Advances in Robust and Adaptive Optimization: Algorithms, Software, and Insights

by

Iain Robert Dunning

B.E.(Hons), University of Auckland (2011)

Submitted to the Sloan School of Management
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY IN OPERATIONS RESEARCH

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2016

Author .....
Sloan School of Management
May 13, 2016

Certified by .....
Dimitris Bertsimas
Boeing Professor of Operations Research
Thesis Supervisor

Accepted by .....
Patrick Jaillet
Dugald C. Jackson Professor, Department of Electrical Engineering and Computer Science
Co-director, Operations Research Center

Advances in Robust and Adaptive Optimization: Algorithms, Software, and Insights

by

Iain Robert Dunning

Submitted to the Sloan School of Management
on May 13, 2016, in partial fulfillment of the
requirements for the degree of
DOCTOR OF PHILOSOPHY IN OPERATIONS RESEARCH

Abstract

Optimization in the presence of uncertainty is at the heart of operations research. There are many approaches to modeling the nature of this uncertainty, but this thesis focuses on developing new algorithms, software, and insights for an approach that has risen in popularity over the last 15 years: robust optimization (RO), and its extension to decision making across time, adaptive optimization (AO).

In the first chapter, we perform a computational study of two approaches for solving RO problems: "reformulation" and "cutting planes". Our results provide useful evidence for what types of problems each method excels in.

In the second chapter, we present and analyze a new algorithm for multistage AO problems with both integer and continuous recourse decisions. The algorithm operates by iteratively partitioning the problem's uncertainty set, using the approximate solution at each iteration. We show that it quickly produces high-quality solutions.

In the third chapter, we propose an AO approach to a general version of the process flexibility design problem, whereby we must decide which factories produce which products. We demonstrate significant savings for the price of flexibility versus simple but popular designs in the literature.

In the fourth chapter, we describe computationally practical methods for solving problems with "relative" RO objective functions. We use combinations of absolute and relative worst-case objective functions to find "Pareto-efficient" solutions that combine aspects of both. We demonstrate through three in-depth case studies that these solutions are intuitive and perform well in simulation.

In the fifth chapter, we describe JuMPeR, a software package for modeling RO and AO problems that builds on the JuMP modeling language. It supports many features including automatic reformulation, cutting plane generation, linear decision rules, and general data-driven uncertainty sets.

Thesis Supervisor: Dimitris Bertsimas
Title: Boeing Professor of Operations Research

Acknowledgments

I would like to acknowledge and thank my supervisor Dimitris Bertsimas, who has been both a friend and mentor to me during my time at the ORC. I'll always appreciate his patience during the early years of the degree, and his encouragement as I started to hit my stride. I am grateful to my doctoral committee members Juan Pablo Vielma and Vivek Farias, as well as Patrick Jaillet for serving on my general examination committee and co-advising me during my first two years. I'm also grateful to Laura Rose and Andrew Carvalho for all their help over the years.

I must especially thank Miles Lubin for being a constant friend throughout the last four years, and an outstanding collaborator. I'd also like to thank Joey Huchette for joining us on our Julia-powered optimization mission, and all the Julia contributors at both MIT and around the world.

I'd like to thank everyone in my cohort for the conversations, the parties, the studying, and the support. I'd especially like to acknowledge John Silberholz, who has been an excellent friend and collaborator. I'd also like to acknowledge everyone who I worked with on the IAP software tools class, as well as all my co-TAs throughout the years. Special mention must go to Martin Copenhaver, Jerry Kung, and Yee Sian Ng, for all the conversations about life, the universe, and everything.

Finally, I'd like to acknowledge my family for their support not only during my time at MIT, but over my whole life – I would not be where or what I am today without them. I'd like to thank my friends David and Alec who have been supporting me for almost 15 years now – looking forward to many more. Last, but not least, I'd like to acknowledge my wife Kelly, who changed my life and makes it all worthwhile.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Optimization in the presence of uncertainty is at the heart of operations research. The scope of these optimization problems is broad: from deciding how much of each crop to plant when future market prices are uncertain, to whether or not a coal-fired power plant should be run tomorrow given uncertainty in wind and solar power generation, to how much inventory should be ordered for future delivery when there is uncertainty in demand. Despite the importance of these problems, and the amount of effort dedicated to their solution, these optimization-under-uncertainty problems still represent a great challenge to solve in a variety of applications. George Dantzig, one of the fathers of operations research, wrote the following in an article in 1991:

> "In retrospect, it is interesting to note that the original problem that started my research is still outstanding – namely the problem of planning or scheduling dynamically over time, particularly planning dynamically under uncertainty. If such a problem could be successfully solved it could eventually through better planning contribute to the well-being and stability of the world." [Dantzig, 1991]

This thesis does not claim to completely solve these problems, but instead provides a greater understanding of the relative merits of different solution techniques, presents new methods for some difficult classes of problems, explores their application to problems of practical interest, and offers computer modeling tools to express them.

When addressing uncertainty in optimization problems, a decision maker must consider several things, including what information is available about the uncertain parameters, what computational resources can be applied, and what are the risk preferences for this situation. Examples of different types of information that may be available include past realizations of the uncertain parameters, a provided probability distribution (possibly fitted to data, or perhaps from expert judgement), or a set of scenarios. The amount of computational resources available dictates, for a problem of a given size, what methods are available to use. For example, dynamic programming techniques can solve a wide variety of optimization-under-uncertainty problems "exactly", but in general scale very poorly to larger problems (the "curse of dimensionality"). Finally, the risk preferences of a decision maker guide the types of objective functions and constraints we should use. For example, we may care mostly about the performance of a financial portfolio in expectation, or we may want to construct a portfolio that guarantees that it performs no worse than a certain factor relative to a benchmark. In an engineering application we may need to ensure that a constraint holds for all realizations of the parameters up to a certain limit, whereas in a power generation context we may only require that a constraint hold with a certain probability given a distribution for the uncertain parameters.

The temporal nature of the decisions may also be important to consider. If we are designing a structure that must not fail for a given set of realizations of the uncertain parameters, then a decision is made before the uncertainty is realized and the situation is resolved – the structure holds, or it does not. However, in many cases of interest in the operations research literature there are often (a sequence of) recourse decisions to be made. For example, we must decide how much of each crop to plant "here-and-now", at the start of the season and before we know the market prices at the end of the growing season. Then, at the end of the season we make a "wait-and-see" decision of how much to sell with full knowledge of the market prices. In the case of controlling inventory for a retailer, we can model the ordering levels as a series of these "wait-and-see" decisions. Consider a setting where we make end-of-week orders: at the end of the first week, a "here-and-now" order must be made,

before demand for the next week is known. At the end of the next week, we now know the demand and make a "wait-and-see" decision – however, we still don't know the demand for the weeks after that. This sequence of realizations and decisions is known as a multistage decision problem, and modeling that the decision at week n is made with the benefit of knowing the uncertain parameters between now and week n with certainty can dramatically increase the difficulty of these problems.

In this thesis, we focus on the *robust optimization* (RO) perspective of optimization-under-uncertainty. In Section 1.1, we provide a brief overview of what RO is, and how RO problems can be solved. In Section 1.2, we provide a similar overview of how RO has been applied to multistage decision problems, which we will refer to as adaptive RO (ARO) or simply adaptive optimization (AO). Finally, in Section 1.3, we provide an overview of the contributions of the following five chapters to the fields of robust and adaptive optimization.

1.1 Overview of Robust Optimization

While a comprehensive history of RO is beyond the scope of this introduction, we provide a general introduction to some of the key concepts here which are reinforced in the introductions to the individual chapters. For a broader overview, including key theoretical results and a summary of applications, we recommend the survey by Bertsimas et al. [2011a] and the book by Ben-Tal et al. [2009].

Robust optimization has taken a variety of forms over the past 30 years, but the key defining properties are that uncertain parameters in an optimization problem are modeled as belonging to an *uncertainty set*, and that we evaluate the feasibility and value of solutions with respect to the worst-case realization of the parameters over the uncertainty set. In the 1990s, much of the focus was on uncertainty in the objective function. The book by Kouvelis and Yu [1997] describes much of this work, which is primarily interested in combinatorial optimization problems. If the feasible set of solutions to a problem is $\mathcal{X}$, the uncertain parameters $\boldsymbol{\xi}$ belong to an uncertainty set Ξ, the objective function is $f(\boldsymbol{\xi}, \mathbf{x})$, and $f^*(\boldsymbol{\xi})$ is the objective function value you could

obtain with *a priori* knowledge of $\boldsymbol{\xi}$, then the three variants on RO that Kouvelis and Yu [1997] consider are as follows:

- **Absolute robust**: $\max_{\mathbf{x}\in\mathcal{X}} \min_{\boldsymbol{\xi}\in\Xi} f(\boldsymbol{\xi}, \mathbf{x})$
- **Robust deviation**: $\max_{\mathbf{x}\in\mathcal{X}} \min_{\boldsymbol{\xi}\in\Xi} f(\boldsymbol{\xi}, \mathbf{x}) - f^*(\boldsymbol{\xi})$
- **Relative robust**: $\max_{\mathbf{x}\in\mathcal{X}} \min_{\boldsymbol{\xi}\in\Xi} \frac{f(\boldsymbol{\xi},\mathbf{x})}{f^*(\boldsymbol{\xi})}$

The first, absolute robust, guarantees that we will do no worse than a certain absolute level of performance (e.g., "the portfolio return will not be worse than -2%"). The second, robust deviation, is also known as **robust regret** and attempts to incorporate that in many settings there exist values of the uncertain parameters for which no solution would have done well. The best any solution can achieve by this objective function is to have no (zero) regret, or colloquially, never leave anything on the table. The third, relative robust, is also known as **competitive ratio** and is very similar in spirit to the second. It guarantees our performance relative to some optimal benchmark: e.g., "the portfolio returns will not be worse than 90% of what we could have achieved". The best any solution can achieve is a ratio of one, and although a solution that achieves this will also have zero regret, it is not the case that solution with minimal regret always has the best competitive ratio. Much of the work from this period used very simple uncertainty sets, often a "box" or "interval" set where each uncertain parameter was constrained to an interval. In Chapter 2 and Chapter 3 we focus on the absolute robust objective function, but we consider these relative notions in an application in Chapter 4, and explore in detail how to solve a wide variety of problems with these objective functions in Chapter 5.

In the late 1990s and early 2000s, a different line of research developed that was focused on absolute robustness, not only for objective functions but also for constraints. The general problem considered is

$$\begin{aligned} &\max_{\mathbf{x}\in\mathcal{X}} \min_{\boldsymbol{\xi}\in\Xi} f(\boldsymbol{\xi}, \mathbf{x}) \\ &\text{subject to } g_i(\boldsymbol{\xi}, \mathbf{x}) \leq 0 \qquad \forall i \in \mathcal{I}, \boldsymbol{\xi} \in \Xi, \end{aligned} \tag{1.1}$$

where $\mathcal{X}$ captures deterministic constraints, f is the objective function, g_i are constraints, $\mathcal{I}$ is an index set, and Ξ is the uncertainty set. Assuming that Ξ is not a finite set of discrete scenarios, which is uncommon, then this is a problem with an infinite number of constraints. We note that we can simplify the problem somewhat by moving the objective function into a constraint, and thus need only consider the treatment of uncertain constraints, i.e.,

$$\begin{aligned} &\max_{\mathbf{x}\in\mathcal{X},z} \; z \\ \text{subject to } & z \leq f(\boldsymbol{\xi}, \mathbf{x}) \qquad \forall \boldsymbol{\xi} \in \Xi \\ & g_i(\boldsymbol{\xi}, \mathbf{x}) \leq 0 \qquad \forall i \in \mathcal{I}, \boldsymbol{\xi} \in \Xi, \end{aligned} \tag{1.2}$$

Ben-Tal and Nemirovski [1999] considered this problem for the case where f and g are biaffine functions of $\mathbf{x}$ and $\boldsymbol{\xi}$, or in other words, a linear optimization (LO) problem where the coefficients and bounds on the constraints may be uncertain parameters or affine functions of uncertain parameters. They show that, in the case where Ξ is an ellipse, the RO problem (1.2) can be *reformulated* to a problem with a finite number of constraints – although the resulting problem is a second-order cone optimization (SOCO) problem. This result doesn't depend on $\mathbf{x}$ being continuous, so a mixed-integer LO (MILO) problem can also be reformulated to a mixed-integer SOCO (MISOCO) problem. However, at the time this work was done, solvers for MISOCO problems were not generally available and were not considered to be computationally practical. This problem was revisited by Bertsimas and Sim [2004], who considered a polyhedral uncertainty set instead of an ellipsoidal set. Their "budget" uncertainty set, which is used throughout this thesis, allows at most Γ uncertain parameters to deviate from their "nominal" values by a per-parameter "deviation". A key benefit over the ellipsoidal set is that the resulting reformulation preserves linearity: a robust LO problem is reformulated as a LO problem, and a robust MILO problem is reformulated as a MILO problem.

