0^+(t)$ respectively:

$$\sum_{t \in \mathbb{T}} (a^+(t) P_0^+(t) + a^-(t) P_0^-(t)) \tau,$$

where $a^+(t) \geq 0$ and $a^-(t) \geq 0$ describe the prices for buying and selling electricity at time t and are assumed to be given. However, this objective can result in solutions with some unwanted properties. First, sudden large changes in power flow between time steps are possible, which can degrade battery lifetime and induce a hidden cost [BFR+15]. Second, optimal solutions will generally result in empty batteries at the end of the time horizon T. While

emptying the battery may be optimal over the short term, if the schedule is being repeatedly computed in closed loop then this can result in some suboptimality in future iterations. To compensate for both of these issues, the additional objective functions are defined for each node $i \in \mathbb{N} = \{1, \ldots, N\}$:

$$\gamma(\overline{E}_i - E_i(T))^2 + \beta \sum_{t \in \mathbb{T}} \left((P_i^+(t) + P_i^-(t) - P_i^{AVG})\tau \right)^2,$$

where $\gamma > 0$ is a constant and $P_i^{AVG} = \sum t \in \mathbb{T}(P_i^+(t) + P_i^-(t))/T$ is the average of $P_i(t)$ over all T time steps and $\beta > 0$ is a scaling constant. The term $\gamma(\overline{E} - E_i(T))^2$ is included to penalize the emptiness of each battery at the end of the scheduling horizon.

The sum of all power flowing from the nodes must always be equivalent to the new power flowing to/from the main grid. This results in the following constraint:

$$\forall t \in \mathbb{T}, \qquad P_0(k) = \sum_{i=1}^{N} P_i(t). \tag{3.5}$$

Further constraints in the problem come from the fact that power supplied to/drawn from each battery cannot exceed the capacity/supply. Therefore the capacity/supply of battery i, $E_i(t)$ must satisfy the following inequality constraint for each time step $t \in \mathbb{T}$: $\underline{E}_i \leq E_i(t) \leq \overline{E}_i$, where the dynamics of each of the batteries for an initial state $E_i(0)$ is given by:

$$E_i(t+1) = E_i(t) + ((1-\ell)P_i^+(t) + (1+\ell)P_i^-(t) - L_i(t))\tau, \tag{3.6}$$

where $L_i(t)$ is the given (mean) local power injected/drawn by the load such that PV generation is taken as negative load, and ℓ is the power conversion loss.

Additionally, the following constraints are introduced to discriminate between power flow directions $\forall\, i \in \mathbb{N}_0 = \{0, \ldots, N\}$:

$$P_i^+(t)P_i^-(t) = 0, \quad P_i^+(t) \geq 0, \quad P_i^-(t) \leq 0. \tag{3.7}$$

However, many NLP solvers face convergence issues at $P_i^+(t) = P_i^-(t) = 0$. Here we consider two methods for dealing with the cusp in the feasible set:

relaxation of the equality constraints (3.7) and reformulation as a convex mixed-integer problem.

NLP with Constraint Relaxation

As in the battery scheduling problem of [AOM+18], replacement of the equality constraint in (3.7) with $-\delta \leq P_i^+(t)P_i^-(t) \leq 0$ for some small $\delta > 0$ can avoid some of the numerical issues caused by the constraint $P_i^+(t)P_i^-(t) = 0$ close to the point $P_i^+(t) = P_i^-(t) = 0$.[2] The battery scheduling problem with the constraint relaxation can be summarized as follows:

$$\begin{aligned}
\min_{P^+,P^-} \sum_{t=1}^{T} \Bigg(& \big[a^+(t)P_0^+(t) + a^-(t)P_0^-(t)\big]\tau \\
& + \sum_{i=1}^{N} \Big(\gamma(\overline{E}_i - E_i(T))^2 \qquad (3.8a) \\
& + \beta\big[(P_i^+(t) + P_i^-(t) - P_i^{AVG})\tau\big]^2 \Big) \Bigg) \\
\text{subject to } \forall t \in \{1,\dots,T\} : & \\
E_i(t+1) = E_i(t) + \big[& (1-\ell)P_i^+(t) \\
& + (1+\ell)P_i^-(t) - L_i(t)\big]\tau, \quad \forall i \in \mathbb{N}_0, \qquad (3.8b) \\
E_i(t) \in [\underline{E}_i, \overline{E}_i], & \quad \forall i \in \mathbb{N}_0, \qquad (3.8c) \\
P_0(t) = \sum_{i=1}^{N} P_i(t), & \quad \forall i \in \mathbb{N}_0, \qquad (3.8d) \\
0 \leq P_i^+(t) \leq \overline{P}_i, & \quad \forall i \in \mathbb{N}_0, \qquad (3.8e) \\
\underline{P}_i \leq P_i^-(t) \leq 0, & \quad \forall i \in \mathbb{N}_0, \qquad (3.8f) \\
-\delta \leq P_i^+(t)P_i^-(t) \leq 0, & \quad \forall i \in \mathbb{N}_0. \qquad (3.8g)
\end{aligned}$$

[2] This is known as the Fischer-Burmeister relaxation. More information can be found in [Fis95a] and [Fis95b].

Unfortunately, if δ is chosen too small then the numerical problems persist, and if δ is chosen to be too large then the solution may contain simultaneously bi-directional power flows. This is observed in the results of Section 3.1.2.

Results for NLP Formulation

In this section, Problem (3.8) is solved using ALADIN for a variety of time horizons and batteries. The PV production and load consumption data, L_i, comes from a clean subset of three years worth of data from 300 Australian households. This data is freely available online and the clean subset of 54 households was chosen according to the specifications listed in [RWKM15]. Likewise, the pricing information comes from the Ausgrid pricing policy for a residential load at the time of publication. As the Ausgrid data is discretized into 30 minute intervals, $\tau = 30$ mins. For the battery, the parameters $\underline{E} = 0$, $\overline{E} = 13.5$kWh, $\underline{P} = -5kW$, $\overline{P} = 5kW$ and $\ell = 0.05$ are chosen according to a commercial provider [Tes17]. The parameters γ and β are both chosen to be 1. The Fischer-Burmeister relaxation parameter δ is chosen to be 10^{-2}.

The algorithm is initialized by setting $P_i(t) = L_i(t)\ \forall\ i \in \{1, \dots, N\}, t \in \mathbb{T}$, and $P_0(t) = \sum_i P_i(t)\ \forall\ t \in \mathbb{T}$. This initialization was chosen since it is guaranteed to be feasible. Table 3.1 presents the performance of ALADIN for a variety of problem sizes and time horizons. The results from the centralized interior point method IPOPT [WB05] are shown for comparison. IPOPT is also used by ALADIN to solve its NLP subproblems. For the results of Table 3.1, the ALADIN parameters $\rho = 500$, $\mu = 5000$, and termination threshold $\epsilon = 10^{-3}$ are chosen. Each problem consists of $2(N+1)T$ real-valued decision variables. Thus, the largest case considered involves 5280 variables, for which ALADIN fails to return a solution within the time limit.

Note that during the computation of the centralized solution in trials involving over 200 variables, IPOPT either exceeds its default maximum iteration limit or fails in its restoration phase. In such cases a point close to the local solution is returned at the cost of significant computation time. This issue is likely due to the complexity of the problem, as constraint relaxation or simplification eliminates the issue. Regardless, despite failing for the case of 54 batteries and 48 time steps, results are obtained for smaller problems.

Table 3.1: Results obtained for Problem (3.8).

N	T	IPOPT Time (s)	Alg. 2 Time (s)	Alg. 2 Iter.	$\frac{f_{ALADIN}}{f_{IPOPT}}$
2	2	0.1074	0.1707	6	1.0361
12	2	0.2756	0.2053	9	1.0276
2	12	0.1562	0.4924	13	1.0052
54	2	4.9145	3.7592	68	1.0000
2	48	14.857	2.6398	15	1.0772
12	12	3.8895	8.4472	86	1.0009
12	48	57.642	203.25	32	1.1170
54	12	4.6093	200.66	29	1.0631
54	48	767.06	> 5200	> 12	N/A

Additionally, the Fischer-Burmeister relaxation of the complementarity constraints (3.7) results in a slight bi-directional power flow. Figure 3.2 illustrates this issue.

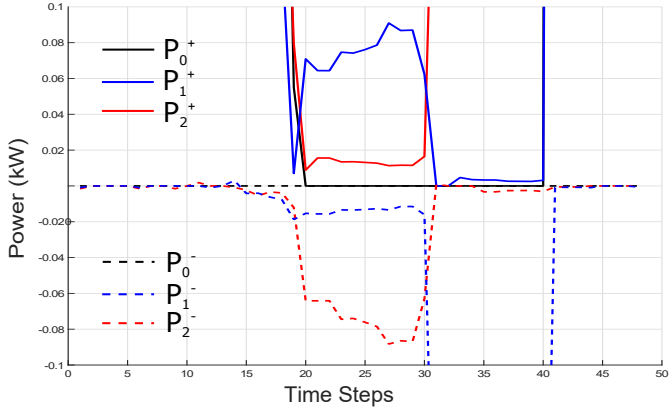

Figure 3.2: Power to/from the grid and each battery for the case of 2 batteries and 48 times steps. Real-valued formulation solved with ALADIN.

One possible solution is to set, $\forall\ i,t,\ P_i^+(t) = 0$ if $P_i^+(t) < |P_i^-(t)|$ and $P_i^-(t) = 0$ otherwise. However, doing so causes an increase in the primal residual $||Ax - b||$ of an entire order of magnitude. This must be taken into account when performing the optimization and thus practical use of the real-valued formulation would require many more iterations than are shown in Table 3.1 in order to account for this phenomenon.

Mixed-Integer Reformulation

The issue of the choice of relaxation parameter δ may be circumvented using integer decision variables. By introducing a binary decision variable z_i for each node $i \in \{0, \dots, N\}$ whose values denote the direction of power flow, the objective functions and constraints associated with each direction can be toggled on or off. Specifically for each time step t, the constraints (3.8e), (3.8f), and (3.8g) are replaced with:

$$0 \leq P_i^+(t) \leq z_i(t)\overline{P}_i, \quad \forall i \in \mathbb{N}_0 \tag{3.9a}$$

$$(1 - z_i(t))\underline{P}_i \leq P_i^-(t) \leq 0, \quad \forall i \in \mathbb{N}_0 \tag{3.9b}$$

$$P_0^+(t) + P_0^-(t) = \sum_{i=1}^{N} P_i^+(t) + P_i^-(t), \quad \forall i \in \mathbb{N}_0 \tag{3.9c}$$

$$z_i(t) \in \{0, 1\}, \quad \forall i \in \mathbb{N}_0 \tag{3.9d}$$

The battery scheduling problem with the integer variables is an MIQP and can be summarized as follows:

$$\begin{aligned}
\min_{P^+,P^-,z} \sum_{t=1}^{T} \Bigg(& \big[a^+(t)P_0^+(t) + a^-(t)P_0^-(t)\big]\tau \\
& + \sum_{i=1}^{N} \Big(\gamma(\overline{E}_i - E_i(T))^2 \\
& + \beta\big[(P_i^+(t) + P_i^-(t) - P_i^{AVG})\tau\big]^2 \Big) \Bigg) && \text{(3.10a)} \\
\text{subject to } \forall t \in \{1,\dots,T\}: & \\
E_i(t+1) = E_i(t) + \big[& (1-\ell)P_i^+(t) \\
& + (1+\ell)P_i^-(t) - L_i(t)\big]\tau, \quad \forall i \in \mathbb{N}_0, && \text{(3.10b)} \\
E_i(t) \in [\underline{E}_i, \overline{E}_i], & \quad \forall i \in \mathbb{N}_0 && \text{(3.10c)} \\
0 \le P_i^+(t) \le z_i(t)\overline{P}_i, & \quad \forall i \in \mathbb{N}_0 && \text{(3.10d)} \\
(1 - z_i(t))\underline{P}_i \le P_i^-(t) \le 0, & \quad \forall i \in \mathbb{N}_0 && \text{(3.10e)} \\
P_0^+(t) + P_0^-(t) = \sum_{i\in\mathbb{N}} P_i^+(t) + P_i^-(t), & \quad \forall i \in \mathbb{N}_0 && \text{(3.10f)} \\
z_i(t) \in \{0,1\}, & \quad \forall i \in \mathbb{N}_0 && \text{(3.10g)}
\end{aligned}$$

Results for MIQP Formulation

As in Section 3.1.2, the PV production and load consumption data comes from [RWKM15], however this section uses the mixed integer formulation of the battery scheduling problem (3.10). As this formulation is exact and does not rely on a Fischer-Burmeister relaxation, no parameter δ is chosen. Apart from this, all other problem parameters remain the same. Due to the inclusion of integer decision variables to reformulate the non-convex complementarity constraints, each problem instance has $2(N+1)T$ real-valued variables and $(N+1)T$ integer decision variables. For the largest problem instance with $N = 54$ and $T = 48$, this results in 5280 real-valued variables and 2640 integer

variables. The results from both Bonmin[3] and Algorithm 5 applied to Problem (3.10) are displayed in Table 3.2. The N/A denotes cases where Bonmin failed to return a solution. The time spent before failure is still recorded.

Table 3.2: Results obtained for Problem (3.10).

N	T	Bonmin Time (s)	Alg. 5 Time (s)	Alg. 5 Iter.	$\frac{f_{ALADIN}}{f_{Bonmin}}$
2	2	0.1533	0.1380	9	1.0014
12	2	0.3335	0.2375	17	1.0020
2	12	0.2742	0.2305	13	1.0061
54	2	0.7612	0.1651	14	N/A
2	48	1.3299	0.6024	23	N/A
12	12	1.8344	0.2449	17	N/A
12	48	1,6787	0.4716	10	N/A
54	12	17.1634	0.4067	14	N/A
54	48	> 3000	1118.5	72	N/A

Problem (3.10) is solved centrally using Bonmin, and in a distributed framework using Algorithm 5. As in Section 3.1.2, the parameters $\rho = 500$, $\mu = 5000$, and termination tolerance $\epsilon = 10^{-3}$ are chosen for Algorithm 5. The iterates of Algorithm 5 for the instance of (3.10) with $N = 2$, $T = 48$ are shown in Figure 3.3.

While Bonmin does terminate for each problem instance, it does not always do so successfully. The N/A values in the f_{ALADIN}/f_{Bonmin} column indicate where the centralized solver fails to return a solution. In the cases where Bonmin does converge, it returns the same values as IPOPT. Figure 3.4 depicts the trajectory for the instance $N = 2$, $T = 48$. Note that for the solution from Algorithm 5, the power flow remains within the feasible region of -5 to 5 kW, and the battery state remains in the feasible region of 0 to 13.5kWh. However, as Bonmin fails to converge for this case, only the initialization is shown.

[3] Bonmin is an open-source local mixed integer solver that uses IPOPT to solve its local non-linear subproblems [BBC+08].

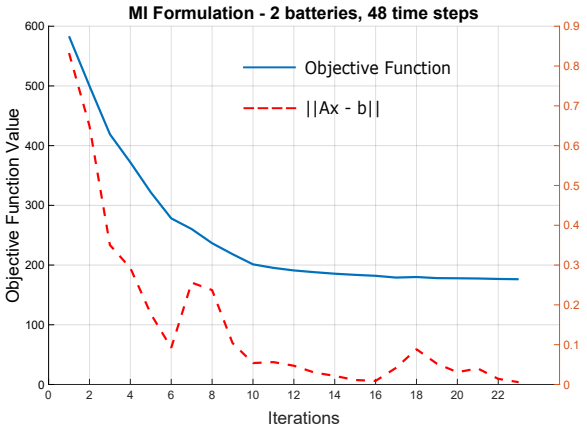

Figure 3.3: Objective value and constraint violation at each iteration of Algorithm 5 for the case of two batteries and forty eight time steps.

An interesting observation to note from the comparison of Tables 3.1 and 3.2 is that the mixed-integer formulation is solved in less time and less iterations than for the real-valued formulation. In some cases, the difference is a factor of four or more. This is in addition to the fact the solutions of (3.10) are exact, while those of (3.8) are not.

3.1.3 Case Study – Reactive Power Dispatch

Section 3.1.2 gives some results for how Algorithm 5 performs for MIQPs, however it is applicable to higher order problems as well [MEHF18]. One particularly difficult MINLP is known as the Reactive Power Dispatch (RPD) problem. The goal of RPD is the minimization of line losses while maintaining voltages within certain limits. This is done by controlling reactive power generation, tap changers, and shunt capacitors. The last two grid components are discrete in nature, and the overall problem is constrained by the highly non-convex AC power flow equations. The result is an MINLP, and thus it is often difficult to quickly obtain solutions, even for relatively small grids.

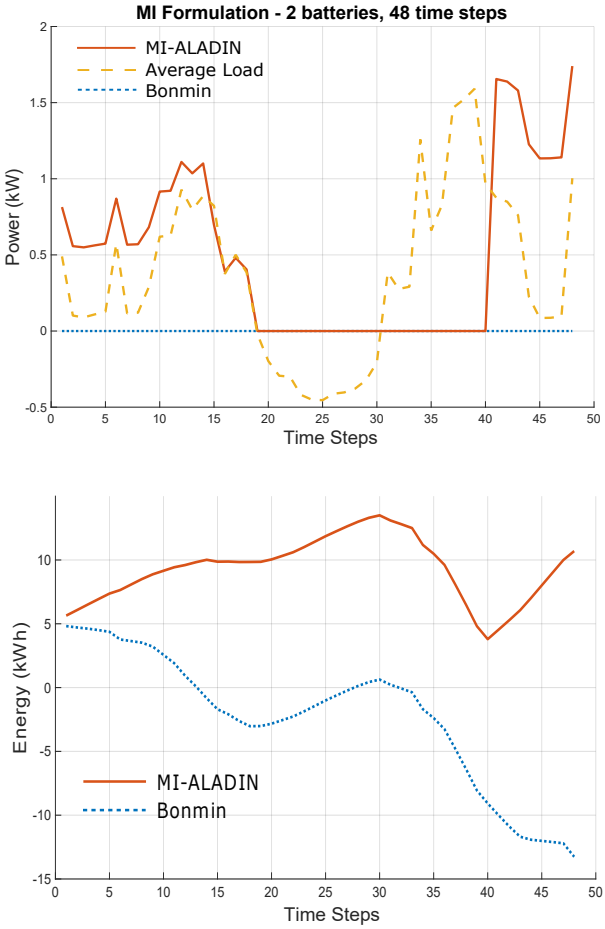

Figure 3.4: Comparison of centralized and hierarchical distributed results. (Top) The average power flow to/from the batteries. (Bottom) The average energy stored in the batteries.

Mathematically, the grid is described by a node set $\mathcal{N}$, a generator set $\mathcal{G}$, a line set $\mathcal{L} \subset \mathcal{N} \times \mathcal{N}$ and a bus admittance matrix $Y = G + jB \in \mathbb{C}^{N \times N}$. The entries $Y_{kl} = G_{kl} + jB_{kl}$ of the bus admittance matrix are given by

$$Y_{kl} = \begin{cases} \sum_{m \in \mathcal{N} \setminus \{k\}} y_{km}, & \text{if} \quad k = l \\ -y_{kl}, & \text{if} \quad k \neq l, \end{cases}$$

where the complex value $y_{kl} \in \mathbb{C}$ is the admittance of the transmission line connecting buses k and l.

For brevity, the real-valued decision variables shall be denoted by

$$u = [v, \theta, p^g, q^g] \in \mathbb{R}^{4|\mathcal{N}|}.$$

This vector collects the voltage magnitudes $v = [v_1, \ldots, v_{|\mathcal{N}|}]$, voltage angles $\theta = [\theta_1, \ldots, \theta_{|\mathcal{N}|}]$,[4] active power generation $p^g = [p^g_1, \ldots, p^g_{|\mathcal{N}|}]$, and reactive power generation $q^g = [q^g_1, \ldots, q^g_{|\mathcal{N}|}]$ at the nodes $\{1, \ldots, |\mathcal{N}|\}$ in the grid. Naturally, the active and reactive power generation are zero for all nodes not in $\mathcal{G}$. Furthermore, in RPD the active power generation p^g, reactive power demand q^d, and active power demand p^d are all fixed, with the exception of the reference bus r. As in optimal power flow, the reference bus has fixed voltage: $v_r = 1$, and $\theta_r = 0$.

Likewise, the discrete decision variables shall be denoted by $z = [s, a]$, where $s = [s_1, \ldots, s_{|\mathcal{N}|}] \in \mathcal{S}$ are the shunt susceptances, and $a_{kl} \in \mathcal{A}_{kl}$ are the transformer tap setting at line $(k, l) \in \mathcal{L}$.[5] For nodes without shunt capacitor or lines that are not connected via a tap-changing transformer, s_k and a_{kl} are

[4] Note that for the sake of brevity, the standard notation $\theta_{kl} = \theta_k - \theta_l$ is used.

[5] Without loss of generality, we assume that the admissible tap and shunt settings are discrete, but not necessarily integer valued. However, they may be transformed into sets of integers with an appropriate constant factor.

zero and one, respectively. The polar form of the reactive power optimization problem is summarized by the following MINLP:

$$\min_{u,z} \sum_{(k,l)\in\mathcal{L}} -G_{kl}((a_{kl}v_k)^2 + (a_{lk}v_l)^2 - 2a_{kl}a_{lk}v_k v_l \cos(\theta_{kl})) \tag{3.11a}$$

$$\text{s.t. } \forall k, l \in \mathcal{N}$$

$$p_k^g - p_k^d = v_k \sum_{l\in\mathcal{N}} a_{lk}a_{kl}(G_{kl}v_l\cos(\theta_{kl}) + B_{kl}v_l\sin(\theta_{kl})), \tag{3.11b}$$

$$q_k^g - q_k^d - v_k^2 s_k = v_k \sum_{l\in\mathcal{N}} a_{lk}a_{kl}(B_{kl}v_l\cos(\theta_{kl}) - G_{kl}v_l\sin(\theta_{kl})), \tag{3.11c}$$

$$\underline{v}_k \le v_k \le \bar{v}_k, \qquad \forall k \in \mathcal{N}, \tag{3.11d}$$

$$\underline{q}_k \le q_k^g \le \bar{q}_k, \qquad \forall k \in \mathcal{N}, \tag{3.11e}$$

where the objective function given by (3.11a) is the loss function for all transmission lines $(k,l) \in \mathcal{L}$.

Commonly used approaches to (3.11) are continuous relaxation of the the discrete decision variables as per [HGA09, YLCW06] and then rounding to the nearest integer after solving an NLP. However, doing so can result in suboptimal results [YLCW06]. Alternatively, one may avoid relaxation and solve the MIP using genetic algorithms [Iba94] or particle swarm optimization [YKF+00]. While good results have been found using these methods [SMY+08], they lack optimality guarantees. Decomposition and parallelization is a promising alternative [MEHF18], however it first requires the problem to be given in a partially separarable form.

Separable Problem Formulation

To be able to use Algorithm 5 to solve (3.11), the problem at hand must be in affine coupled separable form; i.e. the form shown in (3.3). This entails the partitioning of the problem into N regions, where a region R_i contains regional node set $\mathcal{N}_i$ such that $\mathcal{N}_i \cap \mathcal{N}_j = \emptyset$ for all $i \neq j$, and local line sets $\mathcal{L}_i \subseteq \mathcal{N}_i \times \mathcal{N}_i$. Each region is coupled via auxiliary nodes located at the border between two regions. For example, on a line from node k in region R_1 to node ℓ in region R_2, the auxiliary nodes k' and ℓ' are added as illustrated

schematically in Figure 3.5. In addition to the power flow equations (3.11b)-(3.11e) the following coupling constraints hold between the nodes $k \in R_1$ and $k' \in R_2$: $v_k = v_{k'}$ and $\theta_k = \theta_{k'}$. Likewise, $v_\ell = v_{\ell'}$ and $\theta_\ell = \theta_{\ell'}$ for $\ell \in R_2$ and $\ell' \in R_1$.

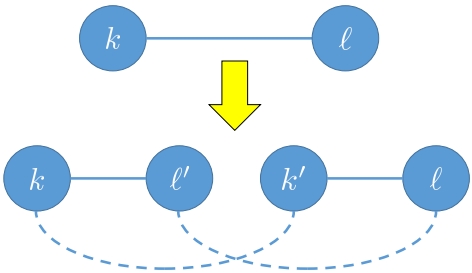

Figure 3.5: An example of how auxiliary buses are constructed and thus how Problem (3.11) can be put in the form of (3.3). Solid lines denote nodes which are coupled by (3.11b)-(3.11e) and dashed lines denote nodes which are affinely coupled. (Top) Two nodes before partitioning. (Bottom) The two original nodes, plus two auxiliary nodes.

The new problem formulation becomes a collection of problems in the form of (3.11), with affine coupling constraints:

$$\begin{aligned} v_{\ell'} &= v_\ell \\ v_k &= v_{k'} \\ \theta_{\ell'} &= \theta_\ell \\ \theta_k &= \theta_{k'} \end{aligned}$$

for each pair of nodes k, ℓ on the ends of an inter-partition line, and their respective auxiliary nodes k', ℓ'. Thus, for each line crossing between regions, two additional buses and four additional equality constraints must be added. Although the resulting problem is mathematically equivalent to the original, this partitioning can add substantial numerical burden to the solution process. Section 4.1 offers some insight into how power grids can be partitioned such that favourable results from distributed methods can be obtained.

Results IEEE 14 Bus Case

As a small benchmark example to evaluate the performance of Algorithm 5 for MINLPs we consider the IEEE 14 bus case. The parameters for the 14 bus case are taken from [YKF+00].[6] Tap changers and shunt capacitors are placed as in [LCW+07] and relevant constants chosen accordingly. Table 3.3 lists the location and feasible set for each decision variable in p.u. of the IEEE 14 case.

Table 3.3: Control variables for the 14 bus case and their admissible values (in p.u.).

Variable	Domain
$[\underline{v}, \overline{v}]$	[0.90,1.10]
a	$\{0.90, 0.91 \ldots, 1.09, 1.10\}$
s_9	$\{0, 0.18\}$
s_{14}	$\{0, 0.18\}$

As no partitions are given in the case file, they are selected as shown in Figure 3.6. For generality, a variety of different partitioning strategies are considered. Partition 1a partitions the grid into approximately equally sized partitions, however Partition 1b was meant to simulate the case where the majority of the grid is controlled by a central entity, but a number of independently controlled enclaves exist, as is the case in Northern Germany [SRLBM12]. Partition 1c investigates the case where every node is contained within its own partition.