Both these approaches use the theory of duality for convex optimization. We now briefly present the key idea of reformulation for RO, and simultaneously introduce an

alternative approach: *cutting planes*. Consider an uncertain constraint

$$\boldsymbol{\xi}^T \mathbf{x} \leq b \qquad \forall \boldsymbol{\xi} \in \Xi, \tag{1.3}$$

with polyhedral uncertainty set $\Xi = \{\boldsymbol{\xi} \mid \mathbf{F}\boldsymbol{\xi} \leq \mathbf{g}\}$. Suppose we have a $\hat{\mathbf{x}}$ that is feasible with respect to all other constraints, but not necessarily (1.3). As we cannot simply consider the infinite number of constraints on $\mathbf{x}$ here simultaneously, we will instead attempt to find a single $\hat{\boldsymbol{\xi}}$ such that the resulting constraint is violated (i.e., $\hat{\boldsymbol{\xi}}^T \hat{\mathbf{x}} > b$). As Ξ is a polyhedron, we can do so by solving the following LO problem:

$$\begin{aligned} &\max_{\boldsymbol{\xi}} \ \hat{\mathbf{x}}^T \boldsymbol{\xi} \\ &\text{subject to } \mathbf{F}\boldsymbol{\xi} \leq \mathbf{g}. \end{aligned} \tag{1.4}$$

The process of finding a solution $\hat{\mathbf{x}}$, adding new constraints by solving problem (1.4), then resolving for a new $\hat{\mathbf{x}}$, is known as the cutting plane method. It is discussed at length in Chapter 2, and the general idea is used throughout this thesis. Returning to reformulation, observe that we can use LO duality (see, e.g., Bertsimas and Tsitsiklis [1997]) to formulate an equivalent dual problem

$$\begin{aligned} &\min_{\boldsymbol{\pi} \geq \mathbf{0}} \ \mathbf{g}^T \boldsymbol{\pi} \\ &\text{subject to } \mathbf{F}^T \boldsymbol{\pi} = \hat{\mathbf{x}}. \end{aligned} \tag{1.5}$$

We know that, for any feasible $\hat{\boldsymbol{\pi}}$ and optimal solution $\hat{\boldsymbol{\xi}}$ to the primal problem (1.4), the inequality $\mathbf{g}^T \hat{\boldsymbol{\pi}} \geq \hat{\mathbf{x}}^T \hat{\boldsymbol{\xi}}$ holds. Thus, if we replace our original uncertain constraint with the auxiliary constraints and variables

$$\begin{aligned} \mathbf{g}^T \boldsymbol{\pi} &\leq b \\ \mathbf{F}^T \boldsymbol{\pi} &= \mathbf{x} \\ \boldsymbol{\pi} &\geq \mathbf{0}, \end{aligned} \tag{1.6}$$

then we know that any feasible solution to this system of equations is robust, as $\mathbf{g}^T \boldsymbol{\pi}$

is at least the original left-hand-side's worst-case value.

1.2 Overview of Adaptive Optimization

In Section 1.1, we focused on "single-stage" RO problems – that is, problems with no notion of sequential decision making. To motivate the difficulties that arise when considering multistage problems, consider a two-stage problem in which we first make a decision $\mathbf{x}$, then uncertain parameters $\boldsymbol{\xi}$ are revealed, and then we make another decision $\mathbf{y}$, e.g.,

$$\max_{\mathbf{x}} \min_{\boldsymbol{\xi}} \max_{\mathbf{y}} f(\boldsymbol{\xi}, \mathbf{x}, \mathbf{y}). \tag{1.7}$$

We can rewrite this problem in a similar form to the RO problem (1.1):

$$\max_{\mathbf{x}, \mathbf{y}(\boldsymbol{\xi})} \min_{\boldsymbol{\xi}} f(\boldsymbol{\xi}, \mathbf{x}, \mathbf{y}(\boldsymbol{\xi})), \tag{1.8}$$

where we can interpret $\max_{\mathbf{y}(\boldsymbol{\xi})}$ as optimizing over *policies*, i.e., functions of $\boldsymbol{\xi}$. We refer to problems such as these as *adaptive RO* (ARO) problems, or simply adaptive optimization (AO) problems, as the future decisions *adapt* as uncertain parameters are revealed.

The optimal policy $\mathbf{y}^*(\boldsymbol{\xi})$ may be very complicated (indeed, as we showed it here it is the solution of an optimization problem itself), and thus we might expect that this problem is not theoretically tractable in general. This was shown to be true by Ben-Tal et al. [2004], leading to a series of papers over the last decade attempting to find cases where it is tractable, to propose tractable approximations for the optimal policy, and to understand the behavior of those approximations. The first approximation was proposed in that same paper by Ben-Tal et al. [2004], where instead of a general $\mathbf{y}(\boldsymbol{\xi})$ we use an *affine approximation* (or *linear decision rule*):

$$y_j(\boldsymbol{\xi}) := \bar{y}_0^j + \boldsymbol{\xi}^T \bar{\mathbf{y}}^j \boldsymbol{\xi}, \tag{1.9}$$

where $\bar{y}_0^j$ and $\bar{\mathbf{y}}^j$ are auxiliary variables that define the affine policy. This is obviously a

massive simplification, but has shown to be very effective across a variety of settings – partly because its linearity keeps otherwise linear AO problems linear, but also because it has some theoretical backing (e.g., Bertsimas et al. [2010], Bertsimas and Goyal [2012]). A key failing of this approximation is its failure to accommodate discrete variables, which must necessarily be piecewise constant, not affine. Another approximation that has seen some success, and addresses the case of discrete variables, is *finite adaptability* [Bertsimas and Caramanis, 2010]. In finite adaptability, we first partition an uncertainty set Ξ, e.g. $\Xi = \Xi_1 \cup \Xi_2 \cup \cdots \cup \Xi_K$. We then define the policy to be a piecewise constant policy, with a separate decision variable for each partition:

$$\mathbf{y}(\boldsymbol{\xi}) = \begin{cases} \bar{\mathbf{y}}^1, & \forall \boldsymbol{\xi} \in \Xi_1, \\ \vdots & \\ \bar{\mathbf{y}}^K, & \forall \boldsymbol{\xi} \in \Xi_K. \end{cases} \tag{1.10}$$

The key issue with this approach is how to decide the partitioning, and optimizing for the best K partitions is a difficult optimization problem even when $K = 2$. In Chapter 3, we propose a method that produces those partitions, applies the finite adaptability approach to multistage optimization, and incorporates affine adaptability to produce piecewise affine policies for continuous decisions. We also provide a more comprehensive survey in Section 3.1 of the various approaches to AO, including other extensions to affine and finite adaptability.

1.3 Overview of Thesis

This thesis consists of five chapters (apart from this introduction chapter). These are all based on papers submitted to, or accepted by, peer-reviewed journals, except Chapter 6 which describes an open-source software package for RO modeling.

Chapter 2: Reformulation versus Cutting-planes for Robust Optimization

This chapter is based on Bertsimas et al. [2015b], which is published in *Computational Management Science*.

As briefly descried above in Section 1.1, there are two common ways to solve RO problems: reformulation (using duality), and using a cutting plane method. However, there is little guidance in the literature as to when to use each, and how to apply cutting planes to RO problems with integer variables. In this chapter, we precisely describe the cutting plane method for two popular uncertainty sets, including a consideration of how to integrate it into a branch-and-bound procedure. We then conduct a comprehensive computational experiment on a wide instance library, revealing that it is not clear that one method is uniformly superior to the other. Finally, we perform experiments that combine both methods, demonstrating that doing so can produce impressive performance. Cutting plane methods are used in many experiments in subsequent chapters, and the software described in Chapter 6 enables users to easily switch between cutting planes and reformulation.

Chapter 3: Multistage Robust Mixed Integer Optimization with Adaptive Partitions

This chapter is based on Bertsimas and Dunning [2016a], which is accepted for publication in *Operations Research*.

Adaptive robust optimization is difficult, both in terms of computational practicality and theoretical intractability. Despite this, practical approaches have been developed for many cases of interest. The most successful of these is the use of affine policies, or linear decision rules, where the value of future wait-and-see decisions is an affine function of uncertainty revealed previously. However, this approach fails when the wait-and-see decisions are discrete (integer, binary) decisions. In this chapter, we present a new approach to adaptive optimization that leads to piecewise constant policies for integer decisions, and piecewise affine policies for continuous decisions. It does so by iteratively partitioning the uncertainty set, and associating a different

decision with each partition. The solution of the problem with one set of partitions is then used to guide a refinement of the partitions – a procedure that can be repeated until the rate of improvement decreases, or the gap to the bound we describe falls below a tolerance. We provide theoretical motivation for this method, and characterize both its convergence properties and the growth in the number of partitions. Using these insights we propose and evaluate enhancements to the method such as warm starts and smarter partition creation. We describe in detail how to apply finite adaptability to multistage AMIO problems to appropriately address nonanticipativity restrictions. Finally we demonstrate in computational experiments using the JuMPeR software package (Chapter 6) that the method can provide substantial improvements over a non-adaptive solution and existing methods for problems described in the literature. In particular, we find that our method produces high-quality solutions versus the amount of computational effort, even as the problem scales in both number of time stages and in the number of decision variables.

Chapter 4: The Price of Flexibility

This chapter is based on Bertsimas et al. [2015a], which has been submitted to *Operations Research* for review.

Process flexibility is a popular operations strategy that has been employed in many industries to help firms respond to uncertainty in product demand. However, additional flexibility comes at a cost that firms must balance against the reduction of risk it can provide. In this chapter, we reduce the price of flexibility by taking an optimization approach to the process flexibility design problem. Unlike many approaches previously discussed in the literature, we consider systems that may have nonhomogenous parameters and unbalanced capacity and demand. We formulate the problem as a robust adaptive optimization model, and propose a computationally tractable method for solving this model using standard integer optimization software. To further reduce the price we consider Pareto-efficient and relaxed Pareto-efficient robust solutions, and show that they correspond to flexibility designs that perform well in both worst-case and average-case demand scenarios. We demonstrate through

simulation experiments that our method can find flexible designs that reduce the price of flexibility by approximately 30% or more versus common flexibility designs, with a negligible impact on revenues and thus higher profits. We use JuMPeR (Chapter 6) for solving these problems.

Chapter 5: Relative Robust and Adaptive Optimization

This chapter is based on Bertsimas and Dunning [2016b], which has been submitted to *INFORMS Journal on Computing* for review.

Early approaches to RO focused on "relative" worst-case objective functions, where the value of a solution is measured versus the best-possible solution over an "uncertainty set" of scenarios. However, over the past ten years the focus has primarily been on "absolute" worst-case objective functions, which have generally been considered to be more tractable. In this chapter, we demonstrate that for many problems of interest, including some adaptive robust optimization problems, that considering "relative" objective functions does not significantly increase the computational cost over absolute objective functions. We use combinations of absolute and relative worst-case objective functions to find "Pareto-efficient" solutions that combine aspects of both, which suggests an approach to distinguish between otherwise very similar robust solutions. We provide reformulation and cutting plane approaches for these problems and demonstrate their efficacy with experiments on minimum-cost flow, inventory control, and facility location problems. These case studies show that solutions corresponding to relative objective functions may be a better match for a decision maker's risk preferences than absolute.

Chapter 6: JuMPeR: Algebraic Modeling for Robust and Adaptive Optimization

In the final chapter, we present "JuMPeR", an algebraic modeling language for RO and ARO implemented as an extension to the JuMP modeling language. JuMPeR makes implementing and experimenting with ARO models fast and simple, providing features such as automatic reformulation, cutting plane generation, linear decision

rules, and complex uncertainty sets. We first present the overall design and structure of JuMPeR, before diving into some key implementation details. We then present three in-depth case studies, before finishing with a brief comparison with related tools.

Chapter 2

Reformulation versus Cutting-planes for Robust Optimization

2.1 Introduction

Robust optimization (RO) has emerged as both an intuitive and computationally tractable method to address the natural question of how to handle uncertainty in optimization problems. In RO the uncertain parameters in a problem are modeled as belonging to an *uncertainty set* (see, e.g. Ben-Tal et al. [2009] or Bertsimas et al. [2011a] for a survey). A wide variety of uncertainty sets have been described in the literature, but polyhedral and ellipsoidal sets are by far the most popular types. Constraints with uncertain parameters must be feasible for all values of the uncertain parameters in the uncertainty set, and correspondingly the objective value is taken to be the worst case value over all realizations of the uncertain parameters. For this reason RO problems, in their original statement, typically have an infinite number of constraints and cannot be solved directly. The most common approach to date for solving them in the literature is to use duality theory to *reformulate* them as a deterministic optimization problems that may have additional variables, constraints, and even change problem class (e.g. a robust linear optimization (RLO) problem with an ellipsoidal uncertainty set becomes a second-order cone problem (SOCP) [Ben-Tal and Nemirovski, 1999]). Another method, used less frequently, is an iterative *cutting-*

plane method (e.g. Mutapcic and Boyd [2009]) that repeatedly solves a relaxed form of the RO problem with a finite subset of the constraints, checks whether any constraints would be violated for some value of the uncertain parameters, adds them if so, and re-solves until no violated constraint exists.

There is little guidance in the literature about which method should be used for any given RO problem instance. We are aware of one computational study by Fischetti and Monaci [2012] that directly compares the two methods for RLO and robust mixed-integer optimization (RMIO) problems with a polyhedral uncertainty set and finds that the cutting-plane method is superior for RLO and that reformulations are superior for RMIO. They do not attempt to indicate whether this result is statistically significant, and do not experiment with different variants of the cutting-plane method.

Rather than just consider the question of which method is superior, we consider a different question in this chapter that we feel is more relevant for both practitioners and researchers, and acknowledges the ever-increasing popularity of parallel computing. We draw inspiration from modern solvers such as Gurobi [Gurobi Optimization Inc., 2016] which, by default, simultaneously solve an LO problem with both the simplex method and an interior point method and return a solution whenever one of the methods terminates - effectively making the runtime the minimum of the two methods' individual runtimes. This can naturally be extended to the RO case: we can solve an RO problem with both the reformulation method and the cutting plane method simultaneously, and the runtime is defined by the first method to finish. We do not implement in this chapter a general-purpose tool which does this simultaneous solve automatically, but the capability to do so would be a natural extension of the capabilities of an RO modeling system (such as the one described in Chapter 6).

The structure and contributions of this chapter are as follows:

- We provide a precise description of the application of the cutting-plane method to RMIO problems, including an identification of the added complexities over applying the method to RLO problems, and the variants that should be considered by those seeking to implement the method.

- We replicate the findings of Fischetti and Monaci [2012] over an enlarged collection of instances, and extend them by evaluating the performance of the reformulation and cutting-plane methods for RO problems with ellipsoidal uncertainty sets. We use more appropriate metrics and employ statistical techniques to more precisely characterize the differences in performance between methods

- We enhance the performance of the cutting-plane method for RMIO problems by designing rules on when new cutting planes should be added, and evaluate a new heuristic to run alongside the MIO solver to improve runtimes.

- Finally we describe and evaluate our proposed hybrid method which uses both methods in parallel and stops when either method finishes. We discuss implementation considerations and possible extensions.

Structure of the chapter. In Section 2.2, we detail the precise RO problem and uncertainty sets we evaluate. We discuss the details of the different cutting-plane algorithms and the possible variations in when new generated constraints should be added. We provide both the intuition and details for a new heuristic for RMIO problems. Section 2.3 explains the experimental setup, sources of data and the combinations of instance, parameters, and methods we evaluated. We analyze the results in depth to compare the performance of the two methods and their variants on both RLO and RMIO problems. Section 2.4 investigates the proposed hybrid method's performance relative to the individual methods and its implementation. Finally Section 2.5 summarizes the results.

2.2 Problem and Method Descriptions

We will consider the following general linear RO problem

$$\begin{aligned} \min \quad & c^T x \\ \text{subject to} \quad & \tilde{a}_1^T x \leq b_1 \quad \forall \tilde{a}_1 \in U_1 \\ & \vdots \\ & \tilde{a}_m^T x \leq b_m \quad \forall \tilde{a}_m \in U_m \\ & x \in \mathbb{R}^{n_1} \times \mathbb{Z}^{n_2}, \end{aligned}$$

where n_1 and n_2 are nonnegative integers, $b_1, \ldots, b_m$ are given scalars, c is a given vector, and $U_1, \ldots, U_m$ are the uncertainty sets for each constraint. We will denote uncertain values using a tilde, e.g. $\tilde{a}$. In row i a subset J_i of the coefficients are said to be subject to uncertainty, and all uncertain coefficients have a *nominal* value a_{ij} around which they may vary. We define the *nominal problem* to be the optimization problem identical to the above with the exception that all coefficients are at their nominal values with no uncertainty, that is $\tilde{a}_{ij} = a_{ij}$ for all i and j. Later in this section we will detail the properties of the two types of uncertainty sets we consider in this chapter - the polyhedral set introduced in Bertsimas and Sim [2004] and the ellipsoidal set introduced in Ben-Tal and Nemirovski [1999]. These sets are both simple to describe but have substantially different deterministic reformulations and algorithms to calculate a new cutting-plane.

Before presenting the specific details of the polyhedral and ellipsoidal uncertainty sets (in Section 2.2.3 and Section 2.2.4 respectively) we will detail the cutting-plane methods we considered for both RLO problems and RMIO problems. While similar in spirit, there is a substantial difference in the details of their implementation. The method for RLO problems (Section 2.2.1) is fairly simple and comparable to other constraint generation methods. The method for RMIO problems (Section 2.2.2) requires consideration of exactly when in the integer optimization solution process cuts should be added.

2.2.1 Cutting-plane method for RLO

The idea behind the cutting-plane method for RLO is similar to other row generation methods such as Bender's Decomposition. While there is a constraint for every $\tilde{a}$ in the relevant uncertainty set, only a small subset of these constraints is binding for a robust optimal solution. This suggests that only generating constraints as they are needed to ensure robustness of the solution would be an efficient technique. Given this motivation, the algorithm for RLO problems is as follows:

1. Initialize the *master problem* to be the nominal problem, that is the problem described in Eq. (2.1) where all $\tilde{a}_{ij}$ are replaced with their nominal values a_{ij}.
2. Solve the master problem, obtaining a solution x^*.
3. For each uncertain row i (that is, rows i such that $J_i \neq \emptyset$)
   (a) Compute $\bar{a} = \arg\max_{\tilde{a} \in U_i} \tilde{a}^T x^*$.
   (b) If $\bar{a}^T x^* > b_i + \epsilon$, add the constraint $\bar{a}^T x \leq b_i$ to the master problem.
4. If no constraints were added then we declare that x^* is the optimal robust solution to the RO and terminate. If any constraints were added we return to Step 2.

The computational practicality of this method relies on the fact that while adding constraints will make the current master problem solution infeasible, we are able to hot-start the optimization (in Step 2) by using the dual simplex method. As a result almost all commercial and open-source LO solvers can be used to solve the master problem. We will defer discussing guarantees of the termination of this algorithm to Sections 2.2.3 and 2.2.4.

The constraint feasibility tolerance ϵ is a parameter that may be varied depending on the needs of the application and the solver used, but should generally be an appropriately small value such as 10^{-6}. We implemented the cutting-plane generation and related logic in C++ and used the commercial solver Gurobi 5.6 [Gurobi Optimization Inc., 2016] to solve the master problem.

Note that a number of improvements to the classical cutting-plane method have been developed, including bundle methods and the analytic center cutting-plane method (ACCPM) [Briant et al., 2008]. These methods, in general, aim to "stabilize" the sequence of solutions x^*, which can lead to improved theoretical and practical convergence rates. These techniques have been investigated in the context of stochastic programming [Zverovich et al., 2012] but to our knowledge not in the context of RO. Nevertheless, a preliminary implementation of a proximal bundle method did not yield significant improvements in computation time in our experiments, and the complexity of integrating these techniques within branch and bound (for RMIO) discouraged further investigation.

2.2.2 Cutting-plane method for RMIO

The cutting-plane method for RMIO is more complicated than the RLO case, although the principle of only adding constraints as needed is the same. The primary difference is that detailed consideration must be given to the most appropriate time to add new cuts, and that whatever timing is selected guarantees that the final solution is indeed robust.

Perhaps the most obvious place to add cuts is at the root LO relaxation in the branch-and-bound tree. However, it is not sufficient to apply the RLO cutting-plane method to just the root relaxation as the optimal robust integer solution may be affected by constraints that are not active at the fractional solution of the relaxation. Consider this simple example that demonstrates this behavior:

$$\begin{aligned} \max \quad & x \\ \text{subject to} \quad & x \leq 1.5 \\ & u \cdot x \geq 1 \quad \forall u \in [0.9, 1.1] \\ & x \text{ integer} \end{aligned}$$

The relaxed master problem for this RMIO consists of two constraints, $x \leq 1.5$ and $x \geq 1$ (taking the nominal value of u to be 1), leading to the trivial solution $x^*_{relax} = 1.5$ when we relax integrality. This solution is feasible to all constraints, so no new cutting planes are generated at the root. If we now branch once on the value of x (i.e. $x \leq 1$ or $x \geq 2$) we will obtain the integer solution $x = 1$ which is not feasible with respect to the uncertain constraint - in fact, there is no feasible integer solution to this problem. Although making the root relaxation feasible to the uncertain constraints doesn't guarantee the feasibility of subsequent integer solutions, the addition of constraints at the root may guide the search along more favorable lines - we will analyze this effect in our computational results.

Another possibility is to check for and add new constraints at every node in the branch-and-bound tree, ensuring that the fractional solution at each node is feasible with respect to the uncertain constraints before further branching. This appears to have been the approach taken in Fischetti and Monaci [2012]. While this will ensure that any integer solution produced by branching will be feasible to the uncertain constraints, there is likely to be a heavy computational cost as we add cuts that are *unnecessary*, in the sense that they are being added to nodes on a branch of the search tree that may not produce an optimal integer solution or improve the bound.

A third possibility is to only check for and add new constraints when we obtain candidate integer-feasible solutions. These constraints are often described as *lazy constraints* as they are not explicitly provided to the solver before they are needed. Instead they are implemented by providing a *callback* to the solver that is responsible for checking the feasibility of all integer solutions and reporting any violated constraints. The solver then discards the candidate integer solution (if it is not feasible) and adds the violated constraints to the active nodes of the branch-and-bound tree. Like adding constraints at each node, using lazy constraints is sufficient to guarantee that the final integer solution is feasible with respect to the uncertain constraints, but given the relatively small number of integer solutions for many MIO problems we decided that it would be a far more efficient method.

One option that is not practical is to completely solve the master problem to

provable optimality before adding new constraints. If we wait until the end we are most likely to be adding useful constraints but we pay a huge computational cost as we essentially solve the master problem from scratch multiple times; unlike the case of RLO, no hot-start is possible as even the previous integer solution cannot be provided as an incumbent solution. Lazy constraints are a more appropriate technique as they are added before the solve is completed, allowing us to retain the bound corresponding to the fractional solution.

The final cutting-plane method we implemented is as follows:

1. As for RLO, initialize the *master problem* to be the nominal problem, that is the problem described in Eq. (2.1) where all $\tilde{a}_{ij}$ are replaced with their nominal values a_{ij}.

2. Apply the RLO cutting-plane method to the root fractional relaxation of the RMIO master problem until the fractional solution is feasible with respect to all uncertain constraints. Note that while we may generate unnecessary cuts with respect to the final integer solution, we may also gain by requiring less cuts later on when we have progressed further down the tree.

3. The MIO solver now begins the branch-and-bound process to solve the master problem.

    - Whenever an integer solution is found we will check all uncertain constraints to see if any are violated by the candidate solution. If any violations are detected we report these new lazy constraints to the MIO solver, which will discard the candidate integer solution. This ensures that only cuts active at integer solutions are added, and that any integer solution will be feasible with respect to the uncertain constraints. If no constraints are violated, the MIO solver will accept the integer solution as its incumbent solution, and the integrality gap will be evaluated relative to this solution.
    - If we do find that the integer solution violates one of the uncertain constraints, we will apply a heuristic that will try to obtain a feasible integer

solution using this candidate integer solution as a starting point. For more details, see Section 2.2.2.

We used Gurobi 5.6 [Gurobi Optimization Inc., 2016] to solve the MIO problem, with all other logic implemented in C++.

A heuristic to repair infeasible candidate integer solutions

The MIO solver will find multiple integer solutions as it attempts to solve the master problem to provable optimality. When it does so we will check whether any new constraints can be generated from the uncertainty sets that would make the candidate integer solution infeasible. If we do find any, we add these lazy constraints to the master problem and declare the found integer solution to be infeasible. The solver will discard the solution and keep on searching for another integer solution.

We hypothesize that in many cases these integer solutions that violated the uncertain constraints can be made feasible with respect to these constraints *solely by varying the value of the continuous variables*. The solver cannot explore this possibility by itself, as it is unaware of the uncertain constraints until we provide them. This may lead to possibly high-quality integer solutions being rejected and only rediscovered later in the search process. To address this issue we developed a heuristic that attempts to *repair* these integer solutions:

1. Our input is an integer solution x^* to the master problem that violates one or more uncertain constraints. Create a duplicate of the master problem, called the *subproblem*.

2. Fix the values of all integer variables in the subproblem to their values in the integer solution x^*. It follows that the subproblem is now equivalent to the master problem for a RLO problem.

3. Using the cutting-plane method for RLO problems to solve the subproblem.

4. If a solution to the subproblem is found, then this solution is a valid integer solution for the original problem that satisfies all constraints, uncertain and

deterministic. Report this solution to the MIO solver so that it may update its incumbent solution and bounds.

5. If no solution can be found for the subproblem then we take no action, and conclude that we must change the values of integer variables in order to obtain a robust integer solution.

We take a number of steps to improve the computational tractability of this heuristic. For example, we keep only one copy of the LO subproblem in memory and use it for all runs of the heuristic. This avoids the overhead of creating multiple copies, and we can use the dual simplex method to efficiently solve the subproblem given a new candidate solution. An additional effect is that we accumulate generated constraints from previous heuristic runs, further reducing the time for new integer solutions to be *repaired* or rejected. Secondly, we can take the lazy constraints we generate for the master problem and apply them to the heuristic subproblem to avoid re-generating them later. Finally all constraints we generate for the heuristic subproblem are valid for the master MIO, so we can add them to the master as lazy constraints to avoid generating integer solutions that will violate uncertain constraints in the first place.

The benefits of the heuristic are not guaranteed and may vary from instance-to-instance. The heuristic adds computational load that can only be made up for by improvements in speed with which we tighten the bound. In particular, by adding constraints back into the MIO master problem we may avoid needlessly generating non-robust integer solutions, but at the price of longer solve times at each node. We will explore these trade-offs between using the heuristic or not in Section 2.3.2.

2.2.3 Polyhedral uncertainty sets

We considered the polyhedral uncertainty set defined in Bertsimas and Sim [2004] with two properties: for a set U_i corresponding to constraint i

- the uncertain coefficients $\tilde{a}_{ij}\ \forall j \in J_i$ lie in the interval $[a_{ij} - \hat{a}_{ij}, a_{ij} + \hat{a}_{ij}]$, where a_{ij} is the *nominal value*, and

- at most Γ uncertain coefficients in the row are allowed to differ from their nominal value.