Table 3.4 presents the results obtained from both Bonmin and Algorithm 5 for each of the partitions. In [YKF+00], Reactive Tabu Search (RTS) and Particle Swarm Optimization (PSO) were used to solve this problem for the 14 bus case. Shown in Table 3.5 is a comparison of their best results and the fastest solution obtained by Algorithm 5.[7] Algorithm 5 terminates for every case with a termination threshold of $\varepsilon = 10^{-3}$ and $\rho = 5 \cdot 10^2$.

[6] The power system test case archive is available at `www.ee.washington.edu/research/pstca`.

[7] It should be noted that the runtimes for PSO and RTS are those reported in the original papers and thus part of the difference is due to differing hardware.

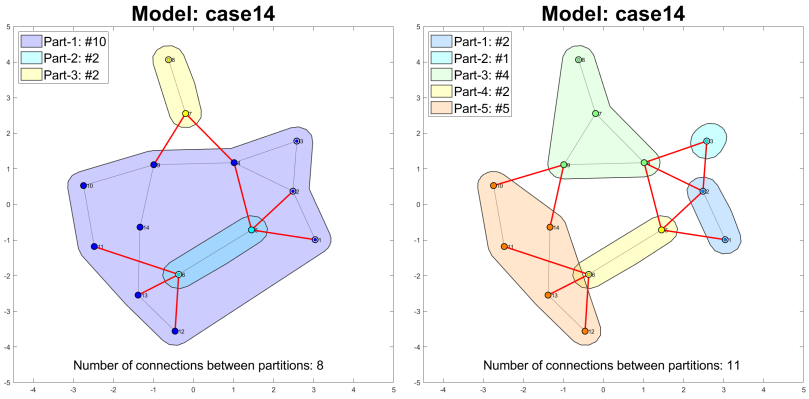

Figure 3.6: Partitions 1a and 1b for the IEEE 14 bus case. The red lines denote the connections between partitions.

Table 3.4: Results obtained for the 14 bus case.

	Part. 1a	Part. 1b	Part. 1c
Bonmin Time (s)	4.554	4.554	4.554
Algorithm 5 Time (s)	5.526	30.485	154.950
Algorithm 5 Iter.	287	277	814
Bonmin Obj. (MW)	12.684	12.6841	12.684
Algorithm 5 Obj. (MW)	12.588	13.2940	12.682

Table 3.5: Results from Partition 1a and those of [YKF+00].

	Initial Losses	RTS	PSO	Algorithm 5
Min. loss (MW)	13.3933	13.2366	13.3509	12.5884
Calc. time (s)	-	19.5	16.5	5.5267

As for the difference between the objective values in Table 3.5, it should also be noted that there is no unique notion of “closest local solution” in Mixed-Integer

Programming (choice of norm etc.). Moreover, the solution tolerance ε in the consensus constraints ($\|Ax - b\|_1 \leq \varepsilon$) can be understood as a slight problem relaxation; i.e. it enlarges the feasible set compared to the centralized solution. Thus, the integer nature of the problem and how it is handled by Algorithm 5 appears to be source of this difference. It should also be noted that the values in Table 3.5 come from [YKF+00], and thus use hardware that is 20 years older than that used for Algorithm 5.

Results for IEEE 30 Bus Case

For a slightly larger benchmark example, we consider the IEEE 30 bus case. As in Section 3.1.3, the parameters for the IEEE 30 bus case are taken from the power system test case archive with tap changers and shunt capacitors are placed as in [LCW+07]. Table 3.6 lists the location and feasible set for each decision variable in p.u. of the IEEE 30 case. The total initial line losses for the IEEE 30 bus case are 20.8674MW.

Table 3.6: Control variables for the IEEE 30 bus case and their admissible values (in p.u.).

Variable	Domain
$[\underline{v}, \overline{v}]$	[0.90,1.10]
a	$\{0.90, 0.92 \ldots, 1.08, 1.10\}$
s_{10}	$\{0, 0.20\}$
s_{24}	$\{0, 0.04\}$

As no partitions are given in the case file, they are selected as shown in Figure 3.7. As in the 14 bus case, Partition 2a partitions the grid equally and Partition 2b has a single large partition with a pair of enclaves, in analogy to Partition 1b. Likewise, Partition 2c mirrors Partition 1c and splits every node into their own partitions. The results returned by Bonmin and Algorithm 5 are shown in Table 3.7.

Both Bonmin and Algorithm 5 are initialized as in the 14 bus case and with termination threshold $\varepsilon = 10^{-3}$. However, the parameter ρ is chosen to be

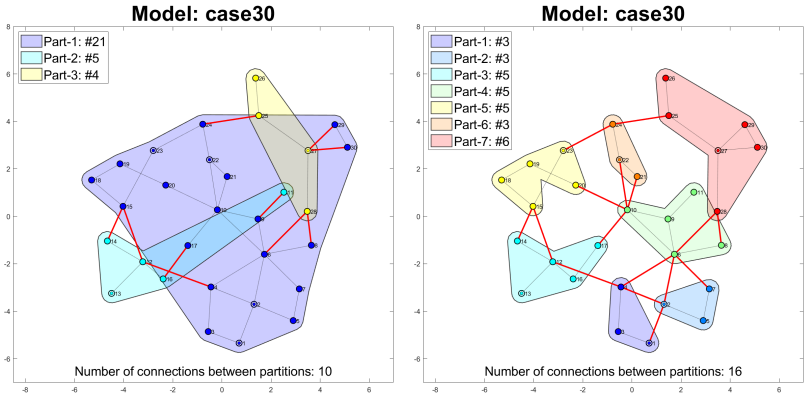

Figure 3.7: Partitions 2a and 2b for the IEEE 30 bus case. The red lines denote the connections between partitions.

Table 3.7: Results obtained for the 30 bus case.

	Partition 2a	Partition 2b	Partition 2c
Bonmin Time (s)	8.8515	8.8515	8.8515
Algorithm 5 Time (s)	41.0716	83.7757	107.8566
Algorithm 5 Iterations	302	446	1383
Bonmin Obj. (MW)	20.1249	20.1249	20.1249
Algorithm 5 Obj. (MW)	20.1379	20.7797	22.2365

10^4 for the 30 bus case. In general, larger values of ρ will lead to slower convergence but if ρ is chosen to be too small then Algorithm 5 may not converge at all. Furthermore, as the problem size increases, the other terms of the subproblem objective functions take on larger values and thus larger values of ρ must be chosen to compensate. The iterates of Algorithm 5 for two of the tested partitions are shown in Figure 3.8.

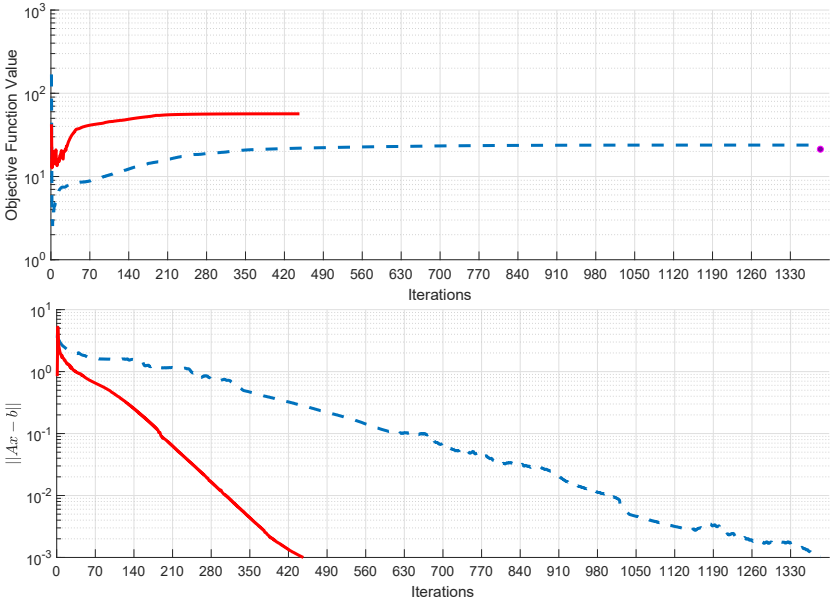

Figure 3.8: (Top) The objective function value per iteration of Algorithm 5 for Partitions 2b and 2c in semi-log scale. The purple dot indicates the solution returned by the centralized solver. (Bottom) The consensus constraint violation per iteration for Partitions 2b and 2c.

Overall, the best results seem to come from the case where the partitions are split evenly. These partitions allow for a moderate degree of parallelization without introducing too many auxiliary variables into the subproblem computations. The worst results come from Partitioning 2c, where every node is in its own partition. This leads to the inclusion of auxiliary variables at every branch and is almost certainly the cause of the long run times and large number of iterations required for partition 2c.

3.2 Partially Distributed Outer Approximation

As shown in Sections 3.1.2 and 3.1.3, Algorithm 5 can be used to solve mixed-integer optimization problems with a partially separable structure. However, as

mentioned in Section 3.1.1, its convergence properties are somewhat limited. In this section a partially distributed outer approximation algorithm (PaDOA) for mixed integer optimization is presented which has stronger convergence properties than Algorithm 5.

Recall from Section 2.1.2 and Section 3.1 how distributed algorithms typically alternate between solving small-scale decoupled subproblems with augmented objective functions and an equality constrained coupling problem. This structure will serve as both an inspiration and a template for the distributed mixed-integer optimization algorithm presented in this chapter. One key distinction between PaDOA and previously described distributed algorithms is that the algorithm presented in this section does not used augmented Lagrangians. This is due to the fact that Lagrange multipliers give a notion of sensitivity, which is of limited use in integer programming. More information on duality in mixed-integer programming can be found in [BOW16, Joh79, Jer78].[8] Furthermore, while Algorithm 5 solves equality an constrained QP in the coupling step, PaDOA solves an MILP which allows for application to problems of the form:

$$\begin{aligned} \min_{x,z} \; & \sum_{i=1}^{N} f_i(x_i, z_i) \\ & \text{s.t. } \forall i \in \{1, \dots, N\}, \\ & x_i \in \mathcal{X}_i, \\ & z_i \in \mathcal{Z}_i, \\ & \sum_{i=1}^{N} A_i x_i + B_i z_i = c, \end{aligned} \tag{3.12}$$

where $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_N$ and $conv(\mathcal{Z}) = conv(\mathcal{Z}_1 \times \cdots \times \mathcal{Z}_N)$ are polytopes.

Remark 3. *Note that the coupling constraint $Ax + Bz = c$ is completely general. Suppose that some (integer or real-valued) variables w are coupled via the inequalities $Dw \leq k$.One can then include an additional auxiliary variable w' and reformulate the coupling inequality as $Dw + w' = k$, with appropriate constraint on w'.*

[8] For an intuition of this phenomenon consider how small changes to constraints in integer programming will either have no impact on the solution or a large one, depending on whether an integer feasible point is gained/lost in the change.

The main idea of PaDOA is to construct piecewise lower bounding functions $\Theta_i^\star(x^\star, z^\star)$ which together form a piecewise affine lower bound on the objective function f. This lower bounding function together with the coupling constraints forms an MILP, which is solved by the central coordinator to update the integer solution $z \leftarrow z^+$.

One way to construct functions $\Theta_i^\star : \mathcal{X} \times \mathcal{Z}_i \to \mathbb{R}$ is by taking a linearization[9] of f at the solution to a subproblem of the form:

$$\begin{aligned} V_i(z) = \min_{x,\zeta_i} \; & f_i(x_i, \zeta_i) + \Psi_i(x, z) \\ \text{s.t. } & Ax + \tilde{B}_i z + B_i \zeta_i = c \\ & x \in \mathcal{X} \\ & \zeta_i \in \mathcal{Z}_i, \end{aligned} \tag{3.13}$$

where $[\tilde{B}_i \; B_i] = [B_1, \ldots, B_{i-1}, B_{i+1}, \ldots, B_N, B_i]$, and the function $\Psi_i(x, z)$ is defined as

$$\forall (x, z) \in \mathcal{X} \times \mathcal{Z}, \qquad \Psi_i(x, z) = \sum_{j \neq i} f_j(x_j, z_j)$$

and are introduced in order to keep the x-dependence of the remaining summands, i.e., all objective terms whose index is not equal to i. This is in analogy to Gauss-Seidel or more general block-coordinate descent methods [Tse01, Wri15] in the sense that a partial decoupling is obtained by fixing some of the integer variables while others are optimized.

Notice that if Problem (2.16) is an MICP then the lower bounding functions $\Theta_i^\star$ must satisfy the condition

$$V_i(z) \leq \min_{x \in \mathcal{X}, \zeta \in \mathcal{Z}_i} \Theta_i^\star(x, \zeta) \quad \text{s.t. } Ax + \tilde{B}_i z + B_i \zeta_i = c. \tag{3.14}$$

In practice the function $\Theta_i^\star$ can be stored by maintaining a set of hyperplane coefficients as explained in detail in Section 2.2.3. Moreover, in order to avoid the accumulation of too many hyperplanes, one can discard all hyperplanes

[9] Specifically, this linearization can be constructed as per (2.20) and the hyperplanes of (2.19) as defined in Section 2.2.3.

that are inactive at the optimal solution of the last MILP relaxation, because this operation does not affect the right hand expression of (3.14).

Given the aforementioned means of constructing lower bounding functions, PaDOA proceeds according to five main steps: In the first step, the partially decoupled MICPs of the form (3.13) are solved by using a traditional outer approximation method. Under the assumption that the original MICP is feasible, the partially decoupled MICPs are feasible, too. Thus, the outer approximation solvers will return optimal integer solutions $\zeta_i^\star$ and associated piecewise affine lower bounds $\Theta_i^\star$ such that (3.14) is satisfied. The second step updates the associated upper bound U and the third step updates the piecewise affine lower bound. In the third step, a large scale MILP problem is solved which yields a lower bound $\Theta(x^+, z^+)$ on the objective value $V^\star$ of (3.13). Thus, the difference between the current upper and lower bounds,

$$U - \Theta(x^+, z^+) \ ,$$

can be used as a termination criterion, which is implemented in the final step of the algorithm. If the termination is not successful, the integer variables z are updated, and the algorithm subsequently proceeds to the next iteration. These steps are summarized in Algorithm 6.

Algorithm 6: Partially Distributed Outer Approximation (PaDOA)

Input: Initial guess $z \in \mathcal{Z}$ and a numerical tolerance $\varepsilon > 0$.

Initialization: Set $\Pi = \emptyset$, $\Theta(\cdot, \cdot) = -\infty$, and $U = \infty$.

Repeat:

1. Solve for all $i \in \{1, \ldots, N\}$ the partially decoupled MICPs

$$V_i(z) = \min_{x,y,\zeta_i} \quad f_i(x_i, \zeta_i) + \Psi_i(x, z) \qquad \text{s.t.} \quad \begin{cases} Ax + \tilde{B}_i z + B_i \zeta_i = c \\ x \in \mathcal{X} \\ \zeta_i \in \mathcal{Z}_i \end{cases} \tag{3.15}$$

   where $\Psi_i(x, z) = \sum_{j \neq i} f_j(x_j, z_j)$. If (3.15) is infeasible, terminate and return a certificate of infeasibility. Otherwise, update the set $\Pi \leftarrow \Pi \cup \{\zeta_i^\star\}$ and construct a piecewise affine model $\Theta_i^\star$ such that condition (3.14) is satisfied.

2. Update the upper bound $U \leftarrow \min\{U, V_1(z), \ldots, V_N(z)\}$ and construct the piecewise lower bounding function $\Phi(x, z, \Pi)$ as in (2.20).

3. Update the lower bound

$$\forall x \in \mathcal{X},\ \forall z \in \mathcal{Z},\ \Theta(x, z) \leftarrow \max \left\{ \Theta(x, z)\ ,\ \Phi(x, z, \Pi)\ ,\ \max_i\ \Theta_i^\star(x, z_i) \right\}$$

4. Solve the MILP problem

$$(x^+, z^+) \in \operatorname*{argmin}_{x \in X, z \in Z} \quad \Theta(x, z) \quad \text{s.t.} \quad Ax + Bz = c \tag{3.16}$$

5. If $U - \Theta(x^+, z^+) \leq \varepsilon$, terminate. Otherwise, update $z \leftarrow z^+$ and go to Step 1.

Output: (x^+, z^+).

Notice that the main difference between Algorithm 4 and Algorithm 6 is the introduction of partially decoupled MICP problems that can be solved separately and which contain much fewer integer variables than the original MICP. The theoretical results in Section 3.2.1 will elaborate further on the benefits of this alternation strategy. Moreover, in Section 3.2.3 a numerical case study is examined which illustrates the practical advantages of Algorithm 6.

3.2.1 Convergence Analysis

In this section we provide a concise overview of the convergence properties of Algorithm 6. Namely, that for a class of MICPs, Algorithm 6 converges to the global solution after finitely many iterations, and after just a single iteration if initialized at the optimal solution.

The following theorem establishes the fact that Algorithm 6 converges after a finite number of iterations under exactly the same conditions under which convergence of Algorithm 4 can be established.

Theorem 3. *For MICP* (3.12)*, Algorithm 6 terminates after a finite number of steps.*

Proof. We may assume that the coupled equality constraint is feasible, as infeasibility would be detected immediately in Step 1 of Algorithm 6. Similar to the proof of Theorem 2, the main idea is to show that the cardinality of the set Π increases in every iteration of Algorithm 6 under the assumption that the termination criterion is not satisfied. Let us assume that the integer solution z^+ from Step 1 satisfies $z^+ \in \Pi$. Then we have

$$U \leq V_i(z^+) \overset{(3.14)}{\leq} \min_{x\in\mathcal{X},\zeta\in\mathcal{Z}_i, Ax+\tilde{B}_i z+B_i\zeta_i=c} \Theta_i^\star(x,\zeta) \ .$$

Furthermore, by the construction of Θ in Step 3 we also have

$$\min_{x\in\mathcal{X},\zeta\in\mathcal{Z}_i, Ax+\tilde{B}_i z+B_i\zeta_i=c} \Theta_i^\star(x,\zeta) \leq \min_{x\in\mathcal{X},z\in\mathcal{Z}, Ax+\tilde{B}_i z+B_i\zeta_i=c} \Theta^\star(x,\zeta) = \Phi(x^+,z^+) \ ,$$

for all i. Putting these inequalities together yields $U - \Phi(x^+, z^+) \leq \epsilon$, a contradiction to our assumption that the termination criteria are not satisfied. Thus, it must be the case that $z^+ \notin \Pi$. However, the cardinality of $\mathcal{Z}$ is finite and thus this process can only repeat finitely many times. It follows that the termination criteria of Algorithm 6 must be satisfied in a finite number of iterations. □

Although finite convergence is a useful property, the actual number of iterations is upper bounded only by the number of integer feasible points. For many

MIPs, this can be more than the number of atoms in the universe.[10] However, a stronger convergence property can also be proven for Algorithm 6. If the initial guess is chosen as $z = z^\star$, where $z^\star$ is an optimal solution of (3.12), then Algorithm 6 converges after just one iteration. This is a property which is not shared by Algorithm 4, from which Algorithm 6 is based.

Theorem 4. *Let $(x^\star, z^\star)$ be a minimizer of MICP* (3.12)*. If Algorithm 6 is initialized with $z = z^\star$ then the termination criterion in Step 4 is satisfied. In other words, the algorithm terminates after one step.*

Proof. Let $V^\star = \sum_{i=1}^{N} f_i(x_i^\star, z_i^\star)$ denote the optimal value of (3.12). If Problem (3.12) is an MICP, then the partially decoupled optimization problems are all feasible and return piecewise affine lower bounds that satisfy the termination condition (3.14) with $V_i(z^\star) = V^\star$, i.e., we have

$$V^\star \leq \min_{x \in X, \zeta \in Z_i} \Theta_i^\star(x, \zeta) \quad \text{s.t.} \quad Ax + \tilde{B}_i z^\star + B_i \zeta = c \tag{3.17}$$

for all $i \in \{1, \ldots, N\}$. By construction, the function Θ is an upper bound on Θ_i (for any i), and thus

$$\min_{x \in X, \zeta \in Z_i, Ax + \tilde{B}_i z^\star + B_i \zeta_i = c} \Theta_i^\star(x, \zeta) \leq \min_{x \in X, z \in Z, Ax + Bz = c} \Theta(x, z) = \Theta(x^+, z^+) \,,$$

where (x^+, z^+) denotes the solution of from Step 4 of Algorithm 6. By substituting the above inequalities we find that

$$V^\star \leq \Theta(x^+, z^+) \,.$$

and thus it follows that

$$U - \Theta(x^+, z^+) = V^\star - \Theta(x^+, z^+) \leq \epsilon \,.$$

Thus, the termination condition is satisfied and Algorithm 6 terminates after the first step. □

[10] It is easy to verify that a MIP with over 273 binary variables has over 10^{82} integer-feasible points.

Recall from Section 2.2.1 that other global optimization algorithms, such as branch-and-bound, often find a global solution quite quickly and then spend a considerable amount of time proving optimality. This makes the statement of Theorem 4 to be of fundamental relevance and a very favourable property of PaDOA. In fact, Theorem 4 implies that global optimality of a point $z^\star \in Z$ can be verified by solving the N instances of the partially decoupled MICPs and the master MILP (3.16). Notice that this result is not in conflict with existing results from the field of complexity theory, because the master MILP (3.16) remains NP-hard [GJ79, MK87].

Remark 4. *One limitation of Algorithm 6 is that it is only applicable to MICPs, due to the requirement that the lower level solvers return piecewise level models Θ_i, which satisfy the termination condition* (3.14) *(this assumption is only reasonable if strong duality holds) and which need to be global lower bounds on f. These properties are in general not satisfied if one considers more general non-convex MINLPs. The result of Theorem 4 relies heavily on the convexity of the functions f_i on the convex hull of $X_i \times Z_i$, although this fact is not highlighted explicitly in the proof.*

3.2.2 Case Study – Battery Scheduling

Once again we shall use the mixed-integer formulation of the battery scheduling problem presented in Section 3.1.2 as means of benchmarking. The battery scheduling problem is scalable both spatially and temporally, and fits perfectly into the class of problems for which Algorithm 6 is applicable, making it an excellent benchmark problem. For consistency, the same constraints, PV production, and load consumption data are used as in Section 3.1.2. The results of Algorithm 6 with a termination tolerance of $\epsilon = 10^{-3}$ are compared in Table 3.8 against those of Bonmin.

While Bonmin does terminate for each problem instance, it does not always do so successfully. As in Table 3.2, the N/A values in the f_{PaDOA}/f_{Bonmin} column indicate where the centralized solver fails to return a solution within 1000s.

In the cases where Bonmin does converge, it returns approximately the same solution as Algorithm 6. There are multiple solutions to Problem (3.10) and it is worth noting the flatness of the solutions returned by Algorithm 6. This

Table 3.8: Results obtained for Problem (3.10).

N	T	Bonmin Time (s)	Alg. 6 Time (s)	Alg. 6 Iter.	$\frac{f_{PaDOA}}{f_{Bonmin}}$
2	2	0.1533	0.2611	1	1.000
12	2	0.3335	0.4085	8	1.000
2	12	0.2742	1.2763	16	1.003
54	2	0.7612	0.9183	10	N/A
2	48	1.3299	0.2544	1	N/A
12	12	1.8344	2.5633	5	N/A
12	48	1,6787	55.541	42	N/A
54	12	17.1634	399.47	76	N/A
54	48	> 1000	500.89	60	N/A

linearity is likely due to the way in which PaDOA uses hyperplanes to reach and verify a solution. As an example, the trajectory for the instance $N = 2$, $T = 48$ is depicted Figure 3.9. Such solutions do have an advantage in practical implementation as they require less precise and constant controls of the batteries.

3.2.3 Case Study – Thermostatically Controlled Loads

As a second benchmark example for Algorithm 6, the problem of thermostatically controlled loads is used. Like battery scheduling, it is also scalable both spatially and temporally, and has several parameters which can change the shape of the objective function. It is an important problem in the planning and operation of a heating and/or cooling system and involves the scheduling of so-called Thermostatically Controlled Loads (TCLs). These are devices that regulate the temperature of a room or building within a certain user-defined interval known as a "deadband." By taking future temperature and electricity costs into account, the heaters/coolers can be designed to coordinate and operate more efficiently than they would by following a simple control scheme that

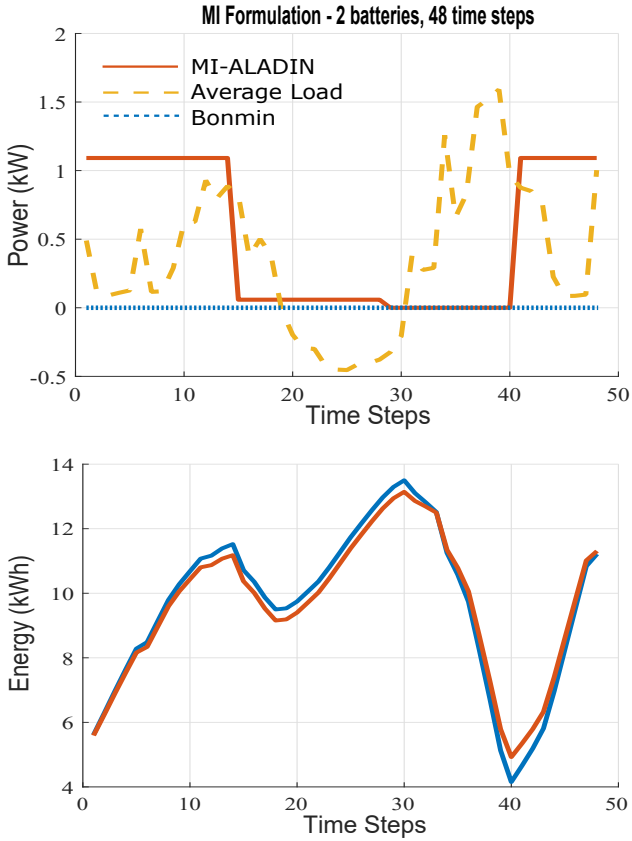

Figure 3.9: Comparison of Bonmin (blue) and PaDOA (red) results. (Top) The average power flow to/from the batteries. The dashed yellow line corresponds to the local power generation/demand. (Bottom) The average energy stored in the batteries.

activates when the deadband is reached. Schematics for two possible problems are illustrated in Figure 3.10.