One may formulate a LO problem to find the coefficients $\tilde{a}_{ij} \in U_i$ that would most violate the constraint $\tilde{a}_i^T x \leq b_i$. In Bertsimas and Sim [2004] the authors demonstrate that duality theory allows us to replace the original uncertain constraint with a finite set of new deterministic constraints and auxiliary variables corresponding to the constraints and variables of the dual of this problem. As detailed in Table 2.1, this reformulation preserves the problem class of the original RO problem - a RLO problem becomes a LO problem, and likewise a RMIO problem becomes a MIO problem.

While we could generate new constraints by solving the aforementioned LO problem, we can exploit the structure to solve the problem more efficiently by applying the following algorithm to each uncertain constraint i:

- Given a current solution x to the master problem, calculate the absolute deviations $z_j = \hat{a}_{ij} |x_j|$ for all $j \in J_i$.

- Sort z_j (all non-negative by definition) from largest to smallest magnitude. The indices J' corresponding to the Γ largest values of z_j will cause the maximum violation of the constraint i if set to their upper or lower bounds.

- If the constraint can be violated by setting those coefficients to their bounds while all other coefficients remain at their nominal values, add this violated constraint.

As we are only interested in a subset of the coefficients of size Γ this algorithm will run in $O(n + \Gamma \log \Gamma)$ time per constraint using an efficient partial sorting algorithm. We also note that many other polyhedral uncertainty sets exist that will require different cutting-plane algorithms, but we consider this set to be representative of many commonly-seen polyhedral sets in the literature. Termination of the cutting-plane algorithm is guaranteed in the case of the polyhedral uncertainty set: cutting-plane generation is equivalent to optimizing a linear function over a polyhedron, and there are only finitely many extreme points of the uncertainty set.

2.2.4 Ellipsoidal uncertainty sets

We evaluated the ellipsoidal uncertainty sets introduced in Ben-Tal and Nemirovski [1999, 2000], which model the coefficients $\tilde{a}_i \in U_i$ as belonging to the ellipse

$$\sqrt{\sum_{j \in J_i} \left(\frac{\tilde{a}_{ij} - a_{ij}}{\hat{a}_{ij}} \right)^2} \leq \Gamma$$

The reformulation method for ellipsoidal sets replaces the uncertain constraints with new second-order cone constraints; thus LO problems become second-order cone problems (SOCPs), and MIO problems become mixed-integer SOCPs (MISOCPs) (Table 2.1).

The algorithm to generate a new deterministic constraint for a given uncertain constraint is equivalent to finding the maximizer of a linear function over a ball, which has a closed-form expression. If we consider the Karush-Kuhn-Tucker conditions for the cutting-plane problem $\max_{\tilde{a} \in U_i} \tilde{a}^T x$ we find that we can efficiently determine a new constraint $\bar{a}$ in $O(n)$ time by evaluating

$$\bar{a}_j^* = \frac{\Gamma}{\left\| \hat{A} x \right\|} \hat{a}_j^2 x_j + a_j$$

where $\hat{A}$ is a diagonal matrix where the diagonal elements are the elements of $\hat{a}$. Note that this closed-form solution applies to generalizations which consider correlation matrices (non-axis aligned ellipses).

An interesting point of difference between the ellipsoidal cutting-plane method and the ellipsoidal reformulation is that the cutting-plane method does not modify the problem's class as no quadratic terms appear in the generated constraints. However, unlike for polyhedral uncertainty sets, finite termination of the cutting-plane method applied to ellipsoidal uncertainty sets is not theoretically guaranteed, which perhaps discourages the implementation of this method in practice. Nevertheless, we will see that this method is indeed practical in many cases.

Table 2.1: Problem class of the deterministic reformulations and cutting-plane method master problems for RLO and RMIO problems, dependent on the choice of uncertainty set. The polyhedral set does not change the problem class. The cutting-plane method master problem remains in the same problem class as the original RO problem regardless of the set selected, but the most efficient algorithm to generate a new cut is dependent on the set.

	Polyhedral set		Ellipsoidal set	
Robust Problem	RLO	RMIO	RLO	RMIO
Reformulation	LP	MIO	SOCP	MISOCP
Cutting-plane master	LP	MIO	LO	MIO
Cut generation	Sorting		Closed-form	

2.3 Computational Benchmarking

Obtaining a large number of diverse RO problems was achieved by taking instances from standard LO and MIO problem libraries and converting them to RO problems. We obtained test problems from four sources:

1. LO problems from NETLIB [Gay, 1985], all of which are relatively easy for modern solvers in their nominal forms.

2. LO problems from Hans Mittelmann's benchmark library [Mittelmann].

3. MIO problems from MIPLIB3 [Bixby et al., 1998].

4. MIPLIB 2010, the current standard MIO benchmark library [Koch et al., 2011]. We selected only the "easy" instances, where the nominal problem is solvable to provable optimality in under an hour.

Uncertain coefficients are identified using a procedure based on the method in Ben-Tal and Nemirovski [2000], that is similar to methods in other papers [Bertsimas and Sim, 2004, Fischetti and Monaci, 2012]):

- A coefficient is *certain* if it is representable as a rational number with denominator between 1 and 100 or a small multiple of 10.

- It is otherwise declared to be *uncertain*, with a 2% deviation allowed from the nominal value: $\hat{a}_{ij} = 0.02\,|a_{ij}|$

- All equality constraints are certain.

After removing all instances which had no uncertain coefficients, we were able to use 183 LO problems and 40 MIO problems.

We compared the reformulation and the cutting-plane methods for RLO (Section 2.3.1) and RMIO (Section 2.3.2). We tried three different values of Γ (1, 3, and 5) to test if the solution methods were sensitive to the amount of robustness. We used single-threaded computation throughout and a time limit of 1,000 seconds was enforced on all instances. All solver settings were left at their defaults unless otherwise stated.

For the case of the polyhedral reformulation for RLO problem we controlled for the choice between interior point ("barrier") and dual simplex as the solution method. In particular, for the interior point method we disabled "crossover" (obtaining an optimal basic feasible solution) as we didn't consider it relevant to the experiment at hand (the solution to the reformulation was not needed for any further use). The choice of solution method is significant in this case as we restricted the solver to one thread in all benchmarks - if multiple threads were available, we could use both simultaneously and take the faster of the two. This implication is explored further in Section 2.4.

We did not experiment with different solver options for the polyhedral set RMIO reformulation but did experiment with a solver option that chooses which algorithm to use to solve the ellipsoidal reformulation (a MISOCP). There are two choices available in Gurobi 5.6 : either a linearized outer-approximation, similar to the cutting plane method we employ, or solving the nonlinear continuous relaxation directly at each node. We solved the reformulation with each algorithm to investigate their relative performance for the RO problems.

We use multiple measures to try to capture different aspects of the relative performance of the various methods and parameters. As the time to solve each problem varies dramatically we used only unitless measures, and avoided measures that are heavily affected by skew like the arithmetic mean employed in Fischetti and Monaci [2012]. We complement these numerical results with the graphical *performance pro-*

files [Dolan and Moré, 2002] which show a more complete picture of the relative runtimes for each method. Where possible we will relate the measures described below to these performance profiles.

The first and most basic measure is the *percentage of problems solved fastest* by each method. This corresponds to the y-axis values in the performance profiles for a performance ratio of 1. This measure is sensitive to small variations in runtime, especially for smaller problems, and does not capture any of the worst-case behaviors of the methods.

The second and third measures are both summary statistics of normalized runtimes. More precisely, for a particular combination of problem class (RLO, RMIO), uncertainty set (polyhedral, ellipsoidal), and Γ we define T_k^M to be the runtime of method M for instance $k \in \{1, \ldots, K\}$, and define T_k^* to be the best runtime for instance k across all methods, that is $T_k^* = \min_M T_k^M$. Then the second measure is, for each method M, to take the *median of* $\left\{\frac{T_k^M}{T_k^*}\right\}$. The median is a natural measure of the central tendency as it avoids any issues with outliers, although it is somewhat lacking in power when more than 50% of the instances take the same value, as is sometimes the case for our data. The third measure, which doesn't have this shortcoming, is to take the *geometric mean* of the same data for each method M, that is to calculate

$$\left(\prod_{k=1}^{K} \frac{T_k^M}{T_k^*}\right)^{-K}. \tag{2.1}$$

We note that the arithmetic mean can give misleading results for ratios [Fleming and Wallace, 1986] and that the geometric mean is considered a suitable alternative.

A natural question is to ask how confident we are in the second and third measures, which we address with confidence intervals. To do so, we must frame our collection of test problems as samples from an infinite population of possible optimization problems. We are sampling neither randomly nor independently from this population, so these intervals must be treated with some caution. As the distribution of these measures is not normal, and the sample sizes are not particularly large, we estimate 95% confidence intervals using bootstrapping (e.g. Efron and Tibshirani [1994]) which

Table 2.2: Summary of RLO benchmark results for the polyhedral uncertainty set for $\Gamma = 5$. *% Best* is the fraction of problems for which a method had the lowest run time. *Median* and *Geom. Mean* are the median and geometric mean respectively of the run times for each instance normalized by the best time across methods. The percentage best for the pseudo-method *Reformulation (Best of Both)* is calculated as the sum of the two reformulation percentages. The two entries marked with an asterisk represent a separate comparison made only between those two methods in isolation from the alternatives.

	% Best	Median (95% CI)	Geom. Mean (95% CI)
Cutting-plane	75.5	1.00 (1.00, 1.00)	1.52 (1.31, 1.89)
Reformulation (Barrier)	15.8	1.57 (1.40, 1.77)	1.88 (1.72, 2.10)
Reformulation (Dual Simplex)	8.7	1.48 (1.40, 1.54)	1.69 (1.57, 1.85)
Reformulation (Best of Both)	24.5	1.27 (1.20, 1.54)	1.39 (1.32, 1.49)
Cutting-plane *	83.7	1.00 (1.00, 1.00)	1.39 (1.22, 1.68)
Reformulation (Dual Simplex) *	16.3	1.42 (1.39, 1.54)	1.54 (1.46, 1.65)

Table 2.3: Summary of RLO benchmark results for the ellipsoidal uncertainty set for $\Gamma = 5$. *% Best* is the fraction of problems for which a method had the lowest run time. *Median* and *Geom. Mean* are the median and geometric mean respectively of the run times for each instance normalized by the best time across methods.

	% Best	Median (95% CI)	Geom. Mean (95% CI)
Cutting-plane	44.8	1.06 (1.00, 1.12)	1.76 (1.51, 2.18)
Reformulation	55.2	1.00 (1.00, 1.00)	1.38 (1.27, 1.57)

avoids some of the assumptions that would otherwise have to be made to get useful results.

Finally we found that runtimes were relatively invariant to the choice of Γ. Any differences were less than the general minor variations caused by running the benchmarks multiple times. For this reason we present results only for $\Gamma = 5$ throughout the chapter. Additionally although we are not interested in comparing in any way the merits of polyhedral versus ellipsoidal sets, we wish to note that the relative "protection" level implied by a given Γ differs for the polyhedral and ellipsoidal sets.

2.3.1 Results for RLO

We found that the cutting-plane method was the fastest method for most (76%) RLO problems with polyhedral uncertainty sets (Table 2.2), which matches previous observations [Fischetti and Monaci, 2012]. It also had a lower runtime geometric mean

than either of the two reformulation methods tried, although the difference is not substantial when viewed together with the confidence intervals. We also performed an alternative comparison between the dual simplex reformulation method and the cutting-plane method that again shows that the cutting-plane methods is faster on a vast majority of instances, but without a definitive difference in geometric mean. An explanation for this is to be found in Figure 2-1 which shows that the cutting plane method is within a factor of two of the fastest running time for approximately 90% of problems, but can be at least an order of magnitude slower than reformulation for the remaining 10%. Finally we have included a pseudo-method in Table 2.2 that takes the minimum of the two reformulation times, as many LO solvers can use both methods simultaneously. We see that while cutting-planes are still superior for a majority of problems, the geometric mean of the "best-of-both" reformulation method is lower than the cutting-plane method, although their essentially overlapping confidence intervals suggest not too much importance should be placed on this difference.

This result is inverted for RLO problems with ellipsoidal uncertainty sets, where the reformulation is superior in a small majority of cases (Table 2.3). Given this, it is unsurprising that there is essentially no difference in their median normalized running times. The geometric mean is more conclusive, with the reformulation method taking a clearer lead. This again seems to be due to significantly worse performance in a small number of instances, as evidenced by the performance profile in Figure 2-1.

2.3.2 Results for RMIO

For RMIO problems we have five major variations available:

1. Reformulation (in two minor variations for ellipsoidal sets).

2. Cutting-plane, without cutting-planes added at root node, heuristic disabled.

3. Cutting-plane, **with** cutting-plane method applied to root node, heuristic disabled.

Table 2.4: Summary of RMIO benchmark results for the polyhedral uncertainty set for $\Gamma = 5$. *% Best* is the fraction of problems for which a method had the lowest run time. *Median* and *Geom. Mean* are the median and geometric mean respectively of the run times for each instance normalized by the best time across methods. The two entries marked with an asterisk represent a separate comparison made only between those two methods in isolation from the alternatives.

	% Best	Median (95% CI)	Geom. Mean (95% CI)
Cutting-plane	32.5	1.39 (1.00, 2.39)	2.96 (2.00, 5.26)
Cutting-plane & root	20.0	1.19 (1.01, 1.41)	1.68 (1.40, 2.19)
Cutting-plane & heur.	2.5	1.37 (1.07, 2.12)	2.92 (1.96, 5.23)
Cutting-plane & root & heur.	7.5	1.22 (1.04, 1.55)	1.73 (1.43, 2.23)
Reformulation	37.5	1.28 (1.00, 1.77)	1.88 (1.50, 2.68)
Cutting-plane & root *	57.5	1.00 (1.00, 1.00)	1.58 (1.31, 2.07)
Reformulation *	42.5	1.05 (1.00, 1.62)	1.76 (1.41, 2.56)

4. Cutting-plane, without cutting-planes added at root node, heuristic **enabled**.

5. Cutting-plane, **with** cutting-plane method applied to root node, heuristic **enabled**.

We now investigate in more details the relative benefits of generating constraints at the root node (Section 2.3.2) and of the heuristic (Section 2.3.2).

The results are summarized in Tables 2.4 and 2.5 for the polyhedral and ellipsoidal uncertainty sets respectively. For the polyhedral set we found that the various cutting-plane methods were fastest in 62.5% of instances, although there was little to separate the cutting-plane methods and reformulation in the medians. There was no meaningful difference in the geometric means with the notable exception of the two cutting-plane variants that did not add cuts at the root. We proceeded to simplify the comparison to just the cutting-plane method with cuts at the root and the reformulation. When viewed in this light there seems to be a slight edge to cutting-planes, but the difference is too small to confidently declare as significant. This contradicts the results in Fischetti and Monaci [2012] which concluded that cutting-planes were "significantly worse" than reformulation; this could be attributed to the enlarged test set, the addition of cutting planes at the root, the measurement metric used, or a combination of these factors.

For ellipsoidal uncertainty sets the cutting-plane method was better than reformu-

Table 2.5: Summary of RMIO benchmark results for the ellipsoidal uncertainty set for $\Gamma = 5$. *% Best* is the fraction of problems for which a method had the lowest run time. *Median* and *Geom. Mean* are the median and geometric mean respectively of the run times for each instance normalized by the best time across methods. The two entries marked with an asterisk represent a separate comparison made only between those two methods in isolation from the alternatives.

	% Best	Median (95% CI)	Geom. Mean (95% CI)
Cutting-plane	30.0	1.08 (1.00, 1.42)	2.51 (1.66, 4.53)
Cutting-plane & root	22.5	1.20 (1.00, 1.55)	1.83 (1.44, 2.80)
Cutting-plane & heur.	20.0	1.02 (1.00, 1.39)	2.52 (1.67, 4.64)
Cutting-plane & root & heur.	2.5	1.33 (1.03, 1.59)	1.88 (1.47, 2.81)
Reformulation	12.5	1.83 (1.01, 8.59)	5.92 (3.31,12.54)
Reformulation (outer-approx)	12.5	1.27 (1.01, 1.41)	2.03 (1.52, 3.20)
Cutting-plane & root *	72.5	1.00 (1.00, 1.00)	1.31 (1.12, 1.89)
Reformulation (outer-approx) *	27.5	1.07 (1.00, 1.25)	1.45 (1.24, 1.96)

lation in even more instances (75%) than for the polyhedral sets. Interestingly, and in contrast to the polyhedral set results, the cutting-plane variations without cuts at the root node performed better in the median than any other variation, although the geometric mean results reverse this result. Additionally the outer-approximation variant of the reformulation appears to perform far better overall than the alternative, perhaps due to the structure of reformulated RMIOs. Finally we isolated the best cutting-plane and reformulation variants and compared them, indicating that cutting-planes had an edge but that again the difference in geometric mean was too small relative to the uncertainty to be declared significant.

Treatment of root node

In Section 2.2.2 we considered the option of applying the RLO cutting-plane method to the root relaxation of the RMIO master problem. While this would possibly generate unnecessary constraints, we hypothesized that it may guide the search process to avoid considering integer solutions that are not feasible with respect to the uncertain constraints. We experimented to determine whether this option helps or not by solving with and without it, and the summary of the results is presented in in Table 2.6, where all results are relative to the better of with and without root cuts. We note that there is essentially no difference in median performance, and the difference only

Table 2.6: RMIO benchmark results for $\Gamma = 5$ comparing performance when cuts are added at the root node or not, with methods paired based on whether the heuristic was also used and which uncertainty set they are for. All metrics calculated as described in Table 2.4.

	% Best	Median (95% CI)	Geom. Mean (95% CI)
Poly., no root cuts	50.0	1.00 (1.00, 1.16)	1.96 (1.43, 3.49)
Poly., root cuts	50.0	1.00 (1.00, 1.01)	1.12 (1.05, 1.29)
Poly., heur. & no root cuts	52.5	1.00 (1.00, 1.00)	1.86 (1.35, 3.24)
Poly., heur. & root cuts	47.5	1.00 (1.00, 1.00)	1.10 (1.05, 1.22)
Ell., no root cuts	60.0	1.00 (1.00, 1.00)	1.71 (1.26, 3.03)
Ell., root cuts	40.0	1.00 (1.00, 1.01)	1.25 (1.13, 1.54)
Ell., heur. & no root cuts	65.0	1.00 (1.00, 1.00)	1.67 (1.23, 2.94)
Ell., heur. & root cuts	35.0	1.00 (1.00, 1.09)	1.24 (1.13, 1.49)

Table 2.7: RMIO benchmark results for $\Gamma = 5$ comparing performance when cuts are added at the root node or not, with methods paired based on which uncertainty set they are for and whether the instances are feasible or not. All metrics calculated as described in Table 2.4.

	% Best	Median (95% CI)	Geom. Mean (95% CI)
Poly., feas., no root cuts	56.5	1.00 (1.00, 1.00)	1.27 (1.13, 1.77)
Poly., feas., root cuts	43.5	1.00 (1.00, 1.01)	1.10 (1.04, 1.32)
Poly., infeas., no root cuts	18.2	22.99 (1.00,29.92)	12.43 (1.87,27.05)
Poly., infeas., root cuts	81.8	1.00 (1.00, 1.00)	1.22 (1.00, 1.49)
Ell., feas., no root cuts	67.2	1.00 (1.00, 1.00)	1.08 (1.03, 1.20)
Ell., feas., root cuts	32.8	1.00 (1.00, 1.02)	1.25 (1.12, 1.61)
Ell., infeas., no root cuts	30.8	20.50 (1.00,29.68)	8.54 (1.67,24.84)
Ell., infeas., root cuts	69.2	1.00 (1.00, 1.00)	1.24 (1.00, 1.78)

appears in the geometric means. In particular we note that not adding cuts at the root can result in much longer solve times, dragging the geometric means higher. This behaviour is fairly consistent across uncertainty set type and whether the heuristic is used or not.

Further investigation reveals that the difference in solve times is mostly due to its superior performance on infeasible instances, where the difference between the two is drastic in both median and geometric mean. Something similar is suggested in Fischetti and Monaci [2012], although not explored fully. This can be seen in the performance profiles in Figure 2-2 where the no-root-cut lines fall underneath the other variations on the right side of the plots. For feasible instances there is not a

Table 2.8: RMIO benchmark results for $\Gamma = 5$ comparing performance when the heuristic is used to not using it, with methods paired based on which uncertainty set they are for. All metrics calculated as described in Table 2.4.

	% Best	Median (95% CI)	Geom. Mean (95% CI)
Poly., root cuts	67.5	1.00 (1.00, 1.00)	1.01 (1.00, 1.06)
Poly., heur. & root cuts	32.5	1.00 (1.00, 1.01)	1.04 (1.02, 1.07)
Ell., root cuts	67.5	1.00 (1.00, 1.00)	1.02 (1.00, 1.07)
Ell., heur. & root cuts	32.5	1.00 (1.00, 1.01)	1.05 (1.02, 1.09)

substantial difference between the two. We conclude from this evidence that adding constraints to make the root fractional solution feasible for all uncertain constraints is generally advisable, and that if there is a reasonable expectation that infeasibility is possible this option should definitely be used.

Repair heuristic

Section 2.2.2 details a heuristic that takes integer solutions produced during branch-and-bound that violate one or more uncertain constraints and attempts to make them feasible by modifying only the continuous variables. We evaluated the solution times with the heuristic enabled relative to the solution times with it disabled, and the summary of results is presented in Table 2.8. Using the heuristic was better than not using it in approximately one-third of the cases for both polyhedral and ellipsoidal sets. The geometric means were roughly one, indicating that on average the heuristic doesn't make much of a difference for the problems we considered.

2.3.3 Implementation Considerations

The implementation details for the individual methods could make a substantial difference. In particular, MISOCP functionality in commercial solvers is relatively new compared to LO and MIO as of the time of writing, which suggests that an efficient, problem-specific cutting plane method like the type demonstrated in this chapter may have an edge over a generic solver. Over time, one might expect the performance of MISOCP reformulations to improve with the maturity of commercial implementations. This may explain why the cutting-plane method was better for a majority of

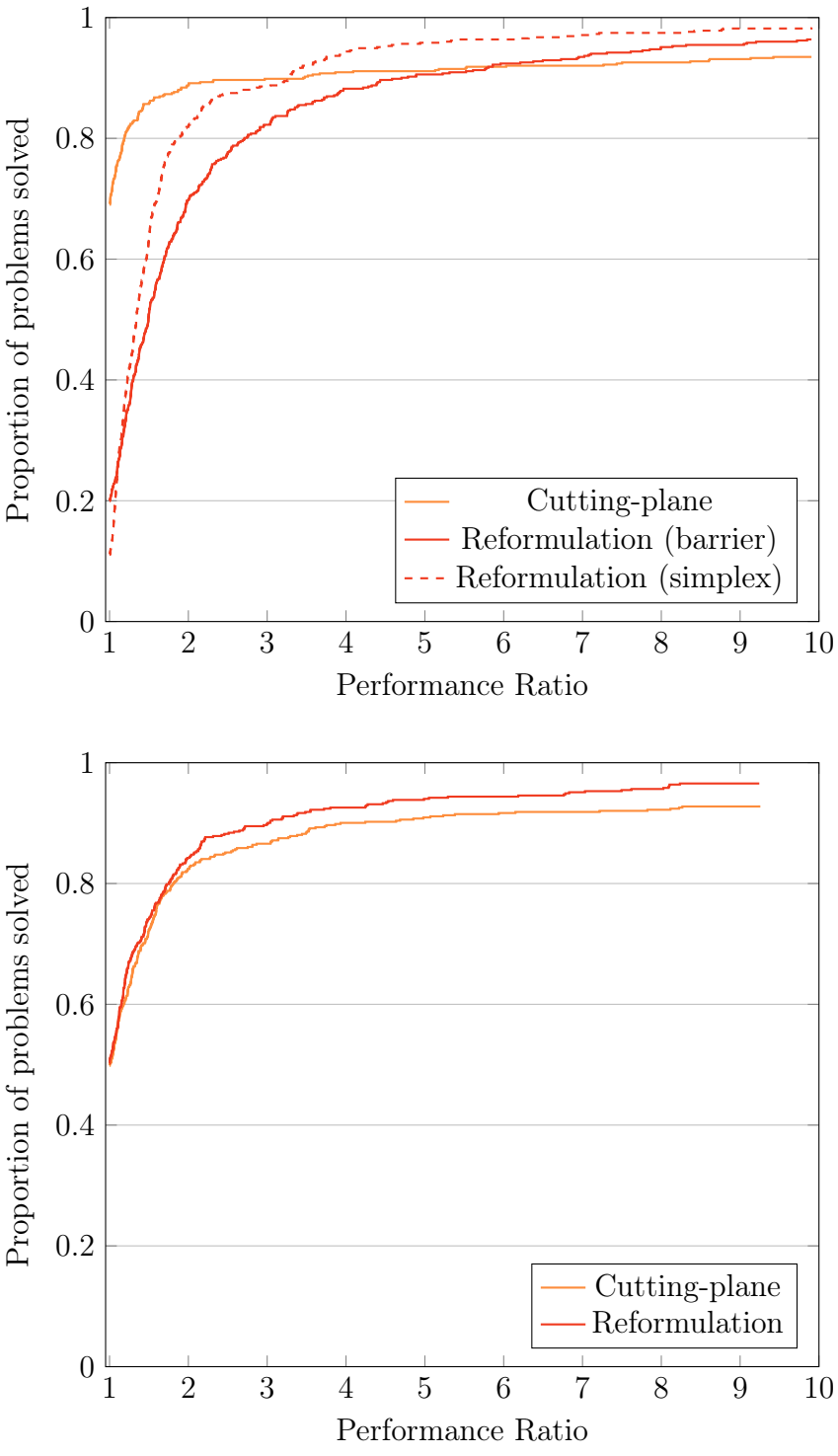

Figure 2-1: Performance profiles [Dolan and Moré, 2002] for RLO instances with polyhedral and ellipsoidal uncertainty, above and below, respectively. Higher lines indicate superior performance.

.