The optimal operation strategy is especially difficult to determine when a non-constant cost function is introduced for a population of heterogeneous TCLs [ZKF+12]. Although the cost function is typically taken to be the cost of electricity, it can also be used to represent user-discomfort from temperatures

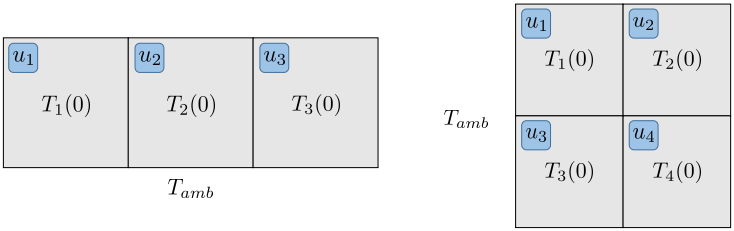

Figure 3.10: Two room configurations with controlled cooling elements u_i, ambient temperature T_{amb} and initial temperatures $T_i(0)$.

that deviate from the setting but are still within the deadband. This can be a useful addition that distinguishes solutions with equal monetary cost, but more temperature fluctuation.

The controllable elements are modelled with an "on" and an "off" setting and thus the resulting scheduling problem is a binary MIP with R regions and a discretized time horizon H, consisting of hour-long time steps:

$$\min_{T(\cdot),u(\cdot)} \sum_{t=0}^{H-1} c(t)u(t) + \gamma(T_i(t) - T_{ref}(t))^2, \tag{3.18a}$$

subject to $\forall i \in \{1, \ldots, R\}$,

$$\underline{T}_i \leq T_i(t) \leq \overline{T}_i, \quad \forall t \in \{0, \ldots, H\} \tag{3.18b}$$

$$u_i(t) \in \{0, 1\}, \quad \forall t \in \{0, \ldots, H-1\} \tag{3.18c}$$

$\forall t \in \{0, \ldots, H\}$,

$$T_i(t+1) = T_i(t) \tag{3.18d}$$
$$+ b_i u_i(t) + a_i \left(\frac{T_i(t) + T_{amb}(t) + \sum_{j \in N_i} T_j(t)}{|N(i)| + 2} - T_i(t) \right),$$

where $c(t)$ is the vector of device costs at time t, γ is a comfort parameter, $\underline{T}_i$ and $\overline{T}_i$ are the deadband temperature limits of device i, a_i and b_i are heat transfer parameters, $T_{amb}(t)$ is the ambient temperature at time t and $N(i)$

are the number of regions neighbouring i. The equations in (3.18) are based on the formulation given in [KMC11], but with linear dynamics modelling the interaction between each region. Specifically, Equation (3.18d) takes an average of the current and surrounding temperatures to update the temperature of the next time step. As the temperature for each room is more influenced by past temperatures than neighbouring rooms, Problem (3.18) is partitioned spatially for Algorithm 6, with each room in its own partition.

Sections 3.2.3, 3.2.3, and 3.2.3 present results for first, second, and fourth order variants of Problem 3.18. Regardless of the order of the problem, each problem instance with R rooms and H time steps contains $R(H+1)$ real-valued variables and RH discrete variables. The results of Sections 3.2.3, 3.2.3, and 3.2.3 use ambient temperature taken from [Deu17] for two days in June 2017 in the Karlsruhe (Germany) area. High prices of \$25.67/kW are set from 2pm to 8pm (time steps 6 to 12 and 29 to 35) with low and medium prices of 2.46cents/kW and 4.62cents/kW in all other time steps. Each region is initialized at 20 degrees with $a_i = 0.2$ and $b_i = -2$.

Results for MILP

If the comfort parameter γ is taken to be zero then Problem (3.18) is linear and partially separable with affine coupling in both its discrete and real-valued variables. Shown in Tables 3.9 and 3.10 are the simulation results for each configuration, respectively. The results of Algorithm 6 are compared with results obtained from a B&B approach as implemented in Bonmin with default settings [BBC+08] as well as the commercial MIQP solvers Gurobi [Gur19] and CPLEX [IBM19]. An example solution for the 3 room case is depicted in Figure 3.11.

At first glance, the results from Tables 3.9 and 3.10 may seem surprising since the 4 room case has more space to keep cool but nonetheless is able to do so at a lower cost than the 3 room case. This is due to an insulation effect that the 4 room configuration enjoys. With the activation of two coolers in the first six time steps, the room temperatures can stay within their deadbands for the entire 48 hour period. In contrast, the 3 room configuration is more susceptible to the ambient temperature and requires more use of the coolers. This also seems to have increased the computational complexity of the problem and requires

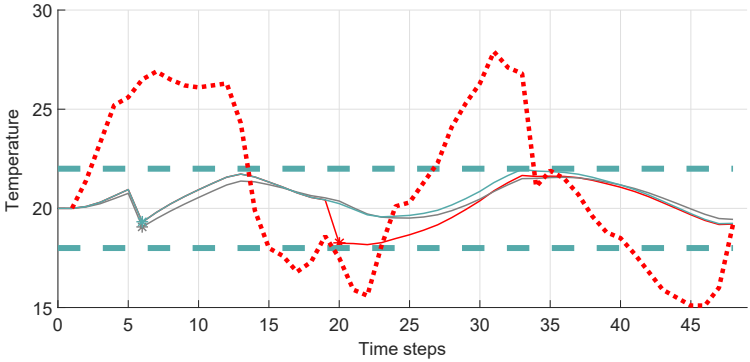

Figure 3.11: The solid lines are the state trajectories for 3 rooms and 48 time steps, with activations highlighted by stars. The red dotted and blue dotted lines are the ambient temperature and deadband, respectively.

Table 3.9: Results for Problem (3.18) with 3 rooms.

	Time steps:	8	24	48	62
Alg. 6	obj.	0	13.86	16.32	21.23
	time(s)	0.15	0.20	0.89	5.23
	iter.	2	1	2	2
B-OA	obj.	0	13.86	16.32	21.23
	time(s)	0.15	31.06	1,345	52,395
B-BB	obj.	0	13.86	16.32	21.23
	time(s)	0.15	29.19	480.27	828.08
Gurobi	obj.	0	13.86	16.32	21.23
	time(s)	0.22	0.56	0.85	4.39
CPLEX	obj.	0	13.86	16.32	21.23
	time(s)	0.07	0.16	0.54	2.81

Table 3.10: Results for Problem (3.18) with 4 rooms.

	Time steps:	8	24	48	62
Alg. 6	obj.	0	9.24	9.24	11.69
	time(s)	0.17	0.25	0.39	1.41
	iter.	2	2	2	2
B-OA	obj.	0	9.24	9.24	11.69
	time(s)	0.14	28.34	48.42	1,610
B-BB	obj.	0	9.24	9.24	11.69
	time(s)	0.18	33.11	56.83	822.48
Gurobi	obj.	0	9.24	9.24	11.69
	time(s)	0.22	0.37	0.51	1.08
CPLEX	obj.	0	9.24	9.24	11.69
	time(s)	0.07	0.09	0.11	0.48

more time for the 3 room case to be solved than the 4 room case. It should be noted that several initializations were tested and the solution times were not significantly affected, implying that this was not the cause of the runtime differences in the two cases.

One of the advantages of using a distributed method is the ability to solve problems that would be otherwise intractable for a centralized solver. Tables 3.9 and 3.10 show results for cases containing up to 496 variables, but even larger problems may be considered. To this end, Table 3.11 shows results for more rooms. Here, the room configuration is instead arranged such that the rooms are in a line. While unrealistic for most buildings, this setup is realistic for the temperature control of a train or rooms next to a corridor. Mathematically, this example differs somewhat from the other two. While the other problems have a significant amount of coupling between the control variables, that is not the case with such a long room configuration. This sparsity allows for Algorithm 6 to outperform both Gurobi and CPLEX (applied to the centralized problem).

Table 3.11: Results for Problem (3.18) with a linear room configuration.

	R :	10	12	18	20
	H :	48	36	24	24
Alg. 6	obj.	37.26	41.88	50.82	55.44
	time(s)	9,127.5	8,294	853.7	16,658
	iter.	2	2	2	2
B-OA	obj.	N/A	N/A	N/A	N/A
	time(s)	N/A	N/A	N/A	N/A
B-BB	obj.	37.26	41.88	50.82	55.44
	time(s)	3,076.0	3,261.5	2,999.1	4,658.4
Gurobi	obj.	N/A	41.88	50.82	N/A
	time(s)	N/A	56,842	31,926	N/A
CPLEX	obj.	37.26	41.88	50.82	N/A
	time(s)	13,416	12,885	57,108	N/A

For the entries which include N/A, the algorithm failed to return a solution within the time limit of 245,000s.

Results for MIQP

As in Section 3.2.3, Problem (3.18) is solved for a variety of room configurations. However, the results in this section are with a comfort parameter $\gamma = 1$.[11] This choice substantially changes the problem at hand since Problem (3.18) is now a convex MIQP rather than an MILP. The results for this new problem as obtained by Algorithm 6, Bonmin, Gurobi, and CPLEX are shown in Tables 3.12, 3.13, and 3.14 for the 3, 4, and multi-room scenarios, respectively.

[11] This value was chosen to be one to allow for an equal weighting of comfort and cost.

Table 3.12: Results obtained for Problem (3.18) with a 3-room configuration.

	Time steps:	8	24	48	62
Alg. 6	obj.	17.83	60.15	104.07	134.43
	time(s)	0.24	0.49	1.36	3.17
	iter.	4	3	3	3
B-OA	obj.	17.83	60.15	104.07	134.43
	time(s)	4.36	46.94	399.49	1,702.60
B-BB	obj.	17.83	60.15	104.07	134.43
	time(s)	9.09	70.40	502.76	1,152.90
Gurobi	obj.	17.83	60.15	104.07	134.43
	time(s)	0.44	0.50	0.66	0.83
CPLEX	obj.	17.83	60.15	104.07	134.43
	time(s)	0.19	0.27	0.25	0.54

Shown in Figure 3.12 are the trajectories obtained for the three-room scenario with a temperature deviation penalization. In contrast to Figure 3.11, a quadratic penalty term is used to model discomfort caused by deviations from the set temperature. Indeed, the solution with $\gamma = 1$ yields a trajectory with a similar number of activations as when $\gamma = 0$ but with temperature trajectories that stay much closer to the middle of the deadband.

Interestingly, the results of Tables 3.12, 3.13, and 3.14 are generally obtained more quickly than in the MILP case of Section 3.2.3, even though one might expect the higher order problem to be more difficult to solve. The most likely explanation for this phenomenon is that the MIQP is better conditioned with a unique minimizer. This makes proving the optimality of a given point much easier for a B&B algorithm than it is for a problem where many of the nodes in the decision tree have the same or similar values. In contrast, Algorithm 6 converges quite quickly for the MILP case since it is under-approximating a linear problem with linear functions and thus can prove optimality quickly and with few iterations. More supporting hyperplanes are needed in the MIQP case and thus we see a couple more iterations and a slightly longer runtime for

Table 3.13: Results for Problem (3.18) with 4-rooms.

	Time steps:	8	24	48	62
Alg. 6	obj.	21.68	52.26	99.68	134.43
	time(s)	0.37	0.65	8.82	31.07
	iter.	5	3	3	4
B-OA	obj.	21.68	52.26	99.68	134.43
	time(s)	14.61	6.78	613.95	7,024.10
B-BB	obj.	21.68	52.26	99.68	134.43
	time(s)	30.39	11.13	661.95	1,495.30
Gurobi	obj.	21.68	52.26	99.68	134.43
	time(s)	0.41	0.54	0.72	2.31
CPLEX	obj.	21.68	52.26	99.68	134.43
	time(s)	0.15	0.16	0.41	2.15

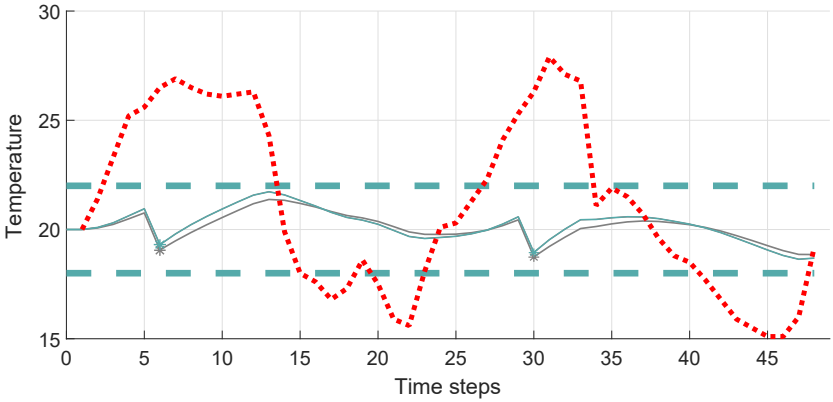

Figure 3.12: Trajectories are defined as in Figure 3.11 for the three room scenario, except with comfort parameter $\gamma = 1$.

Table 3.14: Results for Problem (3.18) with a linear 7 room configuration.

	Time steps:	8	24	48	62
Alg. 6	obj.	39.50	112.08	201.19	267.04
	time(s)	1.57	2.55	260.2	1022.38
	iter.	4	3	4	3
B-OA	obj.	39.50	112.08	N/A	N/A
	time(s)	358.12	1,833.9	N/A	N/A
B-BB	obj.	39.50	112.08	201.19	267.04
	time(s)	593.17	955.77	6,095.6	15,582
Gurobi	obj.	39.50	112.08	201.19	267.04
	time(s)	0.95	0.72	10.83	65.59
CPLEX	obj.	39.50	112.08	201.19	267.04
	time(s)	0.93	0.77	6.53	17.50

Algorithm 6. As shown in Figure 3.13, the upper and lower bound converge quite quickly for the MIQP case, but require several iterations to refine the solution.

It is also worth noting that the majority of the runtime used by Algorithm 6 for the MIQP problem instance is spent in the MILP coupling step. A breakdown of the runtime per iteration is reported in Table 3.15. This may imply that quadratic under-approximating functions and an MIQP coupling step could actually be more efficient for certain problems than the current implementation with affine supporting hyperplanes. This would also have the added benefit of allowing for applicability to a larger problem class.

Results for Higher Order MICP

One of the advantages of Algorithm 6 is that it is applicable to a relatively large class of problems. While Section 3.2.3 shows favourable results for both Gurobi and CPLEX, if the problem were adjusted slightly such that it were

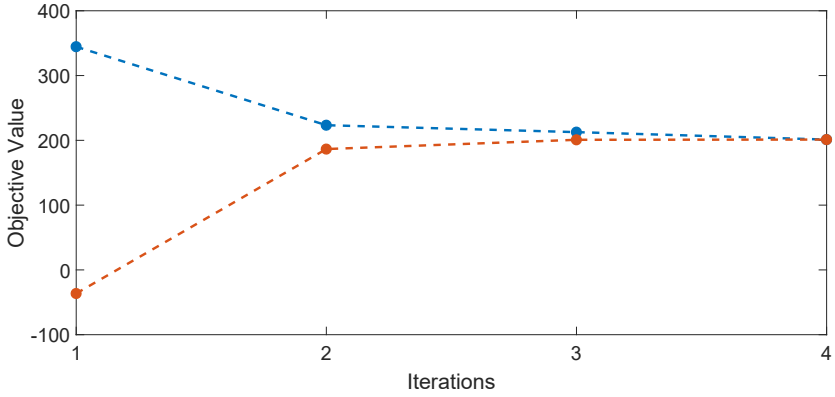

Figure 3.13: Progression of the upper and lower bounds during each iteration while solving the 2^{nd}-order version of Problem 3.18 with 7 rooms and 48 time steps. The blue line depicts the progression of the upper bound and the red shows that of the lower bound.

Table 3.15: Runtime breakdown of Algorithm 6 applied to the 2^{nd}-order version of Problem 3.18 with 7 room TCL problem with 48 time steps.

	Iter. 1	Iter. 2	Iter. 3	Iter. 4
MINLP time (s)	0.27	0.20	0.20	0.20
MILP time(s)	2.20	86.89	82.62	87.62
Hyperplane time (s)	0.001	0.005	0.002	0.002

no longer an MIQP then these solvers would no longer be applicable. For example, if the objective function of Problem (3.18) were replaced with

$$\min_{T(\cdot),u(\cdot)} \sum_{t=0}^{H-1} c(t)u(t) + \gamma(T_i(t) - T_{ref}(t))^4,$$

then it would still be solvable via PaDOA, but not Gurobi or CPLEX. The results for a variety of such problem configurations are shown below in Table 3.16.

Table 3.16: Results obtained for Problem (3.18), but with a 4^{th} order objective function.

		Alg. 6			Bonmin	
Rooms	Time steps	Obj.	Time(s)	Iter.	Obj.	Time(s)
3	8	16.72	4.20	5	16.72	7.51
3	24	76.86	19.14	4	76.86	308.64
3	48	115.62	95.56	3	115.62	547.98
3	62	141.44	405.19	6	141.44	779.41
4	8	21.73	4.35	6	21.73	16.93
4	24	44.94	6.47	3	44.94	9.93
4	48	86.72	151.58	5	86.72	477.60
4	62	120.71	175.78	6	120.71	781.46
7	8	38.41	8.87	7	38.41	374.38
7	24	122.09	34.86	3	122.09	1291.29
7	48	202.54	1,631.6	4	202.54	2,830.93
7	62	263.91	3614.54	4	263.91	9453.89

Note that Algorithm 6 returns the same, global solution as Bonmin, and does so in less time. The runtime difference is particularly striking for the 7 room scenarios as these contain the most variables and have the greatest potential for parallelization.

It is also worth noting that while the majority of the runtime used by Algorithm 6 for the MIQP problem instance is spent in the MILP coupling step, the majority of the time spent solving the 4^{th}-order problem is instead in the partially decoupled MINLP subproblems. This implies higher parallelization potential, and that the linear under-approximating functions are sufficient for this case. A breakdown of the runtime per iteration for the 4^{th}-order problem instance is given in Table 3.17.

Table 3.17: Runtime breakdown of Algorithm 6 applied to the 4^{th}-order version of Problem 3.18 with 4 rooms and 8 time steps.

	Iter. 1	Iter. 2	Iter. 3	Iter. 4	Iter. 5	Iter. 6
MINLP time (s)	0.73	0.75	0.67	0.65	0.65	0.66
MILP time(s)	0.03	0.04	0.04	0.04	0.04	0.04
Hyperplane time (s)	0.004	0.006	0.002	0.002	0.002	0.002

3.3 Distributed Branch and Bound

While Section 3.1.1 establishes some convergence properties for Algorithm 5 and Sections 3.1.2 and 3.1.3 demonstrate some promising numerical results, there are several drawbacks that the method faces. Namely, that the convergence properties are quite restrictive and there is no means of handling consensus constraints that include integer decision variables. Algorithm 6 is shown in Section 3.2 to be capable of handling certain MIPs with affinely coupled integer and real-valued variables. Despite the favourable results shown in Sections 3.2.2 and 3.2.3, Algorithm 6 is limited to MICPs with a polytopic feasible set. The distributed mixed-integer optimization algorithm described in this section is designed to address these drawbacks.

The steps of the algorithm are shown in Algorithm 7, but the main idea behind it is to follow a basic B&B method, as described in Section 2.2.1 except each of the NLP subproblems are solved using a state-of-the-art distributed algorithm. In contrast to parallel branch and bound, which seeks to simultaneously investigate several nodes of the B&B tree [KK84], Algorithm 7 investigates a single node at a time, but in a distributed, parallelizable manner. In doing so, it is possible to be run in a distributed computing context where it is impractical or impossible to transmit full problem information to a single memory location.

As in Section 2.2.1, the vector $\mathbf{Z}_n$ is used to denote the integer variables which fixed and/or relaxed at node n. For a given ordering of individual integer decisions, the k^{th} element of $\mathbf{Z}_n$ corresponds to the k^{th} integer variable. If this element is feasible, then the corresponding integer decision is fixed to $\mathbf{Z}_n(k)$ and relaxed otherwise.

Starting with a candidate node list consisting of the root node $\mathcal{S} = \{0\}$, an upper bound U, lower bound L, and a numerical tolerance $\varepsilon > 0$ Algorithm 7 has the following steps:

1. Choose a node n from the candidate node list $\mathcal{S}$. Generate the corresponding vector $\mathbf{Z}_n$.
2. Solve NLP($\mathbf{Z}_n$) =

$$\text{(2.16) with (2.16c) swapped for } \begin{cases} z = \mathbf{Z}_n & \text{if } \mathbf{Z}_n \in Z \\ z \in \operatorname{conv}(Z) & \text{otherwise} \end{cases}$$

3. If the objective value $J^\star(\mathbf{Z}_n)$ of NLP($\mathbf{Z}_n$) is below the current upper bound U, then add the child nodes of n to the set S.
4. Compute new upper and lower bounds U and L.
5. Terminate if $U - L$ is sufficiently small.

Algorithm 7: Distributed Branch and Bound for Affinely Coupled MINLPs

Input: Upper bound U, lower bound L, candidate node set $\mathcal{S} = \{0\}$ and a numerical tolerance $\varepsilon > 0$.

While $\mathcal{S} \neq \emptyset$**:**

1. **Choose** a node $n \in \mathcal{S}$. Let $\mathbf{Z}_n$ denote its corresponding vector.
2. (Optional) **Predict/verify** feasibility of NLP($\mathbf{Z}_n$). If NLP($\mathbf{Z}_n$) is feasible, proceed to Step 3, else go back to Step 1.
3. **Solve** NLP($\mathbf{Z}_n$) for $J^\star(\mathbf{Z}_n)$ and $z^\star(\mathbf{Z}_n)$ with a distributed algorithm.
4. **If** $z^\star(\mathbf{Z}_n)$ is feasible and $J^\star(\mathbf{Z}_n) < U$, then $U \leftarrow J^\star(\mathbf{Z}_n)$ and proceed to Step 5.
   If $J^\star(\mathbf{Z}_n) > U$ proceed to Step 1.
   Else add the child nodes of n to $\mathcal{S}$ and proceed to Step 5.
5. **Update** $L \leftarrow \min_{\tilde{n} \in \mathcal{P}(\mathcal{S})} \{J^\star(\mathbf{Z}_{\tilde{n}})\}$, where $\mathcal{P}(\mathcal{S})$ denotes the set of parent nodes for a node set $\mathcal{S}$.
   If $U - L \leq \varepsilon$ terminate.
   Else proceed to Step 1.

Output: $J^\star(n)$ and solution $z^\star(\mathbf{Z}_n)$ from NLP($\mathbf{Z}_n$).

Remark 5. *Note that no explicit branching rules are given in Algorithm 7, and these must be provided by the user. Fortunately, there has been much research dedicated to the development and refinement of B&B heuristics and techniques. Some examples of which can be found in [Cla99b]. Section 4.2.3 can also assist in this regard, if sufficient a priori knowledge of the problem is available.*

3.3.1 Convergence Properties

As with standard B&B, Algorithm 7 tests every feasible integer point at most once and thus is guaranteed to terminate in a finite number of iterations. However, as shall be shown in Sections 3.3.3 and 3.3.4, Algorithm 7 typically terminates much earlier. As mentioned in Section, 2.2.1 this is due to the fact that, rather than enumerating each of these integer-feasible points of the MIP and solving using brute force, B&B can solve the problem more efficiently by navigating a decision tree made up of nodes which correspond to NLP subproblems where some integer decisions are fixed and others are relaxed.

The distributed structure of Algorithm 7 results in an iterative method nested inside another iterative method and hence, would be expected to converge quite slowly as a result. There are several ways in which this issue can be alleviated, such as warm starting primal and dual variables from the solutions of previous similar subproblems, or using a method to first determine the feasibility of the subproblem.

Furthermore, if ALADIN is used as the distributed algorithm for the NLP subproblems, then it is guaranteed to converge monotonically provided that ρ is properly chosen [HKJD18]. One can then use U and L to allow for early termination if the objective value $f_k^\star$ of iteration k violates $L < f_k^\star < U$. One common feature of distributed optimization algorithms is that their accuracy and rate of convergence can be influenced by the choice of penalty parameter(s) and numerical termination tolerance. As shown in Sections 3.3.3 and 3.3.4, this can be taken advantage of to quickly obtain slightly suboptimal solutions. This is particularly useful in embedded systems where tight time constraints are present.

3.3.2 MICP-Feasibility Algorithm

The runtime of Algorithm 7 is highly dependent on how quickly it can solve its NLP subproblems. To this end, we present an algorithm for determining the feasibility of MICPs of the form (3.12). In general, finding a feasible solution to Problem (3.12) is as difficult as solving it. While some feasibility algorithms for MIPs have already been developed [BFL07, FL10], they are all centralized methods and are generally inapplicable to a distributed computing context where problem information may not be globally accessible. The coupling constraint $Ax + Bz = c$ is the key challenge when determining feasibility of Problem (3.12) in a decentralized manner, and some method for determining feasibility of this global constraint using only local information is required. First, let us introduce an algorithm for determining the feasibility of an MICP in a centralized manner. Shown in Algorithm 8 is such a method.

Algorithm 8: MICP feasibility solver

Input: $A \in \mathbb{R}^n \times \mathbb{R}^\ell$, $B \in \mathbb{R}^m \times \mathbb{R}^\ell$, $C \in \mathbb{R}^\ell$, and feasible set $\mathcal{X} \times \mathcal{Z} \subset \mathbb{R}^n \times \mathbb{Z}^m$.

Initialize: $\mathbf{Z} = \emptyset$.

1. Solve the matrix equation for $(x, z) \in \mathbb{R}^n \times \mathbb{Z}^m$

$$\begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} = C \tag{3.19}$$

   If there is no point in $\mathbb{R}^n \times \mathbb{Z}^m$ that satisfies (3.19) then FALSE.

2. If $\mathcal{Z} = \mathbf{Z}$ then FALSE. Otherwise, for some $z' \in \mathcal{Z} - \mathbf{Z}$ such that (3.19) is satisfied, calculate the set

$$\mathbf{x}(z') = \{x \in \mathbb{R}^n \mid \begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} x \\ z' \end{bmatrix} = C\}$$

   and update $\mathbf{Z} \leftarrow \mathbf{Z} \cup z'$.

3. Apply the Gilbert-Johnson-Keerthi distance algorithm [GJK88] to $\mathbf{x}(z') \cap G$, where $G = \{(x, z) | x \in \mathcal{X}, z \in \mathcal{Z}\}$.
   If $\mathbf{x}(z') \cap G \neq \emptyset$, then TRUE. Otherwise, proceed to Step 2.

Output: TRUE or FALSE

An visualization of Algorithm 8 is given in Figure 3.14.

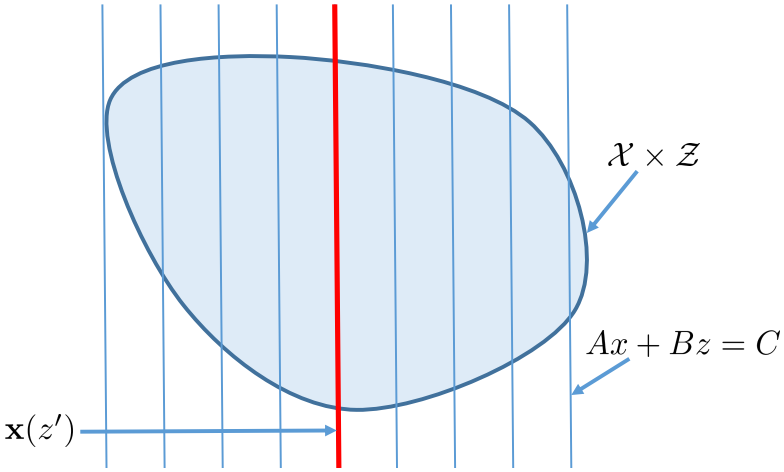

Figure 3.14: A visualization of an iteration of Algorithm 8 for a problem of the form (3.12).

The algorithm begins in Step 1 by solving a relaxation of the feasibility problem and can be useful if $Ax + Bz = C$ is dense or has a high rank. If there is another means of characterizing the points that satisfy the consensus constraints $Ax + Bz = C$ then it may be applied as an alternative. For example, if $A = 0$ then the consensus constraints form a linear Diophantine system of equations, for which polynomial-time algorithms are available for obtaining all integer solutions [Fru77].

Algorithm 8 mostly consists of algebraic operations to construct the sets $\mathbf{x}(z')$ and the GJK algorithm, with Gaussian elimination available as an option to quickly determine infeasibility. It is important to note that the GJK algorithm requires the input convex sets to be of a particular shape in order to achieve the best performance. For example, if the convex set G in Step 3 is neither polytopic nor spherical, then it may be necessary to first create a polytopic approximation of the set.

As is shown in Section 3.3.2, if the consensus constraints have a high rank then Algorithm 8 ultimately turns into several applications of the Gilbert-Johnson-

Algorithm 9: Gilbert-Johnson-Keerthi Distance Algorithm

Input: Compact convex sets $P \subset \mathbb{R}^n$ and $Q \subset \mathbb{R}^n$, and initial points $y_1, \ldots, y_m \in Y = P - Q$ such that $1 \leq m \leq n+1$.
Initialize: $V_0 = \{y_1, \ldots, y_m\}$ and $k = 0$.

1. For some set $\{\lambda_1, \ldots, \lambda_m\}$ such that $\lambda^i > 0$ for all i and $\sum_{i=1}^m \lambda^i = 1$, calculate:
$$v_k = \sum_{i=1}^{m} \lambda^i y_i.$$
2. If $|v_k|^2 + h_Y(v_k) = 0$,
where $h_Y(v_k) = \max\{x \cdot v_k \,|\, x \in Y\}$, then terminate.
Otherwise, continue to Step 3.
3. Set $V_{k+1} = \hat{V}_k \cup \{s_Y(-v_k)\}$,
where $\hat{V}_k \subset V_k$ has at most n elements and satisfies $v_k \in conv(\hat{V}_k)$, and $s_k(-v_k) \in Y$ satisfies
$h_Y(v_k) = s_Y(-v_k) \cdot -v_k$.
Finally, update $k \leftarrow k+1$ and proceed to Step 1.

Output: v_k.

Keerthi (GJK) distance algorithm. The interested reader can find details on GJK algorithm in [GJK88], however it is summarized in Algorithm 9.

Computational Complexity

The overall complexity of Algorithm 8 primarily depends on the construction of the sets $\mathbf{x}(z')$, the arithmetic complexity of the GJK algorithm, and the number and dimension of the subspaces computed in Step 2.

Suppose that there are M points $z' \in \mathbb{Z}^m$ such that $\exists x' \in \mathbb{R}^n$ and (x', z') satisfies (3.19). Additionally, let ν_i be the dimension of each of the subspaces $\mathbf{x}(z'_i)$ for $i \in \{1, \ldots, M\}$. In Step 2, the construction of each set has complexity ν_i^2 and thus, the overall Step has complexity $\sum_{i=1}^M \nu_i^2$. Likewise, each application of the GJK algorithm has complexity $n+m$, as $n+m$ is the dimension of $[A\ B]$. Thus, Algorithm 8's overall arithmetic complexity will be bounded above by $\mathcal{O}(n^3) + M \max\{\nu_1^2, \ldots, \nu_M^2\} + n$.

In principle, M can be quite large. If $\mathcal{Z}$ has dimension m, then $M \geq 2^m$ and Algorithm 8 has non-polynomial complexity. As the NP-completeness of integer programs has been established, this is hardly surprising [PS82]. However, if the consensus constraints form a low-dimensional subspace in $\mathbb{R}^n \times \mathbb{Z}^m$, then Algorithm 8 becomes efficient. For example, if $rank(B) = m - 1$ then the consensus constraints form a linear subspace and Algorithm 8 will have computational complexity $\mathcal{O}(n^3)$.

Decentralized Variant

Notice that each step of Algorithm 8 is parallelizable. Indeed, a decentralized variant of the algorithm can be constructed given that the coupling matrices are decomposed column-wise into $A = [A_1, \ldots, A_k], B = [B_1, \ldots, B_k]$, and $C = [C_1, \ldots, C_k]^\top$. Step 1 can then be solved via accelerated projection based consensus [ARLAH19]. For N processors, this algorithm consists of first applying the following update rule for each processor i:

$$\begin{bmatrix} x_i \\ z_i \end{bmatrix} \leftarrow \begin{bmatrix} x_i \\ z_i \end{bmatrix} + \gamma P_i \left(\begin{bmatrix} \overline{x} \\ \overline{z} \end{bmatrix} - \begin{bmatrix} x_i \\ z_i \end{bmatrix} \right)$$

along with the master problem:

$$\begin{bmatrix} \overline{x} \\ \overline{z} \end{bmatrix} \leftarrow \frac{\nu}{N} \sum_{i=1}^{N} \begin{bmatrix} x_i \\ z_i \end{bmatrix} + (1 - \nu) \begin{bmatrix} \overline{x} \\ \overline{z} \end{bmatrix}$$

where $\gamma \in \mathbb{R}$ and $\nu \in \mathbb{R}$ are chosen parameters, and

$$P_i = I - [A_i \ B_i]^\top \left([A_i \ B_i][A_i \ B_i]^\top \right)^{-1} [A_i \ B_i]$$

is the projection matrix onto the nullspace of $[A_i \ B_i]$.

Likewise, Step 2 can be performed using a similar decomposition. Step 3 can also be performed in parallel, provided that a decomposition of $\mathcal{X} \times \mathcal{Z}$ into $\mathcal{X}_1 \times \cdots \times \mathcal{X}_N \times \mathcal{Z}_1 \times \cdots \times \mathcal{Z}_N$ is possible. Assuming such a decomposition, Step 3 proceeds to calculate $\mathbf{x}_i(z_i') \cap G_i$, where $G_i = \mathcal{X}_i \times \mathcal{Z}_i$ for $i = 1, \ldots, N$.

Remark 6. *As written, Algorithm 8 only returns an indication of whether or not the given optimization problem is feasible. However, it is possible to use it in order to obtain an explicit feasible point* (x', z') *where* z' *is the solution of Step 2 and* x' *is the solution of the GJK algorithm applied to* $\mathbf{x}(z') \cap G$ *in Step 3.*

3.3.3 Case Study – Battery Scheduling

As a first benchmark example we shall consider the problem formulation presented in Section 3.1.2. Recall that this problem is generally an MIQP, without any coupling between integer variables, and with integer decisions for every battery *and* time step. The cost function is relatively flat, and many solutions have near optimal objective values. This makes the problem difficult to solve in a B&B framework since many of the nodes in the decision tree will have the same or similar values. Nonetheless, Algorithm 7 can still be applied to this problem and its results are shown in Table 3.18. As each of the NLP subproblems are convex, ADMM is used as the distributed algorithm, due to its ease of implementation and convergence guarantees for this class of problems.[12] An iteration limit of 1000 is used for both algorithms, with a termination tolerance of $\epsilon = 10^{-3}$ for Algorithm 7.

Recall that ADMM can sometimes require many iterations before converging to an optimal solution, depending on how the parameters are initialized and updated. Thus, when an iteration limit (or other stringent termination criteria) are set, then ADMM may return a suboptimal value. This is the case for some subproblems solved by Algorithm 7, and the reason for it's suboptimality in certain in most problem instances. In the largest case, with 54 batteries and 48 time steps, Algorithm 7 terminates after the limit of 1000s without a feasible solution. Even for the smallest case where an optimal solution is returned in the time limit, Algorithm 7 still requires much more time than the centralized algorithm.

The results of Table 3.18 demonstrate some of the limitations of the current implementation of Algorithm 7. Despite having the potential for parallelization,

[12] If the NLP in Step 1 of Algorithm 7 is non-convex, then ADMM may fail to converge and a more advanced algorithm, like ALADIN may be needed.

Table 3.18: Results obtained for Problem (3.10).

N	T	Bonmin Time (s)	Alg. 7 Time (s)	Alg. 7 Iter.	$\frac{f_{DB\&B}}{f_{Bonmin}}$
2	2	0.1533	26.079	41	1.000
12	2	0.3335	0.8147	1	2.591
2	12	0.2742	> 1000	931	2.9013
54	2	0.7612	0.6114	1	2.7649
2	48	1.3299	> 1000	686	1.3925
12	12	1.8344	1.1852	1	2.2002
12	48	1,6787.0	62.323	38	2.6006
54	12	17.1634	1.7890	1	2.5921
54	48	> 1000	> 1000	171	N/A

good convergence properties, and applicability to a broad class of problems, Algorithm 7 performs relatively poorly on the battery scheduling problem in comparison with Algorithm 5 and Algorithm 6.

3.3.4 Case Study – Abstract MIQP

As a second benchmark example we consider MIQPs of the following form:

$$\begin{aligned} \min_{x,z} \; & \sum_{i=1}^{N} a_1(x_i + z_i)^2 + a_2 x_i^2 + a_3 x_i + a_4 z_i \\ \text{subject to, } & Ax + Bz = 0, \text{ and }, \forall i \in \{1, \ldots, N\} : \\ & b_1 x_i^2 + 2x_i z_i + z_i^2 + b_2 x_i + b_3 z_i \leq b_4, \\ & -3 \leq x_i \leq 3, \\ & z_i \in \{-1, 0, 1\}. \end{aligned} \tag{3.20}$$

where $a_1, a_2, a_3, a_4, b_1, b_2, b_3, b_4 \in \mathbb{R}$.

Problem (3.20) is solved for a variety of parameters using prototypical implementations of a centralized B&B method, and Algorithm 7. To solve the NLPs of Step 1, Algorithm 7 uses ADMM to perform the step in a distributed manner, and IPOPT is used to solve the NLP subproblems. A numerical termination tolerance of $\varepsilon = 10^{-3}$ is used in all cases.

The simulation results are shown in Table 3.20. All of the respective problems are generated such that Problem (3.20) are MICPs. The implemented branch-and-bound algorithm starts with a depth-first branching strategy to obtain an upper bound and then seeks to improve the lower bound as quickly as possible. The ADMM subalgorithm of Algorithm 7 uses initial $\rho = 0.1 f(x_0, z_0)$, where x_0 and z_0 are the solution of the previous subproblem. The parameter ρ is updated according to the rule shown in (3.21). There are many alternative methods for improving the convergence of ADMM via update rules for the penalty parameter ρ [HYW00, WL01], however it is observed that the rule:

$$\rho \leftarrow \begin{cases} 2\rho & \text{if } ||r_k||_2^2 > ||r_{k-1}||_2^2, \\ 0.5\rho & \text{if } \Delta f_k > \Delta f_{k-1}, \\ 1.02\rho & \text{else} \end{cases} \tag{3.21}$$

results in relatively fast convergence of the primal residual r.[13]

For the results displayed in Table 3.20, the discrepancy in number of iterations is due to an iteration of Algorithm 7 being found to be integer feasible, while the IPOPT solution for the same subproblem in the standard B&B algorithm are off by 10^{-8}, and thus the upper bound is not updated and the centralized B&B algorithm continued. Without this upper bound, less nodes are able to be pruned and it takes longer for standard B&B to terminate with $U - L \leq \varepsilon$. The difference in time per iteration is dependent on how quickly ADMM can converge, which itself is highly dependent on the choice of ADMM parameters [GTSJ15, LFPL15, Bol13]. Figure 3.15 illustrates this phenomenon. Overall, Table 3.20 reflects other results in distributed mixed integer optimization which show great speed-up for poorly condition problems, but less so for well-conditioned problems [MFH+20].

[13] Where $\Delta f_k = f(x_k, z_k) - f(x_{k-1}, z_{k-1})$.

Table 3.19: Instances of Problem (3.20) used to benchmark the performance of Algorithm 7.

Prob.	N	$\lvert\mathcal{X}\rvert$	$\lvert\mathcal{Z}\rvert$	a_1	a_2	a_3	a_4	b_1	b_2	b_3	b_4	A	B
1	3	300	3	1	1	1	1	1	1	1	5	[0,...,0]	[1,-1,...,1]
2	10	300	10	1	1	-1	-1	1	-1	-1	5	[0,...,0]	[1,-1,...,1]
3	5	250	5	1	9	-10	-10	1	1	1	4	[0,...,0]	[1,-1,...,1]
4	5	250	5	0	0	-1	-1	1	1	1	3	[0,...,0]	[1,-1,...,1]
5	5	250	5	0	0	1	1	1	1	1	2	[0,...,0]	[1,-1,...,1]
6	5	250	5	0	0	1	1	1	-1	-1	0	[0,...,0]	[1,-1,...,1]
7	10	1000	10	0	0	1	1	1	-1	-1	0	[0,...,0]	[1,-1,...,1]
8	10	1000	10	0	0	1	1	1	-1	-1	0	[1,2,...,10]	[0,...,0]
9	10	500	10	0	0	1	1	1	0	0	3	[0,...,0]	[1,-1,...,1]
10	20	20	20	0	0	10	-10	1	-1	1	3	[0,...,0]	[1,-1,...,1]
11	10	100	10	0	0	1	-1	10	0	-2	4	[1,0,...,1,0]	[1,0,...,1,0]
12	15	150	15	0	0	1	10	10	0	-2	0	[0,...,0]	[1,-1,...,1]
13	15	1500	15	0	0	1	10	10	0	-2	0	[0,...,0]	[1,-1,...,1]
14	15	15	15	0	0	1	10	10	0	-2	0	[0,...,0]	[1,-1,...,1]
15	20	60	20	0	0	1	10	10	0	-2	0	[0,...,0]	[1,-1,...,1]

Table 3.20: Results obtained for Problem (3.20) with the parameters of Table 3.19.

	Alg. 7			B&B		
Prob.	obj.	iter.	time (s)	obj.	iter.	time (s)
1	-0.749	2	0.887	-0.75	4	2.044
2	-2.5	181	64.649	-2.5	181	63.418
3	-88.833	15	3.776	-88.833	11	4.040
4	-6.514	4	1.120	-6.514	78	28.390
5	-10	5	0.741	-10	116	27.199
6	0	5	0.740	0	29	5.250
7	0	12	3.949	0	80	85.361
8	0.180	60	15.279	0	64	26.001
9	-17.321	9	1.358	-17.321	195	> 100
10	-523.607	4	0.977	-523.607	4	0.420
11	-37.272	3	0.578	-37.321	3	0.416
12	-0.0015	44	5.563	-0.0015	45	6.593
13	-134.105	111	51.782	-146.834	81	64.388
14	-0.00047	44	6.154	-0.00047	45	64.388
15	-0.001	59	8.471	-0.001	60	7.105

3.4 Summary and Comparison

All of the algorithms presented in this chapter have different properties, with varying strengths and weaknesses. For example, although Mixed-Integer ALADIN has weak convergence and optimality properties it still demonstrates good results for a wide class of problems. On the other hand, PaDOA has excellent convergence properties but is more limited in applicability and only sees its best results for MILPs and higher order convex MINLPs. Finally, Distributed Branch & Bound is fully distributed and applicable to a wide range of problems; it has excellent theoretical convergence and optimality guarantees, however in practice its performance is somewhat lacklustre. Table 3.21 lists

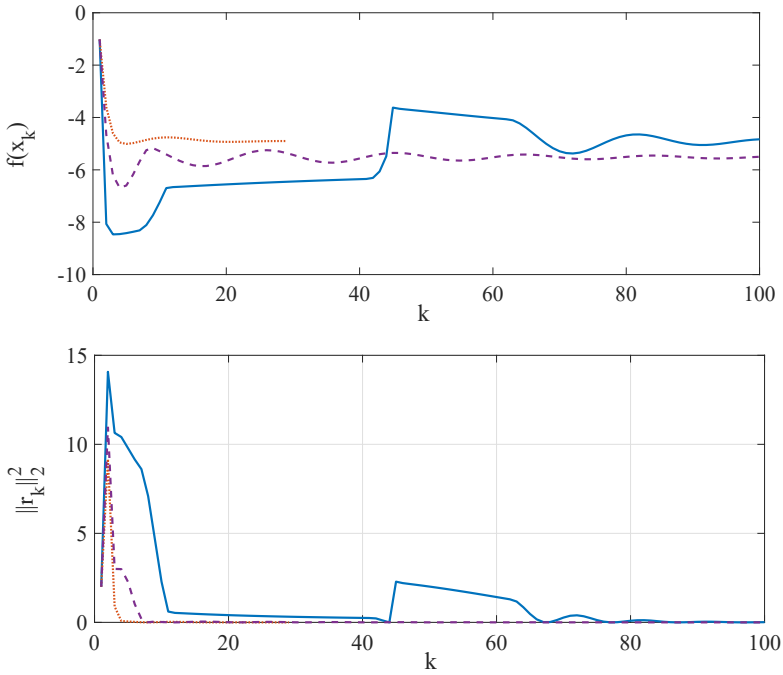

Figure 3.15: An illustrative example of the iterates of ADMM. The solid blue line corresponds to a constant $\rho = 0.4f(x_0)$, the dashed purple line corresponds to $\rho = f(x_0)$, and the dotted red line corresponds to $\rho = 4f(x_0)$. (Top) The objective value of the primal iterates x_k. (Bottom) The primal residual $||r_k||_2^2 = ||Ax_k - b||_2^2$.

the properties and highlights the differences between each of the algorithms from this chapter in more detail. Each of these algorithms is applicable to the battery scheduling problem, and these results are compared in Table 3.22.

Table 3.21: Properties of Mixed-Integer ALADIN, PaDOA, and Distributed Branch & Bound.

	MI-ALADIN	PaDOA	D B&B
Problem Class	MINLP	MICP	MINLP
Fully Distributed	Yes	No	Yes
Global Convergence	No	Yes	Yes
Global Optimality	No	Yes	Yes
Integer Coupling	No	Yes	Yes

Table 3.22: Results of Mixed-Integer ALADIN, PaDOA, and Distributed Branch & Bound for Problem (3.10).

N	K	Bonmin Time (s)	Alg. 5 Time (s)	Alg. 5 Iter.	Alg. 6 Time (s)	Alg. 6 Iter.	Alg. 7 Time (s)	Alg. 7 Iter.
2	2	0.153	0.138	9	0.261	6	26.07	41
12	2	0.333	0.237	17	0.408	8	0.814	1
2	12	0.274	0.230	13	1.306	17	1000	931
54	2	0.761	0.165	14	0.918	10	0.611	1
2	48	1.329	0.602	23	3.735	18	1000	686
12	12	1.834	0.244	17	2.563	5	1.185	1
12	48	1,678	0.471	10	55.54	42	62.32	38
54	12	17.1634	0.406	14	0.911	1	1.789	1
54	48	> 3000	1118	72	> 3000	N/A	1000	171

4 Structure Exploitation and Problem Partitioning

Chapter 3 has given three Algorithms for exploiting separable and partially separable problem structure. Each of these algorithms requires the input problems to already be partitioned and no comment is made on how one ought to partition the problem for use in a distributed optimization context. This chapter seeks to offer first steps towards optimal problem partitioning for the distributed optimization algorithms presented in Section 2.1.2 and Chapter 3. Additionally, a means of time-wise structure exploitation for mixed-integer OCPs is presented in Sections 4.2.2 and 4.2.3. Like the distributed algorithms that have been presented in previous chapters, these methods are meant for large-scale optimization problems. Unlike the distributed algorithms, they do so by exploiting some *a priori* knowledge about the optimal state trajectory rather than problem decomposition.

4.1 Problem Partitioning

Graph partitioning takes different forms throughout the literature and for the sake of consistency and rigour, we shall adopt the following definition:

Definition 7. *Let $G = (V, E, w)$ be a weighted graph with vertex set V, edge set E, and edge weights $w : E \to \mathbb{R}$. Given an integer $k > 1$, a k-partitioning of G is defined as the set*

$$\{G_1(V_1, E_1, w_1), \ldots, G_k(V_k, E_k, w_k)\}$$

such that

- $V_1 \cup \cdots \cup V_k = V$

- $V_i \neq V_j$, $\forall i \neq j$
- $|V_i| \leq (1+\varepsilon)\lceil |V|/k \rceil$, *for some* $\varepsilon > 0$.

If $\mathbb{E}$ *is the set of edges in* $E - \{E_1 \cup \cdots \cup E_k\}$ *then the above partitioning is considered to be optimal*[1] *if* $\sum_{e \in \mathbb{E}} w(e)$ *is minimized.*

The third condition is known as the balance constraint and uses an imbalance parameter ε to enforce each partition to have roughly the same number of vertices.

For $k = 2$, the graph partitioning problem can be formulated into an MIP as

$$\begin{aligned} \min_x \quad & \sum_{e=(i,j)\in E} (x(i) - x(j))^2 \\ \text{s.t. } & x(i) \in \{-1, 1\}, \quad \forall i \in V \\ & \sum_i x(i) = 0, \end{aligned}$$

where the vector x is defined as $x(i) = 1$ if node i is in partition G_1 and $x(i) = -1$ if it is in partition G_2. To obtain balanced partitions, the additional constraint $\sum_i x(i) = 0$ is required.[2] Recall that MIPs are known to be NP-complete, and so too is the graph partitioning problem, where the existence of many local solutions is possible.

Likewise, the related problem of graph clustering shall be defined as:

Definition 8. *Let* $G = (V, E, w)$ *be a weighted graph with vertex set* V*, edge set* E*, and edge weights* $w : E \to \mathbb{R}$*. Given* $k > 1$*, a clustering of* G *is defined as the set*

$$\{G_1(V_1, E_1, w_1), \ldots, G_k(V_k, E_k, w_k)\}$$

such that

- $V_1 \cup \cdots \cup V_k = V$

[1] In a graph-partitioning sense where the objective is to minimize the number of interconnecting edges.

[2] This assumes that the cardinality of V is even. An alternative can be easily formulated if this is not the case.

- $V_i \neq V_j, \forall i \neq j$

The above clustering is considered to be optimal if $\sum_{i=1}^{k} \sum_{e \in E_i} w_i(e)$ *is maximized.*

In graph clustering, it is common to refer to the edge weights w as the *affinity* between pairs of graph nodes. While graph partitioning aims to minimize the number of edges between partitions, graph clustering seeks to maximize the affinity of elements within each cluster. The pairwise affinity between nodes in G is predefined and compiled within an *affinity matrix*. Thus, provided that a proper affinity matrix can be constructed, this method is suitable to problem partitioning. The interested reader can find more information on graph clustering techniques in [BGW03].

In general, there are multiple ways to partition a given problem and the question of how to optimally[3] partition a problem for input to a distributed algorithm is still unanswered. As the partitioned problem is to be input into a distributed optimization algorithm, one must first consider how a given partitioning could affect the algorithmic performance. All of the algorithms described in Chapter 3 follow a similar structure: alternation between a parallel step and a coupling step. The overall runtime R of the algorithm can thus be computed as:

$$R = \sum_{i=1}^{M} \min\{p_1(i), \ldots, p_n(i)\} + s(i)$$

where $s(i)$ is the time spent in the sequential part of the algorithm on the i^{th} iteration, n is the number of parallel components, M is the number of iterations, and $p_j(i)$ is the time spent by the j^{th} parallel component at the i^{th} iteration.

One would expect that for Algorithm 5 the sequential time $s(i)$ would be primarily dictated by the time spent on the QP step in iteration i, while the parallel time $p(i) = \sum_{i=1}^{M} \min\{p_1(i), \ldots, p_n(i)\}$ would be mostly spent on the decoupled MINLP subproblems at iteration i. Theoretically, s should primarily depend on the number of edges that connect partitions, where fewer connections typically results in better performance. Likewise, the size of a partition should be proportional to its runtime. This motivates the need for

[3] Such that runtime or overall iterations until convergence is minimized.

partitioning methods which can minimize the number of interconnecting edges while simultaneously balancing the sizes of each partition. These two criteria perfectly fit the common partitioning problem in graph theory described by Buluç in [BMS+16], however this conjecture is examined for two power systems problems in Section 4.1.1.

Most graph partitioning approaches are hierarchical and iteratively split a graph into two partitions per iteration [NJW02, GHT16a]. Such an approach is called bi-partitioning, and recursive bi-partitioning can create a k-partitioning, but is not a computationally efficient means of doing so [SHH+16].[4] There are other methods which take a more wholistic approach to graph partitioning by directly creating k-partitions rather than relying on recursive bi-partitioning. Spectral clustering and multi-level graph partitioning are the two most common examples of this approach to graph partitioning.

Spectral clustering, as its name suggests, is designed for the related problem of graph clustering, however it can also be used to construct k–partitions of a given undirected graph. In spectral clustering, the partitions are formed via clustering of the eigenvectors of the affinity matrix. As such, if many elements are more similar to each other than the rest of the data set then they will be grouped together. Thus, it is not guaranteed that the sizes of the partitions will be balanced. Implementations of spectral clustering typically use a k-means algorithm to perform the clustering of the eigenvectors of the affinity matrix, where the first k eigenvectors of the affinity matrix are used to generate the initial means of the k-means algorithm. Given, some initial points, the k-means algorithm then aims to, for each of the k clusters, minimize the pairwise squared deviations of each point in the same cluster. The algorithm iteratively alternates between an assignment step, where each element is assigned to the nearest mean, and the update step of the means [Mac67, Llo82]. As the results of the k-means algorithm can change given different initial points, so too can the partitions of the spectral clustering method. The Spectral Clustering algorithm is summarized in Algorithm 10.