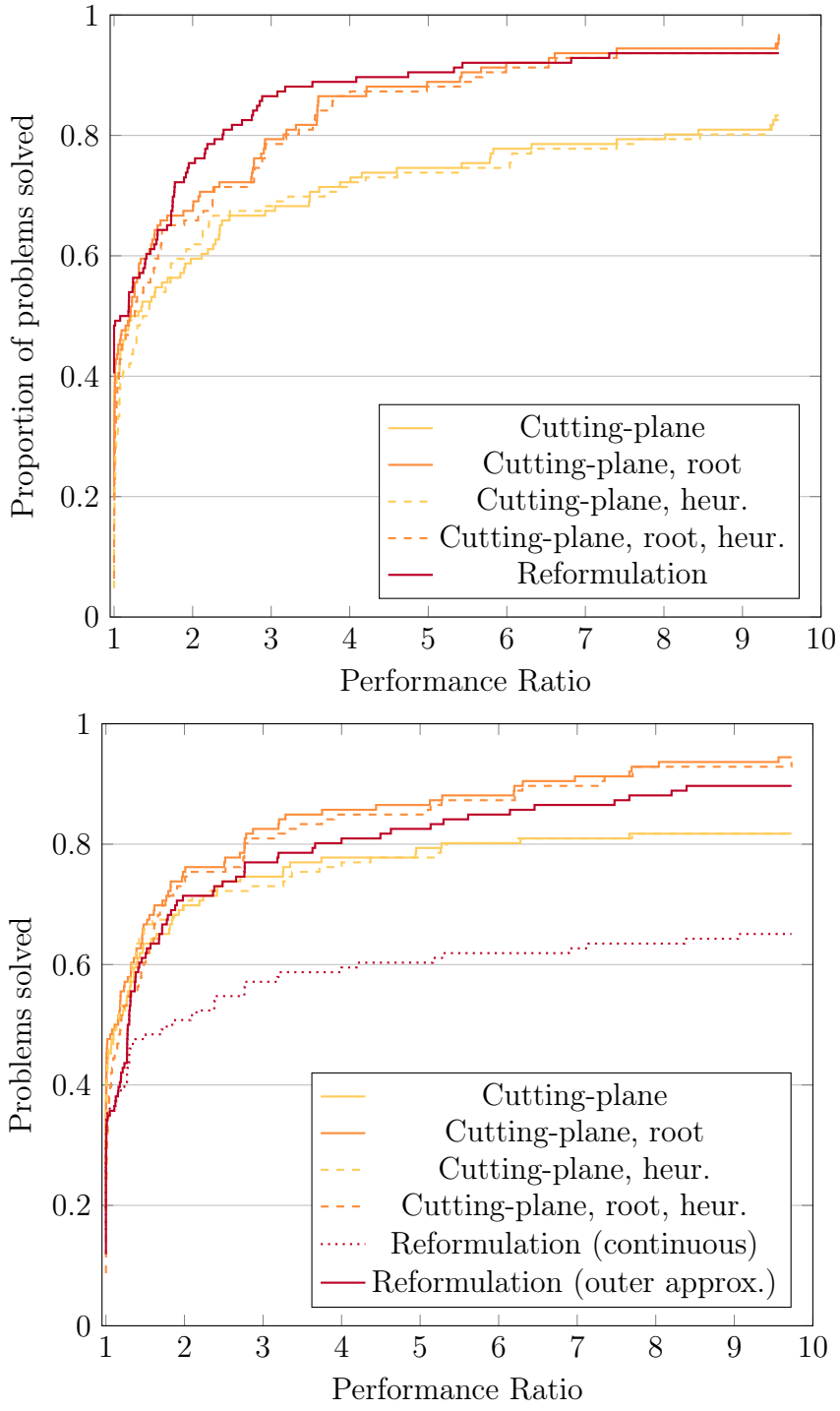

Figure 2-2: Performance profiles [Dolan and Moré, 2002] for RMIO instances with polyhedral and ellipsoidal uncertainty, above and below, respectively. Higher lines indicate superior performance. "root" indicates adding constraints at the B&B root note, and "heur" indicates that the proposed feasibility heuristic is enabled.

the RMIO ellipsoidal uncertainty set instances we evaluated. On the other hand, MIO solvers have presolve functionality that is capable of powerful problem reductions, which would be to the benefit of reformulations. By implementing our cutting-plane generation code in C++ and connecting to the solver using *in memory* libraries, we avoided as much overhead as possible; cutting-plane overhead may grow with a less tightly coupled connection or if a slower language is used to generate cuts. We also used single-threaded computation throughout - cutting-plane generation may interact with a multi-threaded solver in unexpected ways that may slow down the solver.

2.4 Evaluation of Hybrid Method

In the previous section we mainly focused on the direct comparison of different variations of the cutting-plane and reformulation methods for solving RO problems. In this section, we change the focus to how best to use this information in practice. We propose applying both the reformulation and the cutting-plane method *simultaneously* to the same problem, in much the same way that some modern solvers apply both the simplex and interior point methods to solving a (deterministic) linear optimization problem.

We define the *hybrid* method each problem class and uncertainty set type as follows:

- RLO problems with polyhedral uncertainty sets: simultaneously solve with the cutting-plane method and both reformulation methods (simplex and interior point).

- RLO problems with ellipsoidal uncertainty sets: simultaneously solve with the cutting-plane method and the reformulation method.

- RMIO problems with polyhedral uncertainty sets: simultaneously solve with the cutting-plane method (with root cuts) and with the reformulation method.

- RMIO problems with ellipsoidal uncertainty sets: simultaneously solve with

Table 2.9: Summary of hybrid method RLO benchmark results for the polyhedral uncertainty set with $\Gamma = 5$. Both measures are functions of the hybrid method runtime relative to another method's runtime (Equation 2.2).

	Median (95% CI)	Geom. Mean (95% CI)
Cutting-plane	1.00 (1.00, 1.00)	0.66 (0.53, 0.76)
Reformulation (Barrier)	0.64 (0.56, 0.71)	0.53 (0.48, 0.58)
Reformulation (Dual Simplex)	0.68 (0.65, 0.71)	0.59 (0.54, 0.64)
Reformulation (Best of Both)	0.79 (0.73, 0.83)	0.72 (0.67, 0.76)

the cutting-plane method (with root cuts) and with the reformulation method (outer-approximation).

.

The statistic we are interested in is a slight variant of the one considered in Section 2.3. For a particular combination of problem class (RLO, RMIO), uncertainty set (polyhedral, ellipsoidal), and Γ define T_k^H to be run time for the hybrid method. We are interested in the reduction in running time for the hybrid method versus any single method M, that is

$$\frac{T_k^H}{T_k^M}. \tag{2.2}$$

This quantity is bounded by 0 and 1 for RLO problems by construction, but may be greater than 1 for RMIO problems (for example, if we compare against the heuristic method and for a particular instance using the heuristic would have been better). We will consider the median, geometric mean, and general distribution of this quantity.

2.4.1 Results for RLO

For the polyhedral uncertainty set (Table 2.9) we defined the hybrid method to be the best of all three methods. As the cutting-plane method was best in 75% of instances, there is no median improvement versus the cutting-plane method, and roughly 15% median improvement versus the better of the reformulations. However, if we look at the geometric means then the hybrid method is significantly better, with runtimes about 50% to 75% of any single method.

For the ellipsoidal uncertainty set (Table 2.10) the hybrid method is simply the

Table 2.10: Summary of hybrid method RLO benchmark results for the ellipsoidal uncertainty set with $\Gamma = 5$. Both measures are functions of the hybrid method runtime relative to another method's runtime (Equation 2.2).

	Median (95% CI)	Geom. Mean (95% CI)
Cutting-plane	1.00 (0.88, 1.00)	0.57 (0.46, 0.66)
Reformulation	0.83 (0.78, 0.97)	0.68 (0.60, 0.73)

Table 2.11: Summary of hybrid method RMIO benchmark results for the polyhedral uncertainty set with $\Gamma = 5$. Both measures are functions of the hybrid method runtime relative to another method's runtime (Equation 2.2).

	Median (95% CI)	Geom. Mean (95% CI)
Cutting-plane	0.72 (0.33, 0.99)	0.36 (0.20, 0.55)
Cutting-plane & root	1.00 (0.35, 1.00)	0.63 (0.49, 0.77)
Cutting-plane & heur.	0.79 (0.51, 0.98)	0.37 (0.20, 0.55)
Cutting-plane & root & heur.	0.95 (0.66, 0.99)	0.62 (0.47, 0.75)
Reformulation	0.95 (0.59, 1.00)	0.57 (0.39, 0.70)

better of the two available methods. It is nearly twice as fast on average as using the cutting-plane method and has runtimes less than about 70% on average of the reformulation method. Its interesting to note that the performance difference for the median is not as dramatic suggesting that for most problems there is not much difference, but for some instances the difference is substantial.

2.4.2 Results for RMIO

The hybrid method for RMIO problems with the polyhedral uncertainty set (Table 2.11) is defined to be the best of the cutting-plane method with root cuts and the reformulation method. As with RLO problems we see only moderate improvements in median, but the improvement in geometric mean shows improvements in runtime by a factor of approximately two to three.

For ellipsoidal uncertainty sets (Table 2.12) the hybrid method the best of the cutting-plane method with root cuts and the reformulation (outer-approximation) method. Once more we see only moderate improvements in median, but the improvement in geometric mean shows improvements in runtime up to a factor of approximately two (and even more for the alternative reformulation solution method).

Table 2.12: Summary of hybrid method RMIO benchmark results for the ellipsoidal uncertainty set with $\Gamma = 5$. Both measures are functions of the hybrid method runtime relative to another method's runtime (Equation 2.2).

	Median (95% CI)	Geom. Mean (95% CI)
Cutting-plane	0.99 (0.85, 1.00)	0.55 (0.29, 0.86)
Cutting-plane & root	1.00 (0.78, 1.00)	0.76 (0.52, 0.89)
Cutting-plane & heur.	1.00 (0.77, 1.00)	0.55 (0.29, 0.85)
Cutting-plane & root & heur.	0.99 (0.93, 1.00)	0.74 (0.52, 0.87)
Reformulation	0.83 (0.14, 1.00)	0.24 (0.10, 0.47)
Reformulation (outer-approx)	0.93 (0.78, 1.00)	0.69 (0.51, 0.81)

Histograms of the hybrid runtime versus method runtimes for both uncertainty sets are displayed in Figure 2-3. We can see that, as expected, the histograms for the methods selected for inclusion in the hybrid method are mostly bunched around 1.0, but that for the other methods there is another peak between 0.0 and 0.2 for the instances that were solved substantially quicker with the hybrid method. It is these instances that drive the difference in geometric mean and show where a hybrid method could offer the most value.

2.4.3 Implementation Considerations

We note that implementation details may restrict the realizable benefits of the hybrid method. In particular, we considered only single-threaded computation for each individual method, and assumed there was no performance impact from running multiple methods simultaneously. To examine these assumptions and their impact on practical implementations we will assume that we are running on machine that can run $T \geq 2$ threads independently with minimal impact on each other. For example, a "quad core" machine could run $T = 4$ threads with minimal impact. We will analyze the impact of these realities on the two problem classes independently.

For the RLO problem we used three methods, so would require $T \geq 3$ to obtain the results stated above. An examination of the performance profiles suggests that if T is only two then we should use the cutting-plane method and the simplex method for reformulation. If $T \geq 4$ the question of what to do with the extra threads arises. While the reformulation method with the simplex algorithm is single-threaded, the interior-

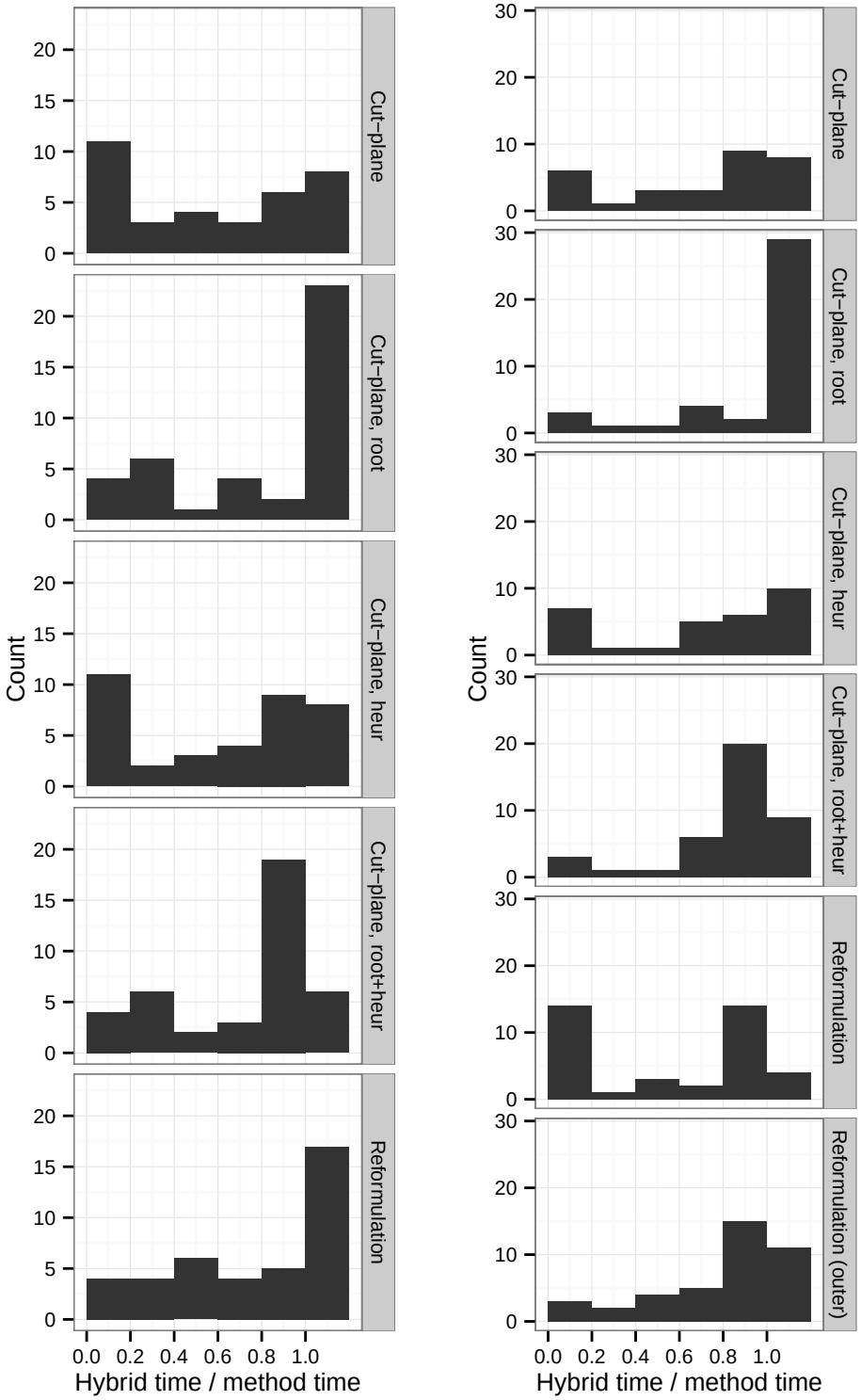

Figure 2-3: Histograms of the runtime for the hybrid method normalized by each of the individual methods (Equation 2.2). The left column is for RMIO problems with polyhedral uncertainty sets, and the right column is for RMIO problems with ellipsoidal uncertainty sets.