Multi-level graph partitioning is another technique which can generate balanced partitions. Such methods consist of three main steps: contraction, initial

[4] However, there is some evidence that it can make more efficient use of memory resources [DGE+02].

Algorithm 10: Spectral Clustering

Input: Affinity matrix A, graph $G(V, E)$, number of clusters k.

1. Define diagonal matrix D where

$$D_{i,i} = \sum_{j=1}^{n} A_{i,j}$$

and construct the matrix

$$P = D^{1/2} A D^{1/2}.$$

2. Construct the matrix V via a concatenation of the eigenvectors corresponding to the k largest eigenvalues of P.
3. Apply a clustering algorithm, such as k-means, to the rows of V and obtain k clusters.
4. Assign bus i to cluster G_ℓ if row i of V was assigned to cluster G_ℓ in Step 3.

Output: $G_1(V_1, E_1), ..., G_k(V_k, E_k)$.

partitioning, and refinement [SS13a, SS13b]. In the first step, the algorithm contracts the input graph to create a smaller representation until it is small enough to be partitioned with a global algorithm. To find edges to contract, the algorithm creates a maximum matching using the global paths algorithm which was presented in [MS07].[5] The matching is then contracted by combining the start and endpoint of every edge in the set, and decreases the size of the input graph. This process is then repeated until the graph is small enough, at which time a global partitioning algorithm is applied. Next, the contraction is then undone step by step, applying local refinement strategies. This involves checking whether moving some vertices that lie at the partition boundary to the neighbouring partition would improve the partition balance or the minimum number of edges cut. The result is a partitioning with evenly sized partitions and a minimal number of edges that are cut in between the partitions. The steps of multi-level graph partitioning are summarized in Algorithm 11

[5] A matching is a set of edges where no two edges in the set have a common endpoint (vertex). More information on the construction of these matchings can be found in Holtgrewe et al. [HSS10].

Algorithm 11: Multi-level Graph Partitioning

Input: Weighted graph $G(V, E, w)$ and number of partitions k.

1. The graph G is transformed into a sequence of smaller graphs $G_1, G_2, \ldots, G_k$ such that $|V_0| > |V_1| > |V_2| > \cdots > |V_k|$.
2. A 2-way partition P_k of the graph $G_k = (V_k, E_k, w_k)$ is computed that partitions V_k into two parts, each containing half the vertices of G.
3. The partition P_k of G_k is projected back to G by going through intermediate partitions $P_{k-1}, P_{k-2}, \ldots, P_1, P_0$.

Output: $G_1(V_1, E_1, w_1), \ldots, G_k(V_k, E_k, w_k)$.

Two commonly used and well-known multi-level graph partitioning algorithms are the "Karlsruhe Fast Flow Partitioner" (KaFFPa) and "METIS." A detailed description of the KaFFPa algorithm is given in [SS11]. Like KaFFPa, METIS is a multi-level partitioning method where the main requirement is an equal number of nodes in each partition and a minimum number of interconnections between the partitions, however it uses a Kernigan-Lin approach in the uncoarsening phase as opposed to the local search method used by KaFFPa [KK99].[6] Overall, the runtime is still comparable to KaFFPa.

4.1.1 Power Grid Partitioning Example

While highly symmetrical problems may have very natural partitionings, this is not necessarily the case for other problems, such as OPF or RPD. Such problems typically represent each irreducible problem element as a node in a graph, with edges representing relationships between problem elements [BGMT13, EMJ+17, MEHF18, GHT16a]. With this representation, one may make use of the vast research available on graph partitioning. Some work on partitioning of the OPF problem has already been done [GHT16a, GHT16b, MKS+20], however it is almost impossible to determine an "optimal" partitioning due to

[6] It is worth noting here that this is not the only viable approach for this step and [Mey12] presents a promising alternative.

the sheer number of possible partitionings.[7] Doing so would require checking each partitioning with the chosen algorithm, and demand a considerable amount of time and computational resources. Instead, one can use one of the previously described graph partitioning strategies to obtain a partitioning with favourable properties. Each of the previously described graph partitioning methods requires an affinity matrix to be given which contains the pair-wise edge weights. In [GHT16a], this is done by first solving it centrally, then evaluating the Hessian of the Lagrangian of Problem (4.1) at the optimal set point. Alternatively, one could use the admittance matrix according to the algorithm described in [NJW02].

In this section, two spectral clustering methods and two multi-level graph partitioning methods are used to partition the IEEE benchmark test cases for input into ALADIN. The first spectral clustering method uses the Y-bus matrix (g and b parameters) as a notion of element-wise affinity, and the second uses the Hessian of the Lagrangian of Problem (4.1) at the optimal operating point. The two multi-level graph partitioning methods are the aforementioned KaFFPa and METIS, and both only use the topology of the grid as the affinity matrix.[8]

For a grid with a node set N, generator set G, and grid parameters g and b, the OPF problem is defined as follows:

$$\min_{\theta,v,p,q} \sum_{i\in G} \alpha_i p_i^2 + \beta_i p_i + \gamma_i \tag{4.1a}$$

$$\text{s.t. } \forall i \in N:$$

$$\underline{p}_i \leq p_i \leq \overline{p}_i, \underline{q}_i \leq q_i \leq \overline{q}_i, \underline{v}_i \leq v_i \leq \overline{v}_i \tag{4.1b}$$

$$v_i \sum_{k\in N_i} v_k (g_{ik} cos(\theta_{ik}) + b_{ik} sin(\theta_{ik})) = p_i - p_{d_i} \tag{4.1c}$$

$$v_i \sum_{k\in N_i} v_k (g_{ik} sin(\theta_{ik}) - b_{ik} cos(\theta_{ik})) = q_i - q_{d_i} \tag{4.1d}$$

[7] The number of partitions of a set of size n is the n^{th} Bell number. For the IEEE 9-bus case this is 21,147.

[8] That is, the affinity matrix is equally weighted.

where for each node $i \in N$, θ_{ik} is the voltage angle between buses i and k, v_i is the voltage magnitude, p_i is the active power injection, and q_i is the reactive power injection. The parameters $\alpha_i \in \mathbb{R}$, $\beta_i \in \mathbb{R}$, and $\gamma_i \in \mathbb{R}$ are the cost coefficients associated with the i^{th} generator. As in Section 3.1.3, the parameters of problem (4.1) are taken from the IEEE test cases. Using the four aforementioned graph partitioning methods, the 2-partitions of the IEEE 14-bus case are shown in Figure 4.1. These partitions are in line with the intuitive partitioning of these grids.

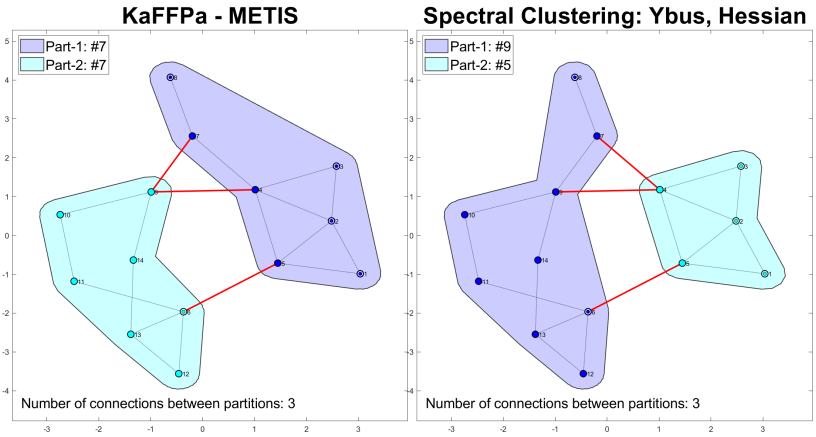

Figure 4.1: Partitions generated for the IEEE 14 bus case.

Partitions are generated for each of the IEEE test cases up to the 300 bus case, and the number of iterations required by ALADIN to converge to a locally optimal solution for each of the partitioned OPF problems are shown in Table 4.1. In all cases, the termination threshold of ALADIN is 10^{-3} and an iteration limit of 500 is set, which is reached in the 300 bus case for several partitionings.

Recall that ALADIN requires certain parameters to be given, which not only can be difficult to choose *a priori*, but also have a large effect on the convergence rate. A number of different combinations of parameters are tested for each of the partitionings and only the best results are shown in 4.2. Interestingly, the best parameters seem to vary from partitioning to partitioning. All parameters

Table 4.1: ALADIN iterations for the partitioned IEEE test cases.

Case	Partitions	KaFFPa	SC Hessian	SC Ybus	METIS
9	2	3	3	3	3
14	2	17	29	29	17
30	2	20	15	15	23
39	2	4	8	8	4
57	2	16	14	20	41
57	3	22	24	29	71
118	2	20	23	23	22
118	3	28	34	40	37
118	4	28	29	25	22
118	5	23	42	44	44
300	3	51	> 500	> 500	102
300	5	97	> 500	> 500	103

are given in the form $(\rho, \mu, \Delta\rho, \Delta\mu)$ where ρ and μ are the initial values of the parameters described in Section 3.1, and where $\Delta\rho$ and $\Delta\mu$ are the factors by which ρ and μ are updated at every iteration. It should be noted that this updating method is a heuristic means of replacing the line search step.

The IEEE 57 bus case is the smallest example examined such that all of the partitioning algorithms provided different partitionings. Shown in Figure 4.2 are the partitionings of each algorithm. Observe that although the partitions generated by KaFFPa and METIS are quite similar, there is nonetheless a large difference in the convergence rate of Algorithm 5. As seen in Figure 4.2, METIS generates many more branches between partitions than KaFFPa, which is likely the reason for its relatively poor performance in the results of Table 4.1. Also of note is the fact that spectral clustering applied to the Y-bus matrix consistently yields almost the exact same partitions as spectral clustering of the Hessian of the Lagrangian at the optimal operating point. This

Table 4.2: Best ALADIN parameters for each partition.

Case	# Part.	KaFFPa	SC Hessian	SC Ybus	METIS
9	2	(500, 500, 1.05, 2)	(500,500, 1.05,2)	(500,500, 1.05,2)	(500,500, 1.05,2)
14	2	(500, 500, 1.05, 2)	(500,500, 1.05,2)	(500,500, 1.05,2)	(500,500, 1.05,2)
30	2	(500, 500, 1.05, 2)	(500,500, 1.05,2)	(500,500, 1.05,2)	(500,1000, 1.05,2)
39	2	(20, 2000, 1.15, 1.15)	(500,2000, 1.15,2)	(500,2000, 1.15,2)	(500,2000, 1.15,2)
57	2	(1000, 2000, 1.05, 2)	(1000,2000, 1.05,2)	(1000,2000, 1.05,2)	(100,100, 1.2,2)
57	3	(100, 2000, 1.2, 2)	(100,100, 1.2,2)	(100,2000, 1.2,2)	(500,2500, 1.05,1.15)
118	2	(100, 100, 1.2, 2)	(100,100, 1.2,2)	(100,100, 1.2,2)	(100,100, 1.2,2)
118	3	(100, 1000, 1.05, 1.5)	(100,1000, 1.05,1.5)	(100,1000, 1.05,1.5)	(100,1000, 1.05,1.5)
118	4	(500, 1000, 1.05, 1.5)	(500,1000, 1.2,1.5)	(500,1000, 1.1,1.5)	(500,1000, 1.1,1.5)
118	5	(500, 2000, 1.1, 1.5)	(500,2000, 1.1,1.5)	(500,2000, 1.1,1.5)	(500,2000, 1.1,1.5)
300	3	(100, 100, 1.1, 2)	N/A	N/A	(100,100, 1.05,1.1)
300	5	(100, 100, 1.1, 2)	N/A	N/A	(500,500, 1.05,2)

seems to imply that the resistances and susceptances of the lines are dominant when considering a partitioning and that very little is gained by the *a priori* knowledge of the optimal operating point. However, as noted in [GHT16a], this may change when line limits are reached.

4.1.2 Reactive Power Dispatch Example

Recall from Section 3.1.3 that RPD is typically performed as a secondary optimization step after an OPF. In practice, any problem partitioning will come along with specific infrastructure requirements, such as communication and data storage constraints. Thus, if RPD were solved in a distributed computing setting,[9] then the partitions of RPD will be fixed by those of OPF.[10] The locations and values of the tap transformers and shunt capacitors for the 14 and 30 bus cases are the same as described in Section 3.1.3, while those of the 57 bus case are listed in Table 4.3.

Table 4.3: Control variables for the 57 bus case and their admissible values (in p.u.).

Variable	Domain
$V_{\mathcal{G}}$	[0.94,1.06]
a	$\{0.90, 0.91s, \ldots, 1.09, 1.10\}$
s_{23}	$\{0, 0.2\}$
s_{27}	$\{0, 0.2\}$
s_{42}	$\{0, 0.2\}$
s_{53}	$\{0, 0.2\}$

Given the partitions used to generate the results of Table 4.1, the convergence rate and runtime of MI-ALADIN applied to the RPD problem (3.11) are

[9] That is, where each partition does not have access to the information of other partitions.

[10] It is worth noting that significant work has been done on grid partitioning for RPD, however all have used continuous relaxation and an optimality condition decomposition algorithm [SGZ+12, YXZ+96] to solve it.

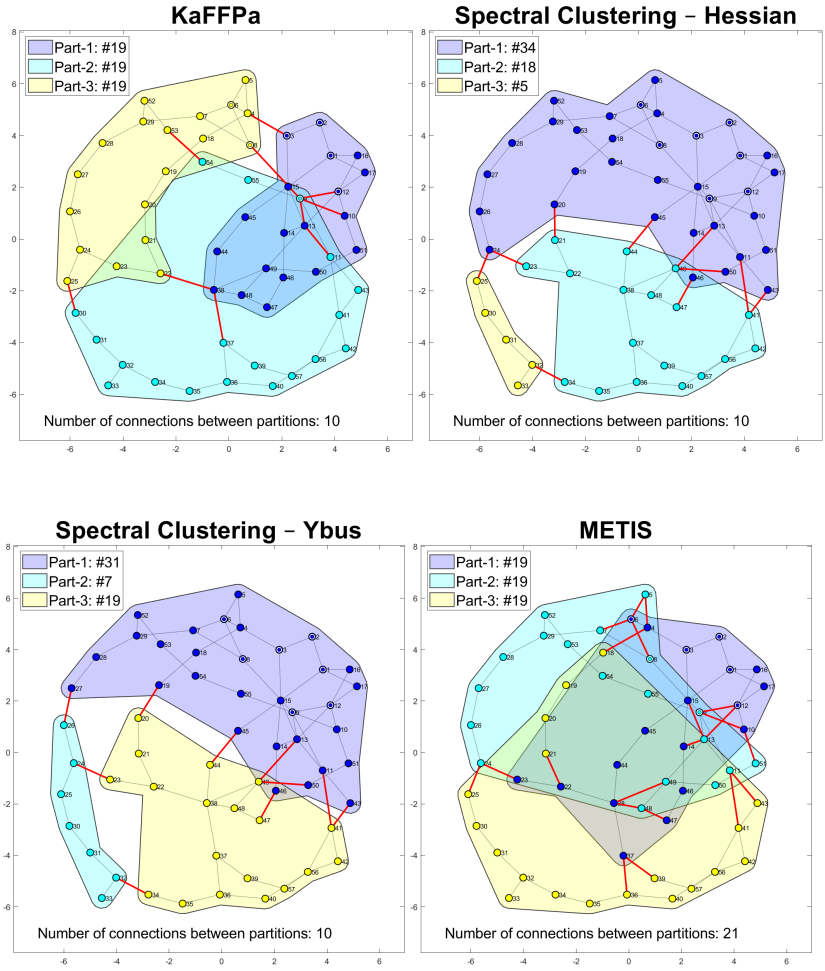

Figure 4.2: Partitions generated for the IEEE 57 bus cases. Thick red lines indicate which branches cross between partitions.

shown in Tables 4.4 and 4.5, and the parameters used to generate them are shown in Table 4.6. Once again, a termination threshold of 10^{-3} and an iteration limit of 500 is set. In Table 4.5, $\overline{p} = \sum_{i=1}^{M} \max\{p_1(i), p_2(i)\}/M$ is the average runtime spent in the parallel steps of Algorithm 5 over all M iterations. Likewise, $\overline{s} = \sum_{i=1}^{M} s(i)/M$ denotes the average runtime of the sequential steps in Algorithm 5.

Table 4.4: Mixed-Integer ALADIN iterations for the partitioned IEEE test cases.

Case-#partitions	KaFFPa	SC Hessian	SC Ybus	METIS
14-2	23	19	19	23
30-2	39	52	48	49
57-2	70	70	65	56
57-3	77	68	62	> 500

Table 4.5: Mixed-Integer ALADIN runtime (ms) for the partitioned IEEE test cases.

	KaFFPa		SC Hessian		SC Ybus		METIS	
Case	$\overline{p}$	$\overline{s}$	$\overline{p}$	$\overline{s}$	$\overline{p}$	$\overline{s}$	$\overline{p}$	$\overline{s}$
14-2	402.1	2.7	381.0	2.9	381.0	2.9	402.1	2.7
30-2	716.5	5.8	541.5	5.5	676.4	5.0	694.8	6.1
57-2	1.9227	0.0225	3.1279	0.0194	2.4660	0.0241	2.0928	0.0207
57-3	1.9010	0.6526	3.0482	0.2308	2.3489	0.0271	N/A	N/A

The results shown in Table 4.4 are a significant improvement upon those obtained in Section 3.1.3 with naive partitionings. This is likely due to each of these partitionings having much fewer cut branches between partitions, which reduces the size of the MINLPs and coupling QP solved by Algorithm 5. The presence of less auxiliary variables can also make convergence easier to achieve due to the presence of fewer coupling constraints. However, as shown

Table 4.6: Best MI-ALADIN parameters for each partition.

Case	KaFFPa	SC Hessian	SC Ybus	METIS
14-2	(20, 10^4, 1.2, 1.5)	(100, 10^4, 1.1, 1.2)	(100, 10^4, 1.1, 1.2)	(20, 10^4, 1.2, 1.5)
30-2	(2, 10^4, 1.3, 1.4)	(2, 10^4, 1.3,1.15)	(2, 10^4, 1.2,1.05)	(2, 10^4, 1.2,1.05)
57-2	(10, 10^4, 1.4, 1.2)	(10, 10^4, 1.4, 1.2)	(10, 10^4, 1.4, 1.2)	(10, 10^4, 1.4, 1.2)
57-3	(10, 10^4, 1.4, 1.2)	(10, 10^4, 1.4, 1.2)	(10, 10^4, 1.4, 1.2)	(10, 10^4, 1.4, 1.2)

in Table 4.4, METIS yielded results comparable to spectral clustering despite having double the number of interconnecting edges between partitions in the 30 bus case. Thus, clearly the number of interconnecting edges is not the only factor that must be considered.

While the number of interconnecting edges seems to have only a mild effect on iterations until convergence, it does have a notable effect on the sequential part of the overall runtime of Algorithm 5 applied to Problem 3.11. Indeed, METIS generates the partitionings with the most interconnecting edges and it also results in the highest sequential runtime. In the 30 bus case, spectral clustering and KaFFPa generate partitionings with half the number of interconnecting edges as METIS and this results in an overall reduction of roughly 20% less runtime in the coupling step of Algorithm 5. Interestingly, balancing the number of buses in each partition does not seem to balance the runtime of each decoupled subproblem. For the 14 bus case, spectral clustering generated the most balanced partitions (in terms of runtime), where each decoupled subproblem requires almost the same amount of time to solve. This is in contrast to the case where the partitioning of KaFFPa/METIS is used where the slowest subproblem is on average 73% slower than the fastest. What these

results indicate is the difficulty involved in predicting how much computing power will be required to solve a given MIP.

Remark 7. *While only the parameters which yielded the best results are shown in Tables 4.2 and 4.6, it was observed that KaFFPa's and METIS' partitions were the most robust to parameter changes. That is, different parameters still result in convergence within the limit – albeit slower than what is reported in Tables 4.1 and 4.4. As shown for the 300 bus case in Table 4.1, no parameter combination was found for either spectral clustering method which resulted in convergence within 500 iterations.*

4.2 Turnpikes in Mixed-Integer Optimal Control

Thus far, the focus has been on the use of spatial decomposition in mixed-integer optimization. In contrast, this section approaches structure exploitation from a different perspective by considering steady-state features in mixed-integer optimal control and presents two methods for taking advantage of this phenomenon [FM20].

As noted in Section 3.1.2, the battery scheduling problem has a partially separable problem structure that can be exploited through distributed optimization. It is tempting to try something similar for other discrete time optimal control problems, since for such problems it generally holds that each time step is coupled only with the one before, and the one after it. While decomposition along these lines can certainly be done, this section will show a more effective method given that certain properties hold.

An OCP is said to exhibit a so-called "turnpike" property in its optimal state trajectories if the time spent outside of any epsilon-neighborhood of the optimal steady state is bounded independent of the horizon length. The notion dates back to the 1950's [Dor58], although early observations of the phenomenon can be traced even further to John von Neumann in the 1930s [vN38].[11] Since then, this property has been well studied [FKJB17, MP01, McK76], and occurs in a variety of contexts, such as economic model predictive control [Grü13]

[11] The same phenomon also appears under different names in the literature: *hyper-sensitive optimal control problems* and *dichtomy in optimal control* [AK87, WK72].

and production planning [TS80, KKTK75]. Regardless of whether a turnpike is present or not, OCPs are generally not solvable analytically and require numerical methods. There are many methods and techniques which have been developed, some of which are listed here: [LSKVI10, Sag05, FFCM15]. However, these methods generally do not apply to MIOCPs. The focus of this section is to provide some algorithms which can be used to efficiently solve MIOCPs with turnpikes.

Recall from Section 2.2.4 the form of an MIOCP without terminal penalty:

$$\min_{x,u,z} \sum_{t=0}^{T-1} f(x(t), u(t), z(t)) \tag{4.2a}$$

$$\begin{aligned}
&\text{subject to } \forall t \in \{0, \dots, T-1\}, \\
&\quad x(t+1) = h(x(t), u(t), z(t)), \quad x(0) = x_0, && \text{(4.2b)} \\
&\quad x(t) \in \mathcal{X} \subseteq \mathbb{R}^n, \quad u(t) \in \mathcal{U} \subseteq \mathbb{R}^\ell, && \text{(4.2c)} \\
&\quad z(t) \in \mathcal{Z} \subseteq \mathbb{Z}^m. && \text{(4.2d)}
\end{aligned}$$

where the stage cost $f : \mathbb{R}^n \times \mathbb{R}^\ell \times \mathbb{Z}^m \to \mathbb{R}$ and the dynamics $h : \mathbb{R}^n \times \mathbb{R}^\ell \times \mathbb{Z}^m \to \mathbb{R}^n$ are assumed to be continuous in x and u. Let $\mathcal{X}_0$ denote the set of admissible initial states of x_0. Optimal solutions, provided they exist, are written as $v^\star(\cdot) = (x^\star(\cdot), u^\star(\cdot), z^\star(\cdot))$.[12]

As an illustrative example of a simple problem in the form of (4.2) consider the following from [FGM18, Grü13]:

$$\begin{aligned}
&\min_{x,u,z} \sum_{t=1}^{T} u(t)^2 + (1/2) z(t)^2 \\
&\quad \text{s.t. } \forall t \in \{1, \dots, T\}, \\
&\quad x(t+1) = 2x(t) + u(t) + z(t) - 1, \\
&\quad (x(t), u(t), z(t)) \in [-2, 2] \times [-3, 3] \times \{-1, 0, 1\}.
\end{aligned} \tag{4.3}$$

[12] The optimal solution $v^\star$ depends on the initial state x_0, however this omitted whenever no confusion can arise.

For $x(0) = -2$, the optimal state and input trajectories are as shown in Figure 4.3.

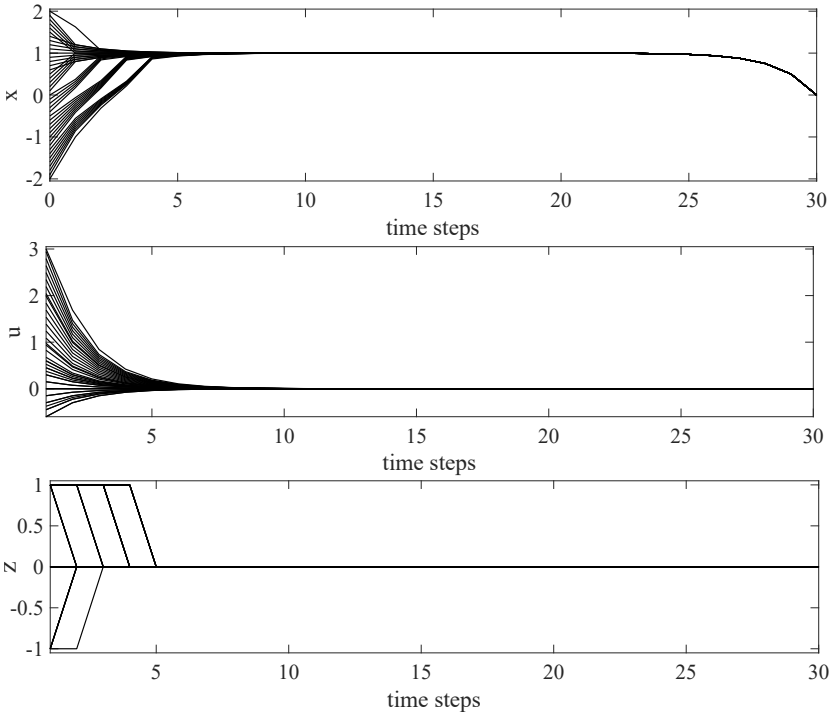

Figure 4.3: Example plot of the optimal solutions for Problem (4.3) with $T = 30$ and a variety of initial states.

Note that x is in a steady state for a large part of the time horizon of Problem (4.3). This is called a turnpike and the formal definition of which for MIOCPs is given in Definition 9 in Section 4.2.1.

4.2.1 Sufficient Turnpike Conditions

The existence of a turnpike in MIOCPs is not guaranteed, and certain conditions must be satisfied in order to allow for such a steady-state optimal trajectory to arise. The definition of the turnpike property is given in Definition 9.

Definition 9 (Mixed-integer turnpike property)**.** *MIOCP* (4.2) *is said to have an input-state turnpike at* $\bar{v} \in \mathcal{X} \times \mathcal{U} \times \mathcal{Z}$ *if there exist* $C \in \mathbb{R}$ *and* $\alpha \in \mathcal{K}_\infty$ [13] *such that, for all* $x_0 \in \mathcal{X}_0$ *and all* $T \in \mathbb{N}$*,*

$$|\mathbb{Q}_\varepsilon| \geq T - \frac{C}{\alpha(\varepsilon)}$$

holds for every $\varepsilon > 0$*, where*

$$\mathbb{Q}_\varepsilon := \{t \in \{0, \dots, T-1\} \mid ||(v^\star(t; x_0)) - \bar{v}|| \leq \varepsilon\}. \tag{4.4}$$

Definition 9 essentially states that the amount of time an optimal tuple spends inside of an ε-ball centred at $(\bar{x}, \bar{u}, \bar{v})$ is at least $T - \frac{C}{\alpha(\varepsilon)}$. Alternatively, one could interpret Definition 9 as stating that the amount of time spend outside of the ε-ball is bounded independently of T by $\frac{C}{\alpha(\varepsilon)}$.

However, to provide sufficient conditions to guarantee the existence of a turnpike we must first recall a notion of dissipativity for MIOCPs as it was introduced in [MSP+20]:

Definition 10 (Strict dissipativity with respect to $\bar{z}$)**.**
A system is said to be dissipative with respect to a steady-state tuple $\bar{v} \in \mathcal{X} \times \mathcal{U} \times \mathcal{Z}$*, if there exists a function* [14] $\lambda : \mathcal{X} \to \mathbb{R}^+$ *such that for all* $v \in \mathcal{X} \times \mathcal{U} \times \mathcal{Z}$

$$\lambda(h(v)) - \lambda(x) \leq f(v) - f(\bar{v}). \tag{4.5a}$$

If, additionally, there exists $\alpha \in \mathcal{K}_\infty$ *such that*

$$\lambda(h(v)) - \lambda(x) \leq -\alpha(\|v - \bar{v}\|) + f(v) - f(\bar{v}), \tag{4.5b}$$

[13] $\mathcal{K}_\infty$ refers to the set of continuous functions $\mathbb{R}_0^+ \to \mathbb{R}_0^+$ which are 0 at 0, strictly monotonously increasing, and satisfy $\lim_{s \to \infty} \alpha(s) = \infty$.

[14] λ is also known as a *storage function*.

then the system is said to be strictly dissipative with respect to $\bar{v} = \begin{bmatrix} \bar{x}, & \bar{u}, & \bar{z} \end{bmatrix}^\top$.

Moreover, if the above dissipativity notions holds solely along optimal solutions of (4.2)*, then the MIOCP* (4.2) *is said to be strictly dissipative with respect to* $\bar{v}$.

It is straightforward to see that dissipativity w.r.t. to $\bar{v}$ implies that $\bar{v} = \bar{v}^\star$, i.e. a system can only be dissipative with respect to steady state $\bar{v}$. The steady state of Problem (4.2), provided it exists, can be found by solving:

$$\min_{\bar{x},\bar{u},\bar{z}} \; f(\bar{x},\bar{u},\bar{z}) \tag{4.6a}$$

subject to

$$\bar{x} = h(\bar{x},\bar{u},\bar{z}), \tag{4.6b}$$

$$\bar{x} \in \mathcal{X} \subseteq \mathbb{R}^n, \quad \bar{u} \in \mathcal{U} \subseteq \mathbb{R}^\ell \tag{4.6c}$$

$$\bar{z} \in \mathcal{Z} \subseteq \mathbb{Z}^m. \tag{4.6d}$$

However, dissipativity alone is not sufficient to guarantee the existence of a turnpike. An MIOCP (4.2) also requires the steady state to be *exponentially reachable*.

Assumption 1 (Exponential reachability of $\bar{x}^\star$)**.** *For all* $x_0 \in \mathcal{X}_0$*, there exist infinite-horizon admissible inputs* $u_\infty, z_\infty : \mathbb{N} \cup \infty \to \mathcal{U} \times \mathcal{Z}$ *and constants* $c \geq 0, \rho \in [0,1)$ *such that* $\forall k \geq 0$*:*

$$\|x(k; x_0, u_\infty(\cdot), z_\infty(\cdot)) - \bar{x}^\star\| \leq c\rho^k,$$

i.e. the optimal steady state $\bar{x}^\star$ *is exponentially reachable.*

Given that the optimal steady state $\bar{x}^\star$ is exponentially reachable and the MIOCP (4.2) is strictly dissipative with respect to $\bar{v}^\star$, then a turnpike is guaranteed to exist at $\bar{v}^\star = \bar{v}$. This proposition is put formally in Proposition 1. The proof of Proposition 1 is very similar to its continuous discrete-time counterpart found in [FGM18, Grü13] and follows a similar pattern.

Proposition 1 (Turnpikes in MIOCPs)**.** *Let Assumption 1 hold and suppose that the MIOCP* (4.2) *is strictly dissipative with respect to* $\bar{v}^\star$*. Then the MIOCP* (4.2) *has an input-state turnpike at* $\bar{v}^\star = \bar{v}$.

Proof. Without loss of generality suppose that $f(\bar{v}) = 0$. Evaluating the strict dissipation inequality (4.5b) along an optimal triplet $v^\star(\cdot; x_0)$ gives

$$\lambda(x^\star(T; x_0)) - \lambda(x_0) \leq \sum_{t=0}^{T-1} -\alpha(\|v^\star(t; x_0) - \bar{v}\|) + f(v^\star(t; x_0)).$$

Let $F_T : \mathcal{X} \to \mathbb{R}$ denote the optimal value function[15] of (4.2). Boundedness of the storage function λ implies

$$F_T(x_0) \geq -2\hat{\lambda} + \sum_{t=0}^{T-1} \alpha(\|v^\star(t; x_0) - \bar{v}^\star\|),$$

whereby $\hat{\lambda} := \sup_{x \in \mathcal{X}} \lambda(x)$. Moreover, Assumption 1 implies that for all $x_0 \in \mathcal{X}_0$, we have the following upper bound on $F_T(x_0)$:

$$L_h c \rho + \frac{Lc}{1-\rho} \geq F_T(x_0),$$

where $L \in \mathbb{R}^+$ and L_h are Lipschitz constant of the stage cost f on $\mathcal{X} \times \mathcal{U} \times \mathcal{Z}$.

Observe that $T - |\mathbb{Q}_\varepsilon|$ is the time that the optimal triplet spends $v^\star(\cdot; x_0)$ spends outside of an ε-ball centered at $\bar{v}$. Hence,

$$\sum_{t=0}^{T-1} \alpha(\|v^\star(t; x_0) - \bar{v}^\star\|) \geq (T - |\mathbb{Q}_\varepsilon|) \cdot \alpha(\varepsilon).$$

The last three inequalities imply $|\mathbb{Q}_\varepsilon| \geq T - \dfrac{L_h c \rho + Lc(1-\rho)^{-1} + 2\hat{\lambda}}{\alpha(\varepsilon)}$. □

One interesting lemma specific to MIOCPS that follows from Proposition 1 is the following:

Lemma 2 (Integer controls often exactly at turnpike)**.** *For all $x_0 \in \mathcal{X}_0$, let MIOCP* (4.2) *have an input-state turnpike at $\bar{v}^\star = \bar{v}$ in the sense of Definition 9. Then, for all $\varepsilon \in (0, 1)$ and sufficiently large $T \in \mathbb{N}$, the optimal integer controls $z^\star(\cdot; x_0)$ satisfy $z^\star(t; x_0) \equiv \bar{z}^\star$ for all $t \in \mathbb{Q}_\varepsilon$.*

[15] That is, the solution to (4.2) given an initial state $x_0 \in \mathcal{X}_0$.

Proof. Recall that $\|v^\star(k;x_0) - \bar{v}^\star\| \geq \|z^\star(k;x_0) - \bar{z}^\star\|$, and note the fact that for the integer controls $\|z^\star(k;x_0) - \bar{z}^\star\| < 1 \Leftrightarrow \|z^\star(k;x_0) - \bar{z}^\star\| = 0$. Combining both and taking the definition of $\mathbb{Q}_\varepsilon$ in (4.4) into account yields the assertion.

□

Lemma 2 will help in Section 4.2.2 to efficiently exploit the existence of turnpikes in MIOCPs.

In practice, one may not be able to guarantee the existence of a turnpike in the optimal solution trajectory *a priori*. To this end, Theorem 5 provides a means of numerically verifying Definition 10.

Theorem 5 (Dissipativity of linear-quadratic MIOCPs)**.** *Consider MIOCP* (4.2) *and let the dynamics and the stage cost be given by*

$$\begin{aligned} h(x,u,z) &= Ax + B_1 u + B_2 z, \\ f(x,u,z) &= x^\top Q x + \begin{bmatrix} u \\ z \end{bmatrix}^\top R \begin{bmatrix} u \\ z \end{bmatrix} + q^\top x + r^\top \begin{bmatrix} u \\ z \end{bmatrix}, \end{aligned}$$

with potentially indefinite matrices $Q \in \mathbb{R}^n \times \mathbb{R}^n$ *and* $R \in \mathbb{R}^{(\ell+m)} \times \mathbb{R}^{(\ell+m)}$.

If there exists matrix $P \in \mathbb{R}^{n\times n}$ *such that*

$$Q + P - A^\top P A \succ 0, \tag{4.7}$$

then MIOCP (4.2) *is strictly dissipative, and there exists* $p \in \mathbb{R}^n$ *such that* $\lambda(x) = x^\top P x + p^\top x$ *is a corresponding storage function.*

Proof. The proof combines the available storage characterization of dissipativity with recent results from [GG18].

The available storage is given by

$$\lambda^a_{\mathcal{Z}}(x) = \sup_{v(\cdot),N} \sum_{k=0}^{N} -\alpha(\|(v_k) - \bar{v}\|) + f(v_k) + f(\bar{v})$$

subject to

$$x(k+1) = Ax(k) + \begin{bmatrix} B_1 & B_2 \end{bmatrix} \begin{bmatrix} u(k) \\ z(k) \end{bmatrix}, \; x(0) = x_0.$$
$$x(k) \in \mathcal{X}, \; u(k) \in \mathcal{U}, \; z(k) \in \mathcal{Z},$$

whereby the subscript $\mathcal{Z}$ in $\lambda_{\mathcal{Z}}$ refers to the constraints on the discrete control input z. Indeed the MIOCP (4.2) is dissipative on $\mathcal{X}_0$ if and only if $\lambda^a_{\mathcal{Z}}(x) < \infty$ for all $x_0 \in \mathcal{X}_0$ [Wil72].

Step 1: Let $\mathrm{conv}(\mathcal{Z})$ be the convex hull of $\mathcal{Z}$. By definition 2, $\mathrm{conv}(\mathcal{Z})$ is compact if $\mathcal{Z}$ is compact and $\mathcal{Z} \subset \mathrm{conv}(\mathcal{Z})$. This implies that

$$\lambda^a_{\mathcal{Z}}(x) \leq \lambda^a_{\mathrm{conv}(\mathcal{Z})}(x),$$

i.e. dissipativity of the continuously relaxed OCP certifies dissipativity of the MIOCP.

Step 2: For the continuously relaxed problem it has been shown in [GG18, Lem. 4.1] that (4.7) is a necessary and sufficient condition for strict dissipativity with quadratic storage function on bounded subsets of $\mathbb{R}^n$. Combining both facts yields the assertion. □

4.2.2 Sequential Move-Blocking

The key property of a turnpike is that the state variable(s) remain nearly constant over some continuous portion of the time horizon. As shown in (4.6), the decision variables that correspond with this steady state can be computed *a priori* via an analytical expression. This turnpike may have a

leaving arc[16]Determining where the turnpike begins however, is much more difficult. As such, the main goal of the sequential moveblocking algorithm presented in this section is turnpike detection. It does so by first fixing the integer variables z to their steady state optimal values $\overline{z}$ over some time interval $[k, \dots, k_2]$ and then "unfixing" the integer(s) of time step k. Each step relies on the well-known "move blocking"[17] technique that has seen some popularity in MPC. For more background on move blocking and its use in MPC see [CGKM07] and [SM15]. Sequential moveblocking performs this technique until the objective value of the current iteration is equal to the objective value of the last iteration, at which point the algorithm will terminate. Otherwise, the next integer(s) are unfixed and the algorithm continues. This process is illustrated in Figure 4.4, and formulated more precisely in Algorithm 12.

Algorithm 12: Sequential Move-blocking

Input: Guess for the start and end of the turnpike k_1 and k_2. Termination parameter K, steady state integer solution $\overline{v} = (\overline{x}, \overline{u}, \overline{z}) \in \mathcal{X} \times \mathcal{U} \times \mathcal{Z}$, and numerical tolerance $\varepsilon > 0$.

Initialization: Set $F_{k-1}(z_0) = \infty$.

While $k \leq k_2$**:**

1. Obtain objective value F_k and solution $v_k = (x_k, u_k, z_k) \in \mathcal{X} \times \mathcal{U} \times \mathcal{Z}$:

$$F_k = \min_{v} \sum_{t=1}^{T} f(v(t)) \quad \text{s.t.} \begin{cases} x(t+1) = h(v(t)), \\ x(t) \in \mathcal{X}, u(t) \in \mathcal{U}, z(t) \in \mathcal{Z} \\ \{z(k), \dots, z(k_2)\} = \overline{z} \end{cases}$$

2. If $k > K + k_1$ and $\sum_{i=0}^{K-1} |F_{k-i-1} - F_{k-i}| \leq \varepsilon$ then terminate. Otherwise, $k \leftarrow k+1$ and go to Step 1.

Output: F_k and v_k.

[16] A leaving arc is a state trajectory that leaves the neighbourhood of a turnpike at the end of the time horizon. An example of this would be where profit is maximized over a finite horizon by following some strategy and then selling off all assets at the end of the time horizon.

[17] A technique wherein certain permissible control actions are restricted, allowing for potentially faster solution on the smaller feasible subset.

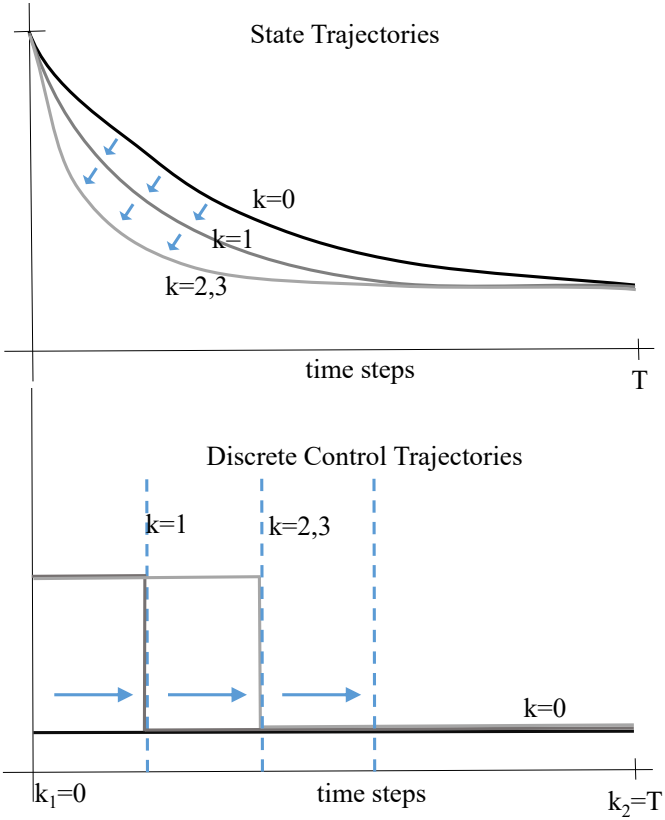

Figure 4.4: An example plot of the state and input trajectories for Algorithm 12. On the left are the state trajectories at each iteration, and on the right are the input trajectories for z.

The convergence properties of Algorithm 12 are given in Section 4.2.2 and its numerical performance is evaluated using some benchmark examples in Section 4.2.4.

Convergence Properties

As long as the MIOCPs in Step 1 of Algorithm 12 terminate in a finite amount of time, Algorithm 12 is guaranteed to converge either due to turnpike detection or will reach the limit k_2, which can be at most T. The optimality of the solution upon convergence is proven in Theorem 6.

Theorem 6. *Given that* (4.2) *has a turnpike from timestep n_1 to n_2, if* $0 < K < \min\{k_2 - k_1, n_2 - k_1, k_2 - n_1, n_2 - n_1 + 1\}$ *then Algorithm 12 applied to* (4.2) *will terminate with an optimal solution.*

Proof. It follows from Lemma 2 that integer controls are exact at the turnpike and thus $F_{n_1} = \cdots = F_{n_2} = f(v^\star)$. Consider the four cases:

- $k_1 \leq n_1$ and $k_2 \leq n_2$.
- $k_1 \geq n_1$ and $k_2 \leq n_2$.
- $k_1 \leq n_1$ and $k_2 \geq n_2$.
- $k_1 \geq n_1$ and $k_2 \geq n_2$.

The values of F_{n_1-1} and F_{n_2+1} are uncertain and thus if $K \geq k_2 - n_1$ then the termination criteria

$$\sum_{i=0}^{K-1} |F_{k-i-1} - F_{k-i}| \leq \varepsilon$$

may not be satisfied for any iteration k. Likewise for the cases $K \geq k_2 - k_1$, $K \geq n_2 - n_1$, and $K \geq n_2 - k_1$. Thus, if $0 < K < \min\{k_2 - k_1, n_2 - k_1, k_2 - n_1, n_2 - n_1 + 1\}$ then Algorithm 12 applied to (4.2) will terminate with with $F_{\max\{n_1,k_1\}+K} = f(v^\star)$. □

Theorem 6 essentially states that if the turnpike is at least as long as the time preceding it, then Algorithm 12 is guaranteed to converge to an optimal solution, given that K is appropriately chosen. In practice, $K = 1$ often works well and can lead to much faster convergence. However, if the optimal state trajectory reaches the turnpike more slowly than is feasible (perhaps due to time-varying costs) then Algorithm 12 would quickly return a suboptimal

solution that satisfies $\sum_{i=0}^{K-1} |F_{k-i-1} - F_{k-i}| \leq \varepsilon$. In this case, a larger choice of K would be required.

4.2.3 Node Weighting

The method proposed in Section 4.2.2 seeks to solve (4.2) via a sequence of smaller MIPs. Alternatively, one could instead attempt to solve the full, original MIP but in a smarter way by reducing the number of nodes which must be searched in the decision tree. Specifically, this is done by exploiting *a priori* knowledge of solution properties which go beyond warm starting. Rather, a prioritization hierarchy of the decision tree nodes is constructed which can save time exploring dead-ends, in favour of more fruitful branches.

For turnpikes, this prioritization comes in the form of weights $\mathcal{W}_0$ for each node associated with the steady state optimal integer decisions at the turnpike. These come in the form of full or partial guesses $\mathbf{Z} \in \mathbb{R}^{m \times N}$ of the sequence of optimal integer decisions $z^\star(\cdot; x_0)$ for Problem (4.2). The vector $\mathbf{Z}(k)$ denotes the k-th column of $\mathbf{Z}$, which corresponds to the integer decisions for the k^{th} time step. Lemma 2 can be leveraged in the construction of $\mathbf{Z}$ at the turnpike in the middle part of the horizon, as these values must be exactly on the turnpike values $\overline{z}$.

The node weighting method itself in outlined in Algorithm 13. Note its resemblance to the standard B&B algorithm (Algorithm 3) described in Section 2.2.1. However one key difference is the inclusion of the integer guesses $\mathcal{Z}_0 = \{\mathbf{Z}_{0,1}, \ldots, \mathbf{Z}_{0,M}\}$ in Algorithm 13. Each iteration of Algorithm 13 requires the selection of a node $\tilde{n}$ from the candidate node set $\mathcal{S}$. Each node $n \in S$ is associated with a vector $\mathbf{Z}_n$ which corresponds to the integer decisions that are fixed and relaxed in the subproblem solved at node n. The default branching rules are encoded by the weight function w, which is modified by the values of $\mathcal{W}_0$ and the distance operator $d_{\mathcal{Z}}(\mathbf{Z}_{0,i}, \mathbf{Z}_n) = |\{\tilde{Z} \subset \mathbf{Z}_{0,i} | \tilde{Z} \subset \mathcal{Z}, \tilde{Z} \subset \mathbf{Z}_n\}|$ from the vector $\mathbf{Z}_{0,i}$ to $\mathbf{Z}_n$.[18] The weighting of the node in relation to the search strategy and in relation to the initial guesses provides the full weight associated with choosing node n. After the node with the highest weighting

[18] This notion of distance is similar to the Hamming distance in that what is counted is the number of feasible elements of $\mathbf{Z}_{0,i}$ that are also in $\mathbf{Z}_n$.

has been selected,[19] the solution of the following NLP given partial or full guess(es) $\mathbf{Z}$ of the integer variables z:

$$\text{NLP}(\mathbf{Z}) := \quad \text{MIOCP (4.2) with integrality constraint (4.2d) swapped for} \quad \begin{cases} z(k) = \mathbf{Z}(k) & \text{if } \mathbf{Z}(k) \in \mathbb{Z} \\ z(k) \in \text{conv}(\mathcal{Z}) & \text{else} \end{cases} \tag{4.8}$$

The condition $\mathbf{Z}(k) \notin \mathbb{Z}$ implicitly encodes the relaxation $\mathbf{V}(k) \in \text{conv}(\mathcal{V})$. Solving (4.8) provides the optimal performance bound $J^\star(\mathbf{Z})$ and subproblem solution triplet $v^\star(\mathbf{Z})$.

Algorithm 13: Branch and Bound with Node Weighting

Input: Guesses $\mathcal{Z}_0 = \{\mathbf{Z}_{0,1}, \ldots, \mathbf{Z}_{0,M}\}$ and corresponding weights $\mathcal{W}_0 = \{w_{0,1}, \ldots, w_{0,M}\}$. Termination tolerance $\epsilon > 0$. Default search strategy (depth-first, breadth-first, ...) and corresponding weights $\mathcal{W}$.
Initialization: Set $U = \infty$, $L = -\infty$, $\mathcal{T} = \emptyset$. Re-index nodes $\mathcal{N}$ according to weights $\mathcal{W}_0$. Candidate node set $\mathcal{S} = \{0\}$.
While $\mathcal{S} \neq \emptyset$**:**

1. $\forall n \in \mathcal{S}$,
   $\tilde{w}(n) = w(n) + \sum_{i=1}^{M} w_{0,i} \left(\mathrm{d}_{\mathcal{Z}}(\mathbf{Z}_{0,i}, \mathbf{Z}_n)\right)$
2. $\tilde{n} = \underset{n \in \mathcal{S}}{\arg\max}\, \tilde{w}(n)$ and $\mathcal{S} \leftarrow \mathcal{S} \setminus \{\tilde{n}\}$
3. **Solve** NLP($\mathbf{Z}_{\tilde{n}}$) for $J^\star(\mathbf{Z}_{\tilde{n}})$ and $v^\star(\mathbf{Z}_{\tilde{n}})$ and $\mathcal{T} \leftarrow \mathcal{T} \cup \tilde{n}$
4. **If** $v^\star(\mathbf{Z}_{\tilde{n}})$ is feasible in MIOCP (4.2) and $J^\star(\mathbf{Z}_{\tilde{n}}) < U$, then $U \leftarrow J^\star(\mathbf{Z}_{\tilde{n}})$ and proceed to Step 5.
   If $J^\star(\mathbf{Z}_{\tilde{n}}) > U$ proceed to Step 1.
   Else add the child nodes of $\tilde{n}$ to $\mathcal{S}$ and proceed to Step 5.
5. $L \leftarrow \underset{n \in \mathcal{P}(\mathcal{S})}{\min} \{J^\star(\mathbf{Z}_n)\}$

   If $U - L \leq \epsilon$ terminate.
   Else proceed to Step 1.

Output: $J^\star(\mathbf{Z}_{\tilde{n}}) > U$ and $v^\star(\mathbf{Z}_{\tilde{n}})$.

[19] Note that this may require some tie-breaking process to be undertaken.

As Algorithm 13 amounts to a standard B&B method with a special branching rule and ordering of the decision tree, no optimality guarantees are lost. Thus, if a B&B method without node weighting will return a globally optimal solution, then so would Algorithm 13. If the weighting is done properly, then fewer iterations will be necessary, however as seen in Section 4.2.4, it is not always easy to find a proper weighting.

An example of Algorithm 13 for a small toy problem with six integer variables is shown in Figures 4.5 and 4.6. Here, the nodes corresponding to certain values of four integer variables are prioritized. This leads to a good upper bound being found quite quickly, which then prunes several nodes and allows the B&B algorithm to converge in fewer iterations.

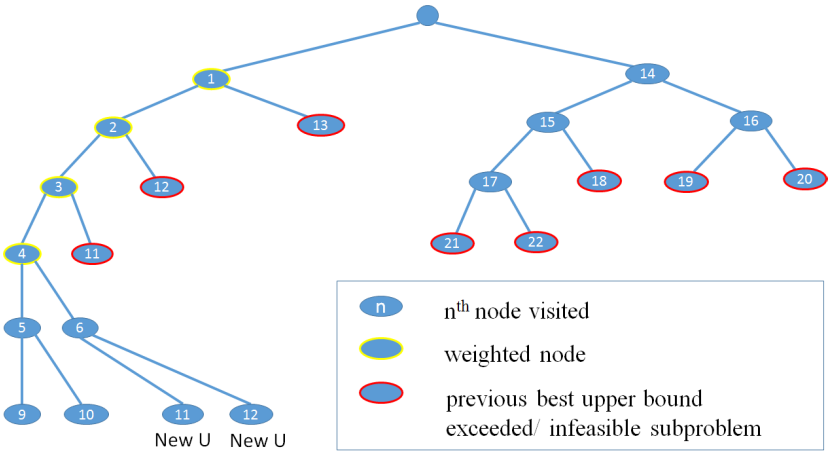

Figure 4.5: An example of the order in which the nodes of a B&B tree are explored when a good weighting strategy is used.