point reformulation method can be run in multi-threaded mode. The cutting-plane method relies on the simplex method to solve the master problem, but the cutting-planes could be generated in parallel. Thus on a machine with $T = 8$, for example, one thread could be allocated to solving the reformulation with the simplex method, two threads could be allocated to the cutting-plane method, and the remainder to the interior-point method for the reformulation.

Both the hybrid methods for RMIO problems use only two methods, requiring only that $T \geq 2$. However, both cutting-plane and reformulation methods for RMIO can use many threads, begging the question of how to distribute T threads among them. This question can only practically be answered with further instance-specific experimentation beyond the scope of this chapter. In general, however, it is far from trivial to achieve a perfect speed-up factor of T when using T threads to solve a single MIO problem Koch et al. [2012], which suggests that a hybrid approach could provide an improvement over a pure strategy of providing all available threads to a single solver.

A parallel hybrid strategy could be further enhanced by communicating between the two solvers. Consider first the reformulation at the presolve stage: any inferences made about variable bounds, including fixing variables, determined from the reformulation are also valid for the cutting-plane method. Thus the cutting-plane method can benefit from the structure of the reformulation, which could provide improvements above those predicted above. Furthermore a feasible solution found with the cutting-plane method is also feasible for the reformulation method, allowing us to tighten the integrality gap from one direction. Finally the lower bound from the reformulation method is also a valid lower bound for the cutting-plane algorithm, allowing us to tighten the integrality gap from the other direction. Carvajal et al. [2014] reported success with a similar approach in the context of general MIO problems.

2.5 Discussion

The relative benefits of cutting planes versus reformulations are a mixed story when compared directly against each other. The two main metrics considered were the median and geometric mean of the runtime for each method normalized by the best runtime across methods. In general, the performance gap between them was not large in either metric for any combination of problem class (RLO, RMIO) and uncertainty set, but especially in the median. The geometric mean generally revealed greater differences due to small numbers of instances that are dramatically better with one method than the other. This is reflected in the tails of the performance profiles in Figures 2-1 and 2-2. We summarize the results as follows:

- For RLO problems with polyhedral uncertainty sets, there was some evidence that the cutting-plane method was superior for a majority of cases, but there is no clear overall winner.

- For RLO problems with ellipsoidal uncertainty sets, the reformulation method was better than the cutting-plane method.

- For RMIO problems with polyhedral uncertainty sets, there was no clear winner between the cutting-plane method (with cuts added at the root) and the reformulation method.

- For RMIO problems with ellipsoidal uncertainty sets, there was some evidence that the cutting-plane method (with cuts added at the root) was better than the reformulation method (using the outer-approximation option).

The most practical benefit of benchmarking the methods was to determine the performance of a hybrid solver that runs both simultaneously. When we consider the performance of this hybrid method we see reductions in runtime to 50% to 75% across all combinations of problem and uncertainty set. This is a clear win, and suggests that a hypothetical robust optimization solver should follow this hybrid strategy. This would be achieved by modeling the RO problem directly (perhaps with a RO-specific

modeling language), then relying on the modeling tool or RO solver to automatically produce both the reformulation and the cutting-plane generator. As we noted in 2.4.3, further synergies over simply running both methods in isolation may be possible, at the cost of increased implementation complexity.

Future avenues for the study of computational aspects of RO would include benchmarking more and tougher MIO instances that take longer to solve. As discussed in Section 2.2.1, methods from nonlinear and stochastic programming may be able to reduce the number of cuts required to reach feasibility, although our preliminary experiments in this domain were not successful. Novel warm-starting methods, presolve techniques and additional heuristics may improve both methods, but especially the cutting-plane method. Finally we note that reformulations have a block structure similar to that seen in stochastic programming, which may be exploitable to speed up the solver.

Chapter 3

Multistage Robust Mixed Integer Optimization with Adaptive Partitions

3.1 Introduction

Robust optimization (RO) is a methodology for addressing uncertainty in optimization problems where *uncertain parameters* are modeled as belonging to *uncertainty sets*, which contrasts with the more traditional approach of modeling uncertainty with probability distributions. We solve an RO problem by optimizing with respect to the worst-case realization of the uncertain parameters over the uncertainty set Ξ,

$$\max_{\mathbf{x}\in\mathcal{X}} \min_{\boldsymbol{\xi}\in\Xi} c(\boldsymbol{\xi}, \mathbf{x})$$
$$\text{subject to } \mathbf{g}(\boldsymbol{\xi}, \mathbf{x}) \leq \mathbf{0} \qquad \forall \boldsymbol{\xi} \in \Xi$$

where $\mathbf{x}$ is the vector of decision variables and $\boldsymbol{\xi}$ is a vector representing the uncertain parameters. For a survey of robust optimization we refer the reader to Bertsimas et al. [2011a] and Ben-Tal et al. [2009]. The optimization problem (3.1) commonly has an infinite number of constraints but is tractable for many uncertainty sets of interest by reformulating the problem to obtain a deterministic optimization problem of finite

size, or by generating constraints only as needed until the solution is optimal and feasible with respect to the uncertain constraints (see Chapter 2 for more details).

RO was introduced in the context of single-stage problems as a way to address uncertainty in problem data. Since then it has been shown to be a more general methodology to address multistage optimization problems, where the uncertain parameters are revealed over time. In most multistage problems our future *recourse* decisions can be made with knowledge of at least some of the uncertain parameters: those realized by the time those decisions must be made. If we place no restrictions on the functional relationship between uncertain parameters and recourse decisions, which is the ideal case, then the resulting problem is known to be hard in both the computational complexity sense [Ben-Tal et al., 2004], and has proved to be hard to solve in practice. Some work on addressing this fully adaptive case includes a Benders' decomposition-like approach to solve a unit commitment problem in Bertsimas et al. [2013b], and the generalized "column-and-constraint generation" framework for two-stage continuous problems of Zeng and Zhao [2013]. A goal of the adaptive optimization (AO) literature then is to provide methods to approximate and approach this fully adaptable ideal and to quantify, if possible, how good these approximations are.

Perhaps the most popular approach in the AO literature to date has been to restrict the recourse decisions to be simple functions of the uncertain parameters. Affine adaptability, also known as linear decision rules, was introduced to the RO setting in Ben-Tal et al. [2004] and has proven to be useful in a wide variety of settings including control problems [Goulart et al., 2006], supply chain problems [Ben-Tal et al., 2005], and project management [Chen et al., 2007]. The choice of these linear decision rules is primarily one of computational practicality with no real expectation that they will be optimal in general (where optimality would be finding an affinely adaptive policy that performs as well as the fully adaptive policy). Many of the papers cited here demonstrate these shortcomings with both examples and more realistic experiments, and many researchers have characterized the conditions under which, and to what degree, affine adaptability performs well, e.g. Bertsimas et al. [2010], Bertsimas and

Goyal [2012], Bertsimas and Bidkhori [2015]. More complicated approaches trade computational efficiency for optimality, for example by using polynomial adaptability [Bertsimas et al., 2011b] and deflected linear decision rules [Chen et al., 2008]. A major shortcoming of all these approaches is that they do not allow for discrete recourse decisions, significantly reducing their modeling power for adaptive mixed-integer optimization (AMIO) problems. Bertsimas and Caramanis [2007] acknowledge this and propose a decision rule for integer decisions as part of their sampling approach. Bertsimas and Georghiou [2015] propose a cutting-plane method that allows for piecewise linear decision rules for continuous recourse decisions and piecewise constant decision rules for integer recourse decisions. This approach was shown to be able to find good solutions to small problems, but does not scale well to problems with many time stages. A recent extension by Bertsimas and Georghiou [2016] for binary recourse decisions that allows for a reformulation demonstrates substantially improved performance on multistage problems.

An alternative approach is finite adaptability, where instead of focusing on the functional form of the recourse decisions we instead partition the uncertainty set and have different recourse decisions for each partition. This can be viewed as modeling the recourse decisions as piecewise constant functions of the uncertain parameters, with the domain of each piece (or equivalently the uncertainty set partitioning) either fixed, or decided endogenously as part of the optimization problem. One of the key benefits of this approach is that it handles discrete recourse decisions naturally, suggesting it is a good choice for AMIO. The quality of solutions obtained with a finite adaptability approach rests entirely on how the partitions are selected. For example, in Vayanos et al. [2011] the uncertainty set is partitioned ahead of time using hyper-rectangles. A bilinear optimization problem that decides the best two-partition recourse decision is proposed in Bertsimas and Caramanis [2010], and the gap between a fully adaptive solution and a finitely adaptive solution is bounded. Finally, Hanasusanto et al. [2015] propose a general methodology for obtaining optimal K-partition recourse decisions for two-stage binary optimization problems, and characterize when their approach is able to match the fully adaptive solution. They

are able to solve a variety of problems, although even problems with as few as three partitions and twenty binary second stage variables cannot be solved to optimality within two hours. While this recent work represents great progress in AMIO, the size of AMIO problems that can be solved in a reasonable amount of time remains small and lags significantly behind the size of tractable continuous AO problems, particularly in the multistage case.

In this chapter we will consider multistage AMIO problems of the general form

$$z_{full} = \min_{\mathbf{x}\in\mathcal{X}} \max_{\boldsymbol{\xi}\in\Xi} \sum_{t=1}^{T} \mathbf{c}^t(\boldsymbol{\xi}) \cdot \mathbf{x}^t\left(\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^{t-1}\right) \tag{3.1}$$

$$\text{subject to} \quad \sum_{t=1}^{T} \mathbf{A}^t(\boldsymbol{\xi}) \cdot \mathbf{x}^t\left(\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^{t-1}\right) \leq \mathbf{b}(\boldsymbol{\xi}) \quad \forall \boldsymbol{\xi} = \left(\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^T\right) \in \Xi$$

where $\boldsymbol{\xi}$ represents the uncertain parameters, $\mathbf{A}$, $\mathbf{b}$ and $\mathbf{c}$ define linear constraints and the objective respectively (all themselves affine functions of $\boldsymbol{\xi}$), and $\mathbf{x}^t$ is the vector of decision variables for stage t, which can be a mixture of discrete and continuous variables as captured by the deterministic set of constraints $\mathcal{X}$. For $t = 1$ we note that $\mathbf{x}^1$ is a non-adaptive here-and-now decision that does not depend on $\boldsymbol{\xi}$.

The key contribution of this chapter is a new "partition-and-bound" method for solving the multistage AMIO problem (3.1) that scales better to larger instances than alternative methods. In particular, as our method and the ones that we are comparing ourselves with do not necessarily find the fully adaptive solution, the metric by which we assess a method is the quality of the solutions it finds versus the computational effort required to obtain these solutions. The method builds on the finite adaptability approach to AMIO by utilizing the information obtained about the (possibly already partitioned) uncertainty set when solving an AO problem to select a new partitioning scheme. The same information is used to calculate a lower bound on the fully adaptive solution by adapting the sampling proposal of Hadjiyiannis et al. [2011]. This process of solving and updating the partitioning can be repeated until a termination criterion in terms of the bound gap, or some other limit (total iterations or computation time) is reached. Our proposal can be extended in many ways: in this chapter we show that

affine adaptability for continuous recourse decisions can be trivially incorporated into our approach (leading to piecewise affine decision rules), but other extensions such as quadratic objectives and constraints can be also easily incorporated. We also give a detailed treatment to the interactions between finite adaptability and the need to enforce *nonanticipativity* for multistage problems - the natural notion that a decision at time t must be made without certain knowledge of uncertain parameters that will be revealed after time t.