4.2.4 Case Studies

To test the performance of algorithms described in Sections 4.2.2 and 4.2.3, several benchmark problems are used. All such MIOCPs are discretized and solved using single shooting. The solutions to the resulting optimization problems are obtained using a custom implementation of B&B (Algorithm 3).

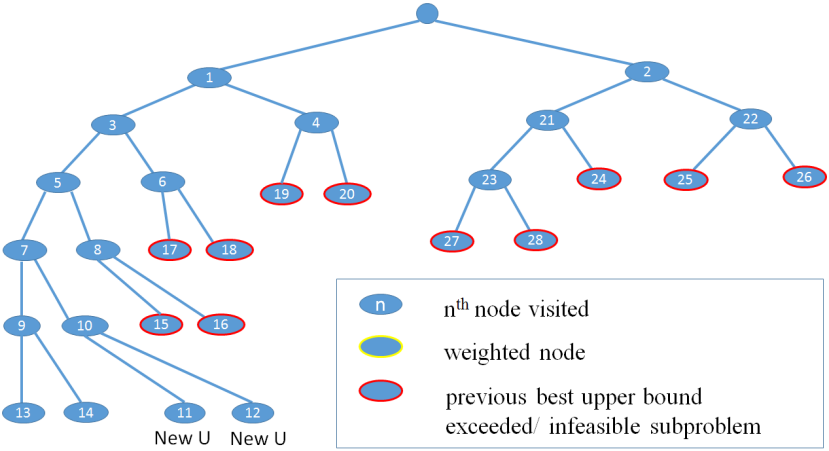

Figure 4.6: An example of the order in which the nodes of a B&B tree are explored when no weighting strategy is used.

The custom B&B algorithm seeks to improve the lower bound as quickly as possible in order to converge by proving optimality.

Example 1

We consider an example with the dynamics of [BM99]:

$$x(t+1) = \begin{cases} 0.8x(t) + u(t), \text{if } x(t) \geq 0, \\ -0.8x(t) + u(t), \text{if } x(t) < 0, \end{cases}$$

with $m \leq x(t) \leq M$. This can be reformulated into a mixed-integer system of equations through the introduction of a continuous variable y and and integer variable z:

$$\begin{aligned} 2z(t)m \leq y(t) + x(t) &\leq 2z(t)M, \\ 2(z(t)-1)M \leq y(t) - x(t) &\leq 2(z(t)-1)m, \\ x(t+1) &= 0.8y(t) + u(t), \\ (1-z(t))m \leq x(t) &\leq z(t)M, \\ \underline{u} \leq u(t) &\leq \overline{u}, z(t) \in \{0,1\}. \end{aligned} \tag{4.9}$$

It is easy to verify that if $z(t) = 0$ then $y(t) = -x(t)$ and $x(t) < 0$, and if $z(t) = 1$ then $y(t) = x(t)$ and $x(t) \geq 0$. Imposing an objective function on the dynamics of (4.9) leads to the following MIOCP:

$$\begin{aligned} \min_{x,y,u,z} \sum_{t=1}^{T} & c_0 u(t)^2 + c_1 x(t)^2 + c_2 x(t) \\ \text{s.t. } & \text{the dynamics given in (4.9)}, \\ & x(0) = x_0. \end{aligned} \tag{4.10}$$

As input Problem (4.10), Algorithm 12 uses $k_1 \in \{1, \ldots, T/2\}$, $k_2 = T$, and $K = 1$. The results for each choice of k_1 are aggregated in the results shown in Table 4.8.

As input for Problem (4.10), Algorithm 13 is provided a collection of complete solution vectors and weights, as shown in Table 4.7. These vectors operate under the assumption that before and after the turnpike $z^\star = 1$ and on the turnpike $z^\star = 0$. Thus, each vector corresponds to a specific guess as to where the turnpike begins and ends.

Table 4.8 compares the performance of Algorithms 3, 12, 13, and standard move blocking. The displayed values aggregate the results from each initial state $x_0 = \{-1, -0.9, \ldots, 0.9, 1\}$ for $T = 10, 20$, and 40. All results presented use $c_0 = 0.5$, $c_1 = 1$, $c_2 = 0.5$, $m = -1$, $M = 1$, $\underline{u} = 0.5$, and $\overline{u} = 0.5$. Shown in Figure 4.7 is an example plot for the case with $T = 20$ and $x_0 = 1$. Note that one can clearly spot the turnpike at $\bar{z}^\star = \begin{bmatrix} -\frac{1}{9} & \frac{1}{9} & -0.2 & 0 \end{bmatrix}$.

Table 4.7: Guesses $\mathcal{Z}_0$ and weights $\mathcal{W}_0$ for Example (4.10).

$\mathbf{Z}_{0,i}$	$w_{0,i}$	$\mathbf{Z}_{0,i}$	$w_{0,i}$
[1,1,0,. . . ,0]	1	[1,1,1,0,. . . ,0]	2
[1,1,1,0,. . . ,0,1]	3	[1,1,1,0,. . . ,0,1,1]	4
[1,1,1,0,. . . ,0,1,1,1]	3	[1,1,1,0,. . . ,0,1,1,1,1]	2

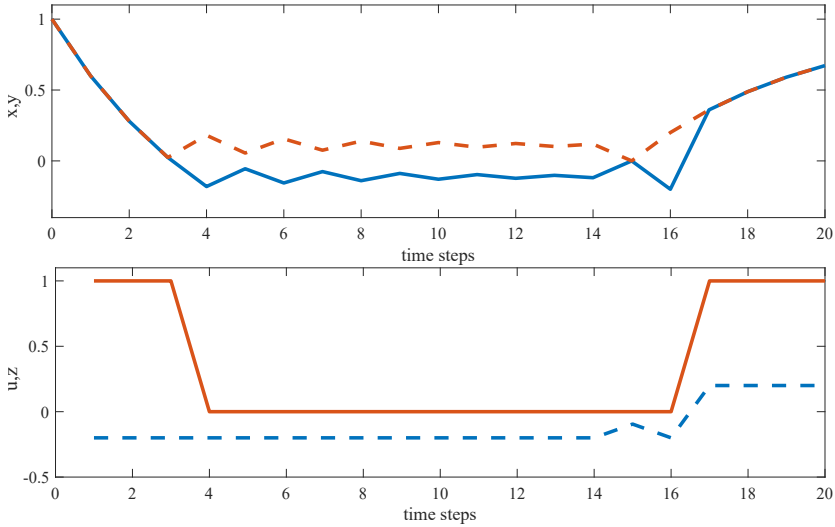

Figure 4.7: An example plot of the state and input trajectories for Problem (4.10) with $T = 20$ and $x_0 = 1$. (Top) Optimal state trajectories, where the solid red and dashed blue lines correspond to the trajectories of y and x, respectively. (Bottom) The solid red and dashed blue lines correspond to the optimal input trajectories for z and u, respectively.

A termination limit of 3000 seconds is set for each test, however, the simple standard branch-and-bound algorithm often failed to terminate within this time for many of the larger problems, which is the cause of some suboptimality. Despite its speed, move blocking removes large parts of the feasible set of the MIP and thus this strategy often fails to obtain a feasible point or gives a suboptimal solution. As Algorithm 12 is specifically designed for problems like (4.10), it is not surprising that it is best at exploiting the turnpike property and

Table 4.8: Aggregated results for Problem (4.10).

$T = 40$	B&B	move blocking	node weighting	seq. move blocking
avg. # nodes	1084	229.72	1003.71	35.13
median # nodes	1084	146	1019	30
avg. runtime (s)	3003.96	49.14	3003.27	2.77
best # nodes	1077	52	842	7
best runtime (s)	3000.09	4.29	3000.65	0.56
avg. subopt.	507.11	3.60	0	0
$T = 20$	B&B	move blocking	node weighting	seq. move blocking
avg. # nodes	1106.48	137.38	934.05	109.00
median # nodes	1106	86	890	84
avg. runtime (s)	3004.33	22.31	591.77	12.92
best # nodes	1105	32	508	3
best runtime (s)	3000.77	2.03	97.71	0.18
avg. subopt.	465.55	3.45	0	0
$T = 10$	B&B	move blocking	node weighting	seq. move blocking
avg. # nodes	679.33	4.2	183.62	93.36
median # nodes	628	4	106	62
avg. runtime (s)	1041.44	0.32	249.61	11.96
best # nodes	272	0	58	22
best runtime (s)	68.62	0.04	3.93	1.16
avg. subopt.	0	141.93	0	0

yields optimal solutions with minimal runtime. One interesting observation from Table 4.8 is that Algorithm 12 solves the instance of Problem (4.10) with $T = 40$ in less time on average than the instances with shorter time horizons. This is simply due to the fact that Algorithm 12 quickly terminates for $k_1 \geq 4$ – when the turnpike begins – and the results are aggregated for $1 \leq k_1 \leq T/2$.

Although node weighting yields solutions much more quickly than standard B&B, it is still nowhere near the speed of moveblocking or Algorithm 12. Even though node weighting allows for very good upper bounds to be quickly attained, many nodes must still be checked in order to establish a matching a lower bound and return an optimality certificate. An example of the progress of the upper and lower bounds per iteration is shown in Figure 4.8.

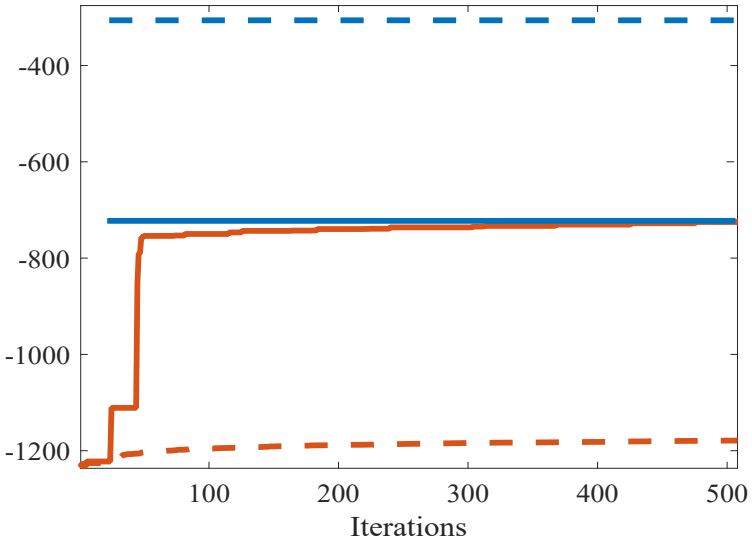

Figure 4.8: An example plot of the upper and lower bounds at each B&B iteration with node weighting (solid) and standard B&B (dashed). The optimal solution is found within 100 iterations by Algorithm 13, and the next 400 are needed to confirm its optimality.

Example 2

As a second example, we propose the following:

$$\begin{aligned} \min_{z} \quad & \sum_{t=0}^{N-1} 10u(t) + z(t) + 100u(t)^2 + 100x(t)^\top x(t) \\ \text{subject to } & \forall t \in \{0, \dots, N-1\}, \\ & x(t+1) = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix} x(t) + B_1 u(t) + B_2 z(t), \\ & v(t) \in \mathbb{R}^{n_x+1} \times \{0,\ 1\} \end{aligned} \tag{4.11}$$

with $I \in \mathbb{R}^{d \times d}$ is the identity matrix, $B_1 = \begin{bmatrix} 0 & 0 & \dots & 0 & 1 \end{bmatrix}^\top$ and $B_2 = \begin{bmatrix} 1 & 1 & \dots & 1 & 1 \end{bmatrix}^\top$. The turnpike is at $\bar{v}^\star = 0$. Notice that each state $x_i(t)$ in the state vector $x(t) = \begin{bmatrix} x_1(t), \dots, x_{n_x}(t) \end{bmatrix}^\top$ is coupled with the state $x_{i+d}(t-1)$ in addition to the control actions $u(t-1)$ and $z(t-1)$ This problem is interesting because turnpikes are often part of the optimal state trajectory of Problem (4.11), and the problem is scalable both in number of state variables as well as time horizon.

Once again, Algorithm 12 uses $k_1 \in \{1, \dots, T/2\}$, $k_2 = T$, and $K = 1$. The results for each choice of k_1 are aggregated in the results shown in Table 4.10.

The initial guesses $\mathcal{Z}_0$ for Algorithm 13 are constructed as partial guesses of the integer controls corresponding to $\overline{z}^\star = 0$ for $k \geq \hat{k}$ with $\hat{k} = \{2, \dots, 6\}$ as shown in Table 4.9.

Table 4.9: Guesses $\mathcal{Z}_0$ and weights $\mathcal{W}_0$ for Example (4.11) with $\hat{k} = 2$.

$\mathbf{Z}_{0,i}$	$w_{0,i}$
[·,0,·,. . . ,·]	$4 \cdot \max_{w \in \mathcal{W}} w$
[·,0,0,·,. . . ,·]	$4 \cdot \max_{w \in \mathcal{W}} w$
. . .	. . .
[·,0,. . . ,0,·]	$4 \cdot \max_{w \in \mathcal{W}} w$
[·,0,. . . ,0]	$4 \cdot \max_{w \in \mathcal{W}} w$

Table 4.10: Aggregated results for Problem (4.11).

$T = 20$ $n = 30$	B&B	move blocking	node weighting	seq. move blocking
avg. # nodes	516.72	2.67	595.58	17.85
median # nodes	342	0	420	6
avg. runtime (s)	545	0.94	543.67	4.27
best # nodes	110	0	4	6
best runtime (s)	32.63	0.26	1.40	1.35
avg. subopt.	0.12	147223	0.01	0.79
$T = 10$ $n = 30$	B&B	move blocking	node weighting	seq. move blocking
avg. # nodes	38	3.24	38.68	18.93
median # nodes	38	2	20	6
avg. runtime (s)	5.59	0.59	5.71	2.41
best # nodes	2	0	2	6
best runtime (s)	0.40	0.14	0.37	0.68
avg. subopt.	0.002	14055	0.002	0.002

Shown in Table 4.10 are the aggregated results for each combination of $x_0 = \{-0.9, -0.8, \ldots, 0.8, 0.9\} + r$, where r is a uniformly distributed random vector whose entries range between -0.1 and 0.1.

The results for Problem (4.11) mirror those for Problem (4.10), but also show how each of the methods performs when the number of real-valued decision variables is increased. Namely, the increase in the number of variables, and the nature of the coupling between them, causes the feasible set to become relatively sparse. This is disastrous for moveblocking since it leads to the algorithm falsely claiming infeasibility in most cases. Sequential move blocking is largely unaffected, and terminates quickly regardless of the number of real-valued variables. However, node weighting does manage to find the solutions albeit sometimes quite slowly. Interestingly, the number of nodes needed to

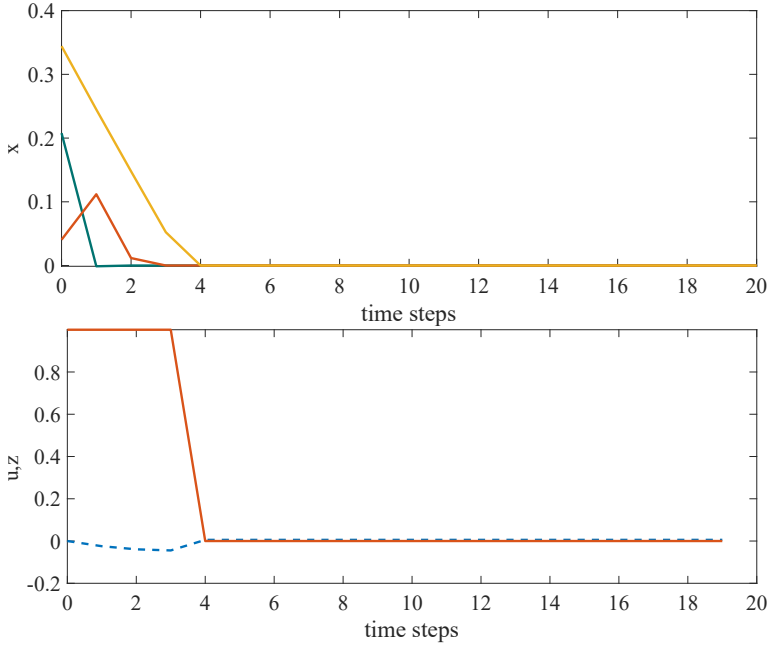

Figure 4.9: An example plot of the state and input trajectories for Example 2 with $T = 20$ and three state variables. The solid red and dashed blue lines correspond to the input trajectories for z and u, respectively.

be searched with the node weighting strategy goes up when more real-valued variables are introduced. This is in part due to the large quadratic penalty term for the state variables which dominate the objective value and lead to many nodes with very similar objective values and thus much more difficult to prune from the B&B decision tree. Figure 4.9 depicts an example plot for the case with $T = 20$ and three state variables. Note the long turnpike from $t = 4$ to $t = 20$. This feature is successfully exploited by Algorithm 12 to yield the excellent scalability observed in Table 4.10 between $T = 10$ and $T = 20$. Unfortunately, Algorithm 13 does not seem to share this property; however it may be the case that a different initial guess would further reduce runtime for this problem and yield similar results to that of Algorithm 12.

5 Conclusions and Outlook

As detailed in Chapter 2, there are a number of challenges and open problems in mixed-integer programming. Most of these stem from the non-convex nature of MIPs, which often makes verifying the optimality of a potential solution quite difficult. Nonetheless, as demonstrated by numerous examples throughout the dissertation, this class of problems appears often in power and energy systems.

To date, there has been very limited research into distributed optimization methods for mixed-integer problems, which would allow for improved tractability, runtime, and applicability to multi-agent settings. To this end, Chapter 3 presented three main algorithms for distributed mixed-integer optimization:

- Mixed-Integer ALADIN
- PaDOA
- Distributed Branch & Bound

Convergence and optimality properties of each of these algorithms was analyzed, and their preformance was evaluated using a variety of benchmark examples.

Mixed-Integer ALADIN is shown to have limited convergence and optimality guarantees, although it is applicable to both a large class of partially separable MIPs and to multi-agent systems. Despite theoretical limitations to its convergence, it is shown to quickly converge for both MIQP battery scheduling problems along with non-convex MINLP reactive power dispatch.

PaDOA has convergence and global optimality guarantees for MICPs, and in fact is proven to converge in a single iteration given certain conditions are satisfied. These properties are verified by both the battery scheduling and TCL problems. Despite only a prototypical implementation, it is shown to outperform the commercial solvers gurobi and CPLEX in several cases.

Distributed Branch & Bound shares the convergence and optimality guarantees of standard B&B, although it has the potential to be faster and more scalable through parallelization. This conjecture was tested on two benchmark problems, one of which was the classic battery scheduling problem used to evaluate PaDOA and Mixed-Integer ALADIN, and the other was a custom-made MIP. Interestingly, despite its positive theoretical properties, its actual performance for the battery scheduling problem was poor compared to both other distributed mixed-integer algorithms.

It is noted in Chapter 3 that the performance of distributed optimization algorithms (mixed-integer or otherwise) are dependent on the given problem decomposition. While not all power and energy problems are freely decomposable, many are. In particular, the optimal power flow and reactive power dispatch problems have asymmetric, graph-like structures which make them interesting candidates for an investigation into problem decomposition. Both the RPD and OPF problems are partitioned in Section 4.1 using four different state-of-the-art graph partitioning techniques. The numerical results of Mixed-Integer ALADIN applied to the partitioned problems demonstrate the difficulty involved with solving large-scale MIPs. Nonetheless, these results represent some first steps towards more general results on optimal partitioning for the five distributed mixed-integer optimization algorithms that were presented.

In Section 4.2 the focus changes from spatial decomposition to exploitation of temporal structures in mixed-integer optimal control problems. Specifically, the turnpike phenomenon is exploited to great effect using two distinct methods:

- Sequential Moveblocking
- Node weighting

Sequential moveblocking is proven to converge within a finite number of iterations with an optimal solution given that the optimal control problem is an MICP. Two benchmark examples demonstrate that this algorithm is significantly faster than standard techniques, and it achieves this by specializing to MIOCPs with turnpikes.

Node weighting is given as an alternative technique which sacrifices speed for flexibility. It maintains the convergence and optimality guarantees of standard B&B, but presents a means of effectively taking advantage of *a priori* infor-

mation in the form of a collection of (partial) guesses. While somewhat less efficient than sequential moveblocking at exploiting the existence of turnpikes in MIOCPs, it nonetheless has the potential for application to general MIPs and thus can be recommended when the existence of a turnpike is not guaranteed.

Despite some runtime advantages, many of the algorithms presented in Chapters 3 and 4 are still prototypical and could be greatly improved through the combination of other methods. For example, the use of other lower bounding functions in Algorithm 6 could be investigated. The use of quadratic lower bounding functions in [FL94] was found to yield some benefit for certain problems when implemented in Algorithm 4, and something similar could be attempted with Algorithm 6 as well. As noted in Section 3.2, MIQPs can be solved much more efficiently than their MILP counterparts, and thus an MIQP coupling subproblem in Algorithm 6 could lead to faster runtime and applicability to a wider class of problems.

There are numerous other variants of the presented algorithms which could allow for better results in different situations. For example, if the problem is such that the coupling step cannot be solved by a central coordinator then it would be interesting to investigate how this could be performed in a more decentralized manner. Furthermore, the cutting plane methods and other heuristics used by Gurobi and CPLEX could be implemented within Algorithm 6 or Algorithm 7 to reduce overall runtime.

In Section 4.1.1, partitioning results for Algorthm 5 were given, however it would be interesting to see whether the same partitioning strategies that worked well for Mixed-Integer ALADIN would also improve the convergence rate of PaDOA or Distributed Branch & Bound. It is also worth questioning whether one ought to search for optimal partitioning strategies for distributed optimization, or whether a more elegant approach may be possible. For example, it is conceivable that the central agent which solves the coupling problems could also dynamically adjust the partitions to iteratively improve performance. This would sidestep the need for computation of an initial partitioning in favour of generating it *throughout* the optimization process. Such a dynamic partitioning method is a priority for future research.

While the Sequential Moveblocking algorithm of Section 4.2.2 is shown to have favourable properties for MIOCPs with turnpikes, one of its weaknesses is that it requires guesses as to where the turnpike begins and ends in the MIOCP. It

may be possible to use machine learning to obtain sufficiently good guesses, however this is left to future research. Finally, it should be noted that all of the subproblems in Sections 4.2.2 and 4.2.3 are solved centrally. In principle, one could extend these algorithms to a distributed computing context. It would be particularly interesting to see how node weighting would affect the results of Algorithm 7.

Bibliography

[ABRM14] D. Axehill, T. Besselmann, D.M. Raimondo, and M. Morari. A parametric branch and bound approach to suboptimal explicit hybrid mpc. *Automatica*, 50(1):240–246, 2014.

[AGH+19] J.A.E. Andersson, J. Gillis, G. Horn, J.B. Rawlings, and M. Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019.

[AK87] B.D.O. Anderson and P.V. Kokotovic. Optimal control problems over large time intervals. *Automatica*, 23(3):355–363, 1987.

[AKM05] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

[AOM+18] R. Appino, A. Ordiano, R. Mikut, T. Faulwasser, and V. Hagenmeyer. On the use of probabilistic forecasts in scheduling of renewable energy sources coupled to storages. *Applied Energy*, 210:1207–1218, 2018.

[ApS19] MOSEK ApS. *The MOSEK optimization toolbox for MATLAB manual. Version 9.0*, 2019. http://docs.mosek.com/9.0/toolbox/index.html.

[ARLAH19] N. Azizan-Ruhi, F. Lahouti, A.S. Avestimehr, and B. Hassibi. Distributed solution of large-scale linear systems via accelerated projection-based consensus. *IEEE Transactions on Signal Processing*, 67(14):3806–3817, 2019.

[Bal79] E. Balas. Disjunctive programming. In *Annals of Discrete Mathematics*, volume 5, pages 3–51. Elsevier, 1979.

[BBC+08] P. Bonami, L.T. Biegler, A.R. Conn, G. Cornuéjols, I.E. Grossmann, C.D. Laird, J. Lee, A. Lodi, F. Margot, N. Sawaya, and A. Wächter. An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optimization*, 5(2):186–204, may 2008.

[BDCHJ10] A. Bemporad, S. Di Cairano, E. Henriksson, and K.H. Johansson. Hybrid model predictive control based on wireless sensor feedback: An experimental study. *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal*, 20(2):209–225, 2010.

[BDM03] M.R. Bussieck, A.S. Drud, and A. Meeraus. MINLPLib—a collection of test models for mixed-integer nonlinear programming. *INFORMS Journal on Computing*, 15(1):114–119, 2003.

[Ben62] J.F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Matematik*, 4(1):238–252, 1962.

[Ber06] D.P. Bertsekas. Lagrange multipliers with optimal sensitivity properties in constrained optimization. In *Large-Scale Nonlinear Optimization*. Springer, 2006.

[BFG+18] P. Braun, T. Faulwasser, L. Grüne, C. Kellett, S. Weller, and K. Worthmann. Hierarchical distributed ADMM for predictive control with applications in power networks. *IFAC Journal on Systems and Control*, 3:10–22, 2018.

[BFL07] L. Bertacco, M. Fischetti, and A. Lodi. A feasibility pump heuristic for general mixed-integer problems. *Discrete Optimization*, 4(1):63–76, 2007.

[BFR+15] U. Bösenberg, M. Falk, C.G. Ryan, R. Kirkham, M. Menzel, J. Janek, M. Fröba, G. Falkenberg, and U.E.A. Fittschen. Correlation between chemical and morphological heterogeneities in LiNi0.5mn1.5o4 spinel composite electrodes for lithium-ion batteries determined by micro-x-ray fluorescence analysis. *Chemistry of Materials*, 27(7):2525–2531, 2015.

[BGMT13] W.A. Bukhsh, A. Grothey, K.I.M. McKinnon, and P.A. Trodden. Local solutions of the optimal power flow problem. *IEEE Transactions on Power Systems*, 28(4):4780–4788, 2013.

[BGW03] U. Brandes, M. Gaertler, and D. Wagner. Experiments on graph clustering algorithms. In *European Symposium on Algorithms*, pages 568–579, 2003.

[BHDS02] A. Bemporad, W.P. Maurice H. Heemels, and B. De Schutter. On hybrid systems and closed-loop mpc systems. *IEEE Transactions on Automatic Control*, 47(5):863–869, 2002.

[Bix12] R.E. Bixby. A brief history of linear and mixed-integer programming computation. *Documenta Mathematica*, pages 107–121, 2012.

[BKL+13] P. Belotti, C. Kirches, S. Leyffer, J. Linderoth, J. Luedtke, and A. Mahajan. Mixed-integer nonlinear optimization. *Acta Numerica*, 22:1–131, 2013.

[BKMP18] H.G. Bock, C. Kirches, A. Meyer, and A. Potschka. Numerical solution of optimal control problems with explicit and implicit switches. *Optimization Methods and Software*, 33(3):450–474, 2018.

[BL12] S. Burer and A.N. Letchford. Non-convex mixed-integer nonlinear programming: A survey. *Surveys in Operations Research and Management Science*, 17(2):97–106, 2012.

[BM99] A. Bemporad and M. Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3):407–427, 1999.

[BMS+16] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz. Recent advances in graph partitioning. In *Algorithm Engineering*, pages 117–158. Springer, 2016.

[BN18] A. Bemporad and V.V. Naik. A numerically robust mixed-integer quadratic programming solver for embedded hybrid model predictive control. *IFAC-PapersOnLine*, 51(20):412–417, 2018.

[BNKF98] W. Banzhaf, P. Nordin, R.E. Keller, and F.D. Francone. *Genetic Programming: An Introduction*, volume 1. Morgan Kaufmann San Francisco, 1998.

[Bol13] 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.

[BOW16] M. Baes, T. Oertel, and R. Weismantel. Duality for mixed-integer convex minimization. *Mathematical Programming*, 158(1-2):547–564, 2016.

[BPC+11] 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.

[BT95] R. Battiti and G. Tecchiolli. Local search with memory: Benchmarking RTS. *Operations-Research-Spektrum*, 17(2-3):67–86, 1995.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge university press, 2004.

[CGKM07] R. Cagienard, P. Grieder, E.C. Kerrigan, and M. Morari. Move blocking strategies in receding horizon control. *Journal of Process Control*, 17(6):563–570, 2007.

[Chv73] V. Chvatal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete mathematics*, 4(4):305–337, 1973.

[CL18] C.G. Cassandras and J. Lygeros. *Stochastic hybrid systems*. CRC Press, 2018.

[Cla99a] M. Clagett. *Ancient Egyptian Science: A Source Book. Volume 3: Ancient Egyptian Mathematics*. American Philosophical Society, 1999.

[Cla99b] J. Clausen. Branch and bound algorithms-principles and examples. *Department of Computer Science, University of Copenhagen*, pages 1–30, 1999.

[CLS19] M.B. Cohen, Y.T. Lee, and Z. Song. Solving linear programs in the current matrix multiplication time. In *Proceedings of the 51st annual ACM SIGACT symposium on theory of computing*, pages 938–942, 2019.

[CYS18] R. Carli, K.Sinan. Yíldírím, and L. Schenato. Multi-agent distributed optimization algorithms for partition-based linear programming (LP) problems. In *2018 European Control Conference (ECC)*, pages 1462–1467, 2018.

[Dan48] G.B. Dantzig. Programming in a linear structure. In *Bulletin of the American Mathematical Society*, volume 54, pages 1074–1074, 1948.

[Dan81] G.B. Dantzig. Reminiscences about the origins of linear programming. Technical report, STANFORD UNIV CA SYSTEMS OPTIMIZATION LAB, 1981.

[Deu17] Deutscher Wetterdienst. `ftp://ftp-cdc.dwd.de/pub/CDC/observations_germany/climate/10_minutes/solar/historical/`, 2017.

[DFJ54] 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.

[DG86] M.A. Duran and I.E. Grossmann. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36(3):307–339, oct 1986.

[DGE+02] R. Drechsler, W. Gunther, T. Eschbach, L. Linhard, and G. Angst. Recursive bi-partitioning of netlists for large number of partitions. In *Proceedings Euromicro Symposium on Digital System Design. Architectures, Methods and Tools*, pages 38–44, 2002.

[DHSZ03] K. Driscoll, B. Hall, H. Sivencrona, and P. Zumsteg. Byzantine fault tolerance, from theory to reality. In *International Conference on Computer Safety, Reliability, and Security*, pages 235–248, 2003.

[Dor58] K.S. Dorfmann. RM linear programming and economic analysis. *New York*, 1958.

[DW60] G.B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8(1):101–111, 1960.

[EB92] J. Eckstein and D.P. Bertsekas. On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators. *Mathematical Programming*, 55:293–318, 1992.

[EK95] R. Eberhart and J. Kennedy. Particle swarm optimization. In *Proceedings of the IEEE international conference on neural networks*, volume 4, pages 1942–1948, 1995.

[EMJ+17] A. Engelmann, T. Mühlpfordt, Y. Jiang, B. Houska, and T. Faulwasser. Distributed AC optimal power flow using ALADIN. *IFAC-PapersOnLine*, 50(1):5536–5541, 2017.

[Ers14] T. Erseghe. Distributed optimal power flow using ADMM. *IEEE Transactions on Power Systems*, 29(5):2370–2380, sep 2014.

[Eve63] H. Everett. Generalized Lagrange multiplier method for solving problems of optimum allocation of resources. *Operations Research*, 11(3):399–417, 1963.

[FFCM15] J. Fu, J.M.M. Faust, B. Chachuat, and A. Mitsos. Local optimization of dynamic programs with guaranteed satisfaction of path constraints. *Automatica*, 62:184–192, 2015.

[FGM18] T. Faulwasser, L. Grüne, and M.A. Müller. Economic nonlinear model predictive control. *Foundations and Trends® in Systems and Control*, 5(1):1–98, 2018.

[Fis95a] A. Fischer. A newton-type method for positive-semidefinite linear complementarity problems. *Journal of Optimization Theory and Applications*, 86(3):585–608, 1995.

[Fis95b] A. Fischer. On the local superlinear convergence of a newton-type method for lcp under weak conditions. *Optimization Methods and Software*, 6(2):83–107, 1995.

[FKJB17] T. Faulwasser, M. Korda, C.N. Jones, and D. Bonvin. On turnpike and dissipativity properties of continuous-time optimal control problems. *Automatica*, 81:297–304, 2017.

[FL94] R. Fletcher and S. Leyffer. Solving mixed integer nonlinear programs by outer approximation. *Mathematical Programming*, 66(1-3):327–349, aug 1994.

[FL10] M. Fischetti and A. Lodi. Heuristics in mixed integer programming. *Wiley Encyclopedia of Operations Research and Management Science*, 2010.

[FM20] T. Faulwasser and A. Murray. Turnpike properties in discrete-time mixed-integer optimal control. *IEEE Control Systems letters*, 4(3):704–709, 2020. arXiv:2002.02049.

[FOBK12] K. Flaßkamp, S. Ober-Blöbaum, and M. Kobilarov. Solving optimal control problems by exploiting inherent dynamical systems structures. *Journal of Nonlinear Science*, 22(4):599–629, 2012.

[Fru77] M.A. Frumkin. Polynomial time algorithms in the theory of linear diophantine equations. In *International Conference on Fundamentals of Computation Theory*, pages 386–392. Springer, 1977.

[Ful63] A.T. Fuller. Study of an optimum nonlinear control system. *Journal of Electronics and Control*, 15:63–71, 1963.

[Gau09] C.F. Gauss. *Theoria motus corporum coelestium in sectionibus conicis solem ambientium*, volume 7. Perthes et Besser, 1809.

[Geo72] A.M. Geoffrion. Generalized benders decomposition. *Journal of Optimization Theory and Applications*, 10(4):237–260, 1972.

[Gey09] T. Geyer. Generalized model predictive direct torque control: Long prediction horizons and minimization of switching losses. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 6799–6804. IEEE, 2009.

[GG18] L. Grüne and R. Guglielmi. Turnpike properties and strict dissipativity for discrete time linear quadratic optimal control problems. *SIAM Journal on Control and Optimization*, 56(2):1282–1302, 2018.

[GHT16a] J. Guo, G. Hug, and O. K. Tonguz. Intelligent partitioning in distributed optimization of electric power systems. *IEEE Transactions on Smart Grid*, 7(3):1249–1258, 2016.

[GHT16b] J. Guo, G. Hug, and O.K. Tonguz. A case for non-convex distributed optimization in large-scale power systems. *IEEE Transactions on Power Systems*, 32(5):3842–3851, 2016.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, first edition edition, 1979.

[GJK88] E.G. Gilbert, D.W. Johnson, and S.S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4(2):193–203, 1988.

[GM75] R. Glowinski and A. Marrocco. Sur l'approximation, par elements finits d'ordre un, et la resolution, par penalisation-dualite, d'une classe de problems de dirichlet non lineares. *Rev. Francaise Automati. Inform. Recherche Oper.*, 9:41–76, 1975.

[GM76] 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.

[Gom58] R.E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical society*, 64(5):275–278, 1958.

[Grü13] L. Grüne. Economic receding horizon control without terminal constraints. *Automatica*, 49(3):725–734, 2013.

[GT96] J. Gondzio and T. Terlaky. A computational view of interior point methods. *JE Beasley. Advances in linear and integer programming. Oxford Lecture Series in Mathematics and its Applications*, 4:103–144, 1996.

[GTSJ15] 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, mar 2015.

[Gur19] Gurobi. Gurobi optimizer reference manual, 2019.

[Han88] S.P. Han. A successive projection method. *Mathematical Programming*, 40(1-3):1–14, 1988.

[HBO14] H. Hijazi, P. Bonami, and A. Ouorou. An outer-inner approximation for separable mixed-integer nonlinear programs. *INFORMS Journal on Computing*, 26(1):31–44, 2014.

[HFD16] B. Houska, J. Frasch, and M. Diehl. An augmented lagrangian based algorithm for distributed Non-Convex optimization. *SIAM Journal on Optimization*, 26(2):1101–1127, jan 2016.

[HGA09] G. Hug-Glanzmann and G. Andersson. Decentralized optimal power flow control for overlapping areas in power systems. *IEEE Transactions on Power Systems*, 24(1):327–336, 2009.

[HKJD18] B. Houska, D. Kouzoupis, Y. Jiang, and M. Diehl. Convex optimization with ALADIN. *Math. Program.*, pages 1–22, 2018.

[Hor72] S. Horsley. Being an account of his method of finding all the prime numbers. *Philosophical Transactions of the Royal Society of London*, (62):327–347, 1772.

[HSS10] M. Holtgrewe, P. Sanders, and C. Schulz. Engineering a scalable high quality graph partitioner. In *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, pages 1–12, 2010.

[Hun83] M.S. Hung. A polynomial simplex method for the assignment problem. *Operations Research*, 31(3):595–600, 1983.

[HYW00] B.S. He, H. Yang, and S.L. Wang. Alternating direction method with self-adaptive penalty parameters for monotone variational inequalities. *Journal of Optimization Theory and applications*, 106(2):337–356, 2000.

[Iba94] K. Iba. Reactive power optimization by genetic algorithm. *IEEE Transactions on Power Systems*, 9(2):685–692, 1994.

[IBM19] IBM. IBM ILOG CPLEX optimization studio CPLEX user's manual, 2019.

[IT02] T. Illés and T. Terlaky. Pivot versus interior point methods: Pros and cons. *European Journal of Operational Research*, 140(2):170–190, 2002.

[Jer78] R. Jeroslow. Cutting-plane theory: Algebraic methods. *Discrete mathematics*, 23(2):121–150, 1978.

[Joh79] E.L. Johnson. On the group problem and a subadditive approach to integer programming. *Annals of Discrete Mathematics*, 5:97–112, 1979.

[KAGB04] P. Kesavan, R.J. Allgor, E.P. Gatzke, and P.I. Barton. Outer approximation algorithms for separable nonconvex mixed-integer nonlinear programs. *Mathematical Programming*, 100(3):517–535, 2004.

[Kal02] E. Kalvelagen. Benders decomposition with gams. *Amsterdam Optimization Modeling Group: Washington, DC, USA*, 2002.

[Kar84] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311. ACM, 1984.

[KBSS11] C. Kirches, H.G. Bock, J.P. Schlöder, and S. Sager. Block-structured quadratic programming for the direct multiple shooting method for optimal control. *Optimization Methods & Software*, 26(2):239–257, 2011.

[Kha79] L.G. Khachiyan. A polynomial algorithm in linear programming. In *Doklady Academii Nauk SSSR*, volume 244, pages 1093–1096, 1979.

[KK84] V. Kumar and L.N. Kanal. Parallel branch-and-bound formulations for and/or tree search. *IEEE transactions on pattern analysis and machine intelligence*, (6):768–778, 1984.

[KK99] G. Karypis and V. Kumar. A fast and highly quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1999.

[KKTK75] P.R. Kleindorfer, C.H. Kriebel, G.L. Thompson, and G.B. Kleindorfer. Discrete optimal control of production plans. *Management Science*, 22(3):261–273, 1975.

[KM70] V. Klee and G.J. Minty. How good is the simplex algorithm? Technical report, WASHINGTON UNIV SEATTLE DEPT OF MATHEMATICS, 1970.

[KM08] D Kempe and F. McSherry. A decentralized algorithm for spectral analysis. *Journal of Computer and System Sciences*, 74(1):70–83, 2008.

[KMC11] S. Koch, J.L. Mathieu, and D.S. Callaway, editors. *Modeling and Control of Aggregated Heterogeneous Thermostatically Controlled Loads for Ancillary Services*. Proc. PSCC, 2011.

[KMS+20] M. Kyesswa, A. Murray, P. Schmurr, H.C. Çakmak, U. Kühnapfel, and V. Hagenmeyer. Impact of grid partitioning algorithms on combined distributed AC optimal power flow and parallel dynamic power grid simulation. *IEEE Transactions on Power Systems (submitted)*, 2020.

[LBR15] J. Liu, M. Benosman, and A.U. Raghunathan. Consensus-based distributed optimal power flow algorithm. In *2015 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, pages 1–5, 2015.

[LCW+07] C. H. Liang, C. Y. Chung, K. P. Wong, X. Z. Duan, and C. T. Tse. Study of differential evolution for optimal reactive power flow. *Transmission Distribution IET Generation*, 1(2):253–260, 2007.

[LD60] A.H Land and A.G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[LFPL15] J. Liang, J. Fadili, G. Peyré, and R. Luke. Activity identification and local linear convergence of douglas–rachford/admm under partial smoothness. In *International Conference on Scale Space and Variational Methods in Computer Vision*, pages 642–653, 2015.

[LH09] M. Lazar and W.P.M.H. Heemels. Predictive control of hybrid systems: Input-to-state stability results for sub-optimal solutions. *Automatica*, 45(1):180–185, 2009.

[Llo82] S. Lloyd. Least squares quantization in PCM. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[LRKS91] J.K. Lenstra, Alexander H.G. Rinnooy K., and A. Schrijver. *History of mathematical programming: a collection of personal reminiscences*. Centrum voor Wiskunde en Informatica, 1991.

[LS99] J.T. Linderoth and M.W.P. Savelsbergh. A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing*, 11(2):173–187, 1999.

[LS15] Y.T. Lee and A. Sidford. Efficient inverse maintenance and faster algorithms for linear programming. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 230–249. IEEE, 2015.

[LSKVI10] F. Logist, S. Sager, C. Kirches, and J.F. Van Impe. Efficient multiple objective optimal control of dynamic systems with integer controls. *Journal of Process Control*, 20(7):810–822, 2010.

[LSP82] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[LU05] D.N. Laikov and Y.A. Ustynyuk. PRIRODA-04: a quantum-chemical program suite. new possibilities in the study of molecular systems with the application of parallel computing. *Russian chemical bulletin*, 54(3):820–826, 2005.

[LYBV18] M. Lubin, E. Yamangil, R. Bent, and J.P. Vielma. Polyhedral approximation in mixed-integer convex optimization. *Mathematical Programming*, 172(1-2):139–168, 2018.

[Mac67] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297, 1967.

[McK76] L.W. McKenzie. Turnpike theory. *Econometrica: Journal of the Econometric Society*, pages 841–865, 1976.

[MDS+17] D.K. Molzahn, F. Dörfler, H. Sandberg, S.H. Low, S. Chakrabarti, R. Baldick, and J. Lavaei. A survey of distributed optimization and control algorithms for electric power systems. *IEEE Transactions on Smart Grid*, 8(6):2941–2962, 2017.

[MEHF18] A. Murray, A. Engelmann, V. Hagenmeyer, and T. Faulwasser. Hierarchical distributed mixed-integer optimization for reactive power dispatch. In *10th Symposium on Control of Power and Energy Systems (CPES 2018)*, volume 51, pages 368–373, 2018.

[Mey12] H. Meyerhenke. Shape optimizing load balancing for MPI-parallel adaptive numerical simulations. *Graph Partitioning and Graph Clustering*, 588:67–81, 2012.

[MFH18] A. Murray, T. Faulwasser, and V. Hagenmeyer. Mixed-integer vs. real-valued formulations of distributed battery scheduling problems. In *10th Symposium on Control of Power and Energy Systems (CPES 2018)*, volume 51, pages 350–355, 2018.

[MFH+20] A. Murray, T. Faulwasser, V. Hagenmeyer, M.E. Villanueva, and B. Houska. Partially distributed outer approximation. *Journal of Global Optimization (submitted)*, 2020.

[MH20a] A. Murray and V. Hagenmeyer. A new distributed optimization algorithm for MINLPs with affinely coupled decision variables. *4th International Conference on Algorithms, Computing and Systems (in print)*, 2020.

[MH20b] A. Murray and V. Hagenmeyer. On convergence of mixed-integer aladin. *4th International Conference on Algorithms, Computing and Systems (in print)*, 2020.

[MHF18] A. Murray, V. Hagenmeyer, and T. Faulwasser. Assessment of a multi-agent mixed-integer optimization algorithm for battery scheduling. In *Proceedings of the Ninth International Conference on Future Energy Systems*, pages 453–455. ACM, 2018. Awarded the Audience Choice award at the ACM International Conference on Future Energy Systems 2018.

[MHF20] A. Murray, V. Hagenmeyer, and T. Faulwasser. Efficient numerical solution of mixed-integer optimal control with turnpikes. *(TBD)*, 2020.

[mip18] MIPLIB 2017, 2018. http://miplib.zib.de.

[Mit03] H.D. Mittelmann. An independent benchmarking of SDP and SOCP solvers. *Mathematical Programming*, 95(2):407–430, 2003.

[MK87] K.G. Murty and S.N. Kabadi. Some np-complete problems in quadratic and nonlinear programming. *Mathematical Programming*, 39:117–129, 1987.

[MKS+20] A. Murray, M. Kyesswa, P. Schmurr, H.K. Çakmak, and V. Hagenmeyer. A comparison of partitioning strategies in ac optimal power flow. *IEEE PES Innovative Smart Grid Technologies Europe (ISGT-Europe) (in print)*, 2020.

[MM57] H.M. Markowitz and A.S. Manne. On the solution of discrete programming problems. *Econometrica: journal of the Econometric Society*, pages 84–110, 1957.

[MP01] M.A. Mamedov and S. Pehlivan. Statistical cluster points and turnpike theorem in nonconvex problems. *Journal of mathematical analysis and applications*, 256(2):686–693, 2001.

[MS07] J. Maue and P. Sanders. Engineering algorithms for approximate weighted matching. In *International Workshop on Experimental and Efficient Algorithms*, pages 242–255, 2007.

[MSP+20] A. Murray, T. Schütz, G. Pan, D. Müller, and V. Hagenmeyer. On modelling effects in the battery and thermal storage scheduling problem. *Journal of Building Performance Simulation (accepted)*, 2020.

[NJW02] A.Y. Ng, M.I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, pages 849–856, 2002.

[NS08] I. Necoara and J.A.K. Suykens. Application of a smoothing technique to decomposition in convex optimization. *IEEE Transactions on Automatic Control*, 53(11):2674–2679, 2008.

[NW06] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Science & Business Media, 2006.

[Pel09] F. Pellegrini. Distillating knowledge about SCOTCH. In *Dagstuhl Seminar Proceedings*, 2009.

[PR91] 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.

[PS82] C.H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity*. Prentice Hall, 1982.

[PW00] F.A. Potra and S.J. Wright. Interior-point methods. *Journal of Computational and Applied Mathematics*, 124(1-2):281–302, 2000.

[RP08] G.R. Raidl and J. Puchinger. Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization. In *Hybrid metaheuristics*, pages 31–62. Springer, 2008.

[RWKM15] E.L. Ratnam, S.R. Weller, C.M. Kellett, and A.T. Murray. Residential load and rooftop PV generation: an australian distribution network dataset. *International Journal of Sustainable Energy*, pages 1–20, oct 2015.

[Sag05] S. Sager. *Numerical methods for mixed-integer optimal control problems*. PhD thesis, Universitaet Heidelberg, 2005.

[Sav94] M.W.P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6(4):445–454, 1994.

[Sca09] R. Scattolini. Architectures for distributed and hierarchical model predictive control – a review. *Journal of Process Control*, 19(5):723–731, may 2009.

[SG91] N.V. Sahinidis and I.E. Grossmann. Convergence properties of generalized benders decomposition. *Computers & Chemical Engineering*, 15(7):481–491, 1991.

[SGZ+12] H. Sun, Q. Guo, B. Zhang, W. Wu, and B. Wang. An adaptive zone-division-based automatic voltage control system with applications in china. *IEEE Transactions on Power Systems*, 28(2):1816–1828, 2012.

[SHH+16] S. Schlag, V. Henne, T. Heuer, H. Meyerhenke, P. Sanders, and C. Schulz. k-way hypergraph partitioning via n-level recursive bisection. In *2016 Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 53–67, 2016.

[SM06] M. Schlegel and W. Marquardt. Detection and exploitation of the control switching structure in the solution of dynamic optimization problems. *Journal of Process Control*, 16(3):275–290, 2006.

[SM15] R.C. Shekhar and C. Manzie. Optimal move blocking strategies for model predictive control. *Automatica*, 61:27–34, 2015.

[SMY+08] T. Senjyu, Y. Miyazato, A. Yona, N. Urasaki, and T. Funabashi. Optimal distribution voltage control and coordination with distributed generation. *IEEE Transactions on Power Delivery*, 23(2):1236–1242, 2008.

[SOM04] O. Stein, J. Oldenburg, and W. Marquardt. Continuous reformulations of discrete–continuous optimization problems. *Computers & chemical engineering*, 28(10):1951–1966, 2004.

[SRLBM12] A.G. Sanchez, Adriel M. Rizzato L., G. Bärwaldt, and M.G. Molina. Análisis regional de la red eléctrica de alemania en relación con la creciente penetración de generación solar fotovoltaica distribuida. *Revista de Ciencia y Tecnología*, (18):21–27, 2012.

[SS11] P. Sanders and C. Schulz. Engineering multilevel graph partitioning algorithms. In *European Symposium on Algorithms*, pages 469–480, 2011.

[SS13a] P. Sanders and C. Schulz. High quality graph partitioning. *Graph Partitioning and Graph Clustering*, 588(1):1–17, 2013.

[SS13b] P. Sanders and C. Schulz. Think locally, act globally: Highly balanced graph partitioning. In *International Symposium on Experimental Algorithms*, pages 164–175, 2013.

[Tes17] Tesla. www.tesla.com/powerwall, 2017.

[TMBB20] R. Takapoui, N. Moehle, S. Boyd, and A. Bemporad. A simple effective heuristic for embedded mixed-integer quadratic programming. *International Journal of Control*, 93(1):2–12, 2020.

[TMW09] M.A. Tomim, J.R. Marti, and L. Wang. Parallel solution of large power system networks using the multi-area thévenin equivalents (mate) algorithm. *International Journal of Electrical Power & Energy Systems*, 31(9):497–503, 2009.

[TS80] G.L. Thompson and S.P. Sethi. Turnpike horizons for production planning. *Management Science*, 26(3):229–241, 1980.

[Tse01] P. Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of Optimization Theory and Applications*, 109(3):475–494, 2001.

[Ulb04] S. Ulbrich. On the superlinear local convergence of a filter-sqp method. *Mathematical Programming*, 1(11):217–245, 2004.

[Vai89] P.M. Vaidya. Speeding-up linear programming using fast matrix multiplication. In *30th Annual Symposium on Foundations of Computer Science*, pages 332–337, 1989.

[Van15] R.J. Vanderbei. *Linear programming*. Springer, 2015.

[vN38] J. von Neumann. Über ein Ökonomisches Gleichungssystem und eine Verallgemeinerung des Brouwerschen Fixpunktsatzes. In *Ergebnisse eines Mathematischen Seminars*. 1938.

[VNR+06] J.A. Vrugt, B.O. Nualláin, B.A. Robinson, W. Bouten, S.C. Dekker, and Peter M.A. Sloot. Application of parallel computing to stochastic parameter estimation in environmental models. *Computers & Geosciences*, 32(8):1139–1155, 2006.

[Wal85] J. Wallis. A treatise of algebra, both historical and practical. *Philosophical Transactions of the Royal Society of London*, 15(173):1095–1106, 1685.

[WB05] A. Wächter and L.T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, apr 2005.

[Wil72] J.C. Willems. Dissipative dynamical systems part i: General theory. *Archive for rational mechanics and analysis*, 45(5):321–351, 1972.

[WK72] R. Wilde and P.V. Kokotovic. A dichotomy in linear control theory. *IEEE Trans. Automat. Contr.*, 17(3):382–383, 1972.

[WL01] S.L. Wang and L.Z. Liao. Decomposition method with a variable parameter for a class of monotone variational inequality problems. *Journal of optimization theory and applications*, 109(2):415–429, 2001.

[Wri15] S. Wright. Coordinate descent algorithms. *Mathematical Programming, Series B*, 151(1):3–34, 2015.

[YKF+00] H. Yoshida, K. Kawata, Y. Fukuyama, S. Takayama, and Y. Nakanishi. A particle swarm optimization for reactive power and voltage control considering voltage security assessment. *IEEE Transactions on Power Systems*, 15(4):1232–1239, 2000.

[YLCW06] W. Yan, F. Liu, C. Y. Chung, and K. P. Wong. A hybrid genetic algorithm-interior point method for optimal reactive power flow. *IEEE Transactions on Power Systems*, 21(3):1163–1169, 2006.

[YXZ+96] Z. Yan, N.D. Xiang, B.M. Zhang, S.Y. Wang, and T.S. Chung. A hybrid decoupled approach to optimal power flow. *IEEE transactions on power systems*, 11(2):947–954, 1996.

[ZKF+12] W. Zhang, K. Kalsi, J. Fuller, M. Elizondo, and D. Chassin. Aggregate model for heterogeneous thermostatically controlled loads with demand response. In *2012 IEEE Power and Energy Society General Meeting*, pages 1–8, 2012.