Our approach lies somewhere between those that determine the optimal partitions simultaneously with the values of the decisions [Bertsimas and Caramanis, 2010, Hanasusanto et al., 2015] and methods that select a partitioning *a priori* [Vayanos et al., 2011]. While finalizing the paper from which this chapter was developed, we became of aware of a very recently submitted paper by Postek and Den Hertog [2014] that presents an iterative finite partitioning approach similar to the idea we present in this chapter. Throughout this chapter we will contrast and compare our approach to this alternative, both from a theoretical and computational point of view.

After the initial submission of the paper from which this chapter was developed, Song and Luedtke [2015] proposed an "adaptive partition-based approach" for two-stage stochastic programs (SP) that partitions sets of scenarios. Their approach uses the solutions to these relaxed problems to change the partitioning scheme – a scenario-based analogue to the approach of this chapter. An interesting point of difference is the perspective of the two approaches: in the SP case we start with many individual scenarios that we group into partitions, whereas in the RO approach we start with a single set and seek to split apart into partitions. Song and Luedtke [2015] build upon other similar aggregation techniques discussed in the SP literature, especially work aimed to at improving the tractability of sample-average approximation. We refer interested readers to their work for a more comprehensive treatment of the related SP literature.

We have structured the chapter as follows:

- In Section 3.2, we present a theoretical result describing a property that a new partitioning of an uncertainty set must have to improve a solution of an AMIO

over an existing partitioning scheme. We use this result to propose a method of constructing partitions using the solution of AMIO problems that is inspired by Voronoi diagrams. In this section, we initially focus on the two-stage case: By combining this with a lower bounding method, we obtain an iterative partition-and-bound method for two-stage AMIO. We explain how affine adaptability can be incorporated into the method, and provide a worked example to aid in implementation.

- In Section 3.3, we analyze our method from two points of view. We show that our method has a desirable non-increasing property that enables computational "warm starts", but provide an example that demonstrates that there is no general guarantee of convergence for the solutions produced by our method to the fully adaptive solution. We also consider the question of the growth in the number of partitions with increasing partitions. We show that while the number of partitions can grow very large after many iterations, the quality of the solution those partitions lead to for a fixed computational budget is the more relevant quantity. We also present a small modification to the method that can greatly reduce the number of partitions necessary.

- In Section 3.4, we generalize the method to the multistage case. We first demonstrate the issues that must be considered in choosing the partitioning scheme while satisfying nonanticipativity. We then describe the general multistage partitioning scheme and a simple routine to determine the minimum set of nonanticipativity constraints to add.

- In Section 3.5, we provide results of computational experiments that show that our method produces good solutions versus total computation time. In particular, we solve instances of a capital budgeting problem and a multistage lot sizing problem, including a comparison with the methods of Hanasusanto et al. [2015], Bertsimas and Georghiou [2015], and Postek and Den Hertog [2014].

- In the final Section 3.6, we provide some concluding remarks and thoughts on

future directions.

Notation

We use lowercase, non-bold face symbols for scalar quantities (e.g., $z \in \mathbb{R}$) and a bold face for vectors (e.g., $\mathbf{x} \in \mathbb{R}^n$). Matrices are denoted by bold face uppercase symbols, e.g., $\mathbf{A} \in \mathbb{R}^{m \times n}$. We use Ξ to denote an uncertainty set and $\boldsymbol{\xi}$ to denote uncertain parameters. Our uncertain parameters are typically a vector-of-vectors, i.e.,

$$\boldsymbol{\xi} = \left(\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^t, \ldots, \boldsymbol{\xi}^T\right), \tag{3.2}$$

where $\boldsymbol{\xi}^t$ is the vector of uncertain parameters for stage t. The j^{th} component of the uncertain parameters for time stage t is expressed with a subscript, e.g., ξ_j^t. For particular realizations of uncertain parameters we use a hat: For example $\hat{\boldsymbol{\xi}}_i$ is the i^{th} realization and $\hat{\boldsymbol{\xi}}_i^t$ is the vector of uncertain parameters for sample i and time stage t.

3.2 Partition-and-bound Method for Two-Stage AMIO

In this section, we describe a partition-and-bound method that takes a finite adaptability approach to two-stage AMIO problems. In particular, we consider the problem

$$\begin{aligned} \min_{\mathbf{x} \in \mathcal{X}, z} \quad & z & & \\ \text{subject to} \quad & \mathbf{c}^1(\boldsymbol{\xi}) \cdot \mathbf{x}^1 + \mathbf{c}^2(\boldsymbol{\xi}) \cdot \mathbf{x}^2(\boldsymbol{\xi}) \leq z & \forall \boldsymbol{\xi} \in \Xi & \\ & \mathbf{a}_i^1(\boldsymbol{\xi}) \cdot \mathbf{x}^1 + \mathbf{a}_i^2(\boldsymbol{\xi}) \cdot \mathbf{x}^2(\boldsymbol{\xi}) \leq b_i(\boldsymbol{\xi}) & \forall \boldsymbol{\xi} \in \Xi,\ i \in \mathcal{I}, & \end{aligned} \tag{3.3}$$

which has m linear uncertain constraints ($\mathcal{I} = \{1, \ldots, m\}$) and the objective expressed in epigraph form. This is a special case of the multistage problem (3.1) where $T = 2$. We first demonstrate some properties of AO solutions that motivate the design of our proposed method, then discuss the implementation details.

3.2.1 Partitioning Scheme

Finite adaptability is a simple approach for approximating fully adaptive wait-and-see decisions that can vary with the uncertain parameters. In the context of our two-stage problem (3.3) we can view finite adaptability as a restriction of the decisions $\mathbf{x}^2(\boldsymbol{\xi})$ to the class of piecewise constant policies, i.e.

$$\mathbf{x}^2(\boldsymbol{\xi}) = \begin{cases} \mathbf{x}_1^2, & \forall \boldsymbol{\xi} \in \Xi_1, \\ \vdots & \\ \mathbf{x}_K^2, & \forall \boldsymbol{\xi} \in \Xi_K, \end{cases} \tag{3.4}$$

where $\Xi_k, k \in \{1, \ldots, K\}$ defines a partitioning of Ξ and $\mathbf{x}_k^2$ is the implemented decision if the realized value of $\boldsymbol{\xi}$ is in the partition Ξ_k. If the realized value lies on the boundary of two partitions then we may select either decision, as the decision associated with each partition is feasible for all values of the uncertain parameters on the boundary. We will converge in the limit to the fully adaptive solution as the number of partitions grows and the diameter of the largest partition approaches 0. Therefore, there is a natural trade-off between the number of partitions (and the solution time) and how "close" our solution is to the fully adaptive solution, and a goal when using finite adaptability is to identify a partitioning scheme that is near the efficient frontier of this trade-off.

To motivate our selection of partitioning scheme we will consider a variant of (3.3) where the second-stage decisions cannot change with $\boldsymbol{\xi}$, i.e. a static policy:

$$\begin{aligned} z_{static} = \min_{\mathbf{x} \in \mathcal{X}, z} \quad & z \\ \text{subject to} \quad & \mathbf{c}^1 \cdot \mathbf{x}^1 + \mathbf{c}^2 \cdot \mathbf{x}^2 \leq z \\ & \mathbf{a}_i^1(\boldsymbol{\xi}) \cdot \mathbf{x}^1 + \mathbf{a}_i^2(\boldsymbol{\xi}) \cdot \mathbf{x}^2 \leq b_i(\boldsymbol{\xi}) \qquad \forall \boldsymbol{\xi} \in \Xi,\ i \in \mathcal{I}. \end{aligned} \tag{3.5}$$

We will initially restrict ourselves to the case where all decision variables are continuous, the uncertainty set is polyhedral (which ensures that a cutting plane method terminates), the objective coefficients are certain, and the problem is feasible and

bounded. Consider using a cutting-plane method to solve this static problem: that is, we iteratively solve a deterministic relaxation of the RO problem with a finite subset of the possible constraints and add violated constraints from the full RO problem until the solution of the relaxation is feasible with respect to all uncertain parameters in the uncertainty set. For each constraint i we define $\mathcal{A}_i$ to be set of *active uncertain parameters* $\hat{\boldsymbol{\xi}}$ corresponding to the generated constraints (cuts) that have zero slack. These sets may be empty, singletons, or even have multiple elements (multiple cuts for i with zero slack). After solving for the static policy we consider partitioning the uncertainty set into two sets Ξ_1 and Ξ_2, that is

$$\begin{aligned} z_{part} = \min_{\mathbf{x}\in\mathcal{X},z} \quad & z \\ \text{subject to} \quad & \mathbf{c}^1 \cdot \mathbf{x}^1 + \mathbf{c}^2 \cdot \mathbf{x}_j^2 \leq z \qquad \forall j \in \{1,2\} \\ & \mathbf{a}_i^1(\boldsymbol{\xi}) \cdot \mathbf{x}^1 + \mathbf{a}_i^2(\boldsymbol{\xi}) \cdot \mathbf{x}_j^2 \leq b_i(\boldsymbol{\xi}) \qquad \forall \boldsymbol{\xi} \in \Xi_j, j \in \{1,2\}, i \in \mathcal{I}, \end{aligned} \tag{3.6}$$

which we hope will have a solution that is better than the static policy (that is, $z_{part} < z_{static}$). Let $\mathcal{A} = \bigcup_i \mathcal{A}_i$. We now seek to show that whether this improvement occurs or not depends on the nature of the partitions Ξ_1 and Ξ_2.

Theorem 1. *If $\mathcal{A}$, the set of all active uncertain parameters for the static problem (3.5), satisfies either $\mathcal{A} \subset \Xi_1$ or $\mathcal{A} \subset \Xi_2$ then $z_{part} = z_{static}$. Otherwise $z_{part} \leq z_{static}$.*

Proof of Theorem 1. Suppose that the set of active uncertain parameters $\mathcal{A}$ is a subset of Ξ_1. Consider solving the problem

$$\begin{aligned} z_{half} = \min_{\mathbf{x}\in\mathcal{X},z} \quad & z \\ \text{subject to} \quad & \mathbf{c}^1 \cdot \mathbf{x}^1 + \mathbf{c}^2 \cdot \mathbf{x}_1^2 \leq z \\ & \mathbf{a}_i^1(\boldsymbol{\xi}) \cdot \mathbf{x}^1 + \mathbf{a}_i^2(\boldsymbol{\xi}) \cdot \mathbf{x}_1^2 \leq b_i(\boldsymbol{\xi}) \qquad \forall \boldsymbol{\xi} \in \Xi_1, i \in \mathcal{I}, \end{aligned} \tag{3.7}$$

which can be viewed as relaxation of (3.5) as we must only satisfy the constraints for a subset of Ξ. Despite being a relaxation, we still have $z_{half} = z_{static}$, as every cut with minimal slack in (3.5) will still be a valid cut for (3.7) and thus the solution will be

equal as we have not relaxed any of the constraints that bound the previous solution. Note that this does not preclude the possibility that the equivalent problem to (3.7) for the other partition Ξ_2 has an objective function value better than z_{static}, but as z_{part} is taken to be the maximum of the objective function values corresponding to the two partitions in (3.6) there can be no improvement of objective function value overall (i.e., $z_{half} = z_{part} = z_{static}$). On the other hand, if we have partitions such that $\mathcal{A} \not\subset \Xi_1$ and $\mathcal{A} \not\subset \Xi_2$, then the possibility of improvement exists: at least one zero-slack cut will not be valid for z_{half}, so the solution to (3.7) is not restricted to having the same solution as (3.5). □

This result naturally leads us to consider partitioning schemes that will achieve this required condition, and thus enable improvement. A partitioning scheme that achieves this is one in which every $\hat{\boldsymbol{\xi}} \in \mathcal{A}$ lies in its own partition. A simple way (both conceptually and computationally) to construct such a set of partitions is to use a *Voronoi diagram* (see e.g. Aurenhammer [1991] for an overview). Given a set of K points $\left\{\hat{\boldsymbol{\xi}}_1, \ldots, \hat{\boldsymbol{\xi}}_K\right\} \subset \Xi$ the Voronoi diagram associated with these points defines a partition for each $\hat{\boldsymbol{\xi}}_i$ that contains only the $\boldsymbol{\xi} \in \Xi$ such that the Euclidean distance between $\hat{\boldsymbol{\xi}}_i$ and $\boldsymbol{\xi}$ is less than or equal to the distance to any other given point $\hat{\boldsymbol{\xi}}_j$. We now apply this to the problem at hand: as $\mathcal{A}$ is a finite set we can express the partition induced by active uncertain parameter $\hat{\boldsymbol{\xi}}_i \in \mathcal{A}$ as

$$\begin{aligned}\Xi\left(\hat{\boldsymbol{\xi}}_i\right) &= \Xi \cap \left\{\boldsymbol{\xi} \,\middle|\, \left\|\hat{\boldsymbol{\xi}}_i - \boldsymbol{\xi}\right\|_2 \leq \left\|\hat{\boldsymbol{\xi}}_j - \boldsymbol{\xi}\right\|_2 \quad \forall \hat{\boldsymbol{\xi}}_j \in \mathcal{A},\ \hat{\boldsymbol{\xi}}_i \neq \hat{\boldsymbol{\xi}}_j\right\} \\ &= \Xi \cap \left\{\boldsymbol{\xi} \,\middle|\, \left(\hat{\boldsymbol{\xi}}_j - \hat{\boldsymbol{\xi}}_i\right) \cdot \boldsymbol{\xi} \leq \frac{1}{2}\left(\hat{\boldsymbol{\xi}}_j - \hat{\boldsymbol{\xi}}_i\right) \cdot \left(\hat{\boldsymbol{\xi}}_j + \hat{\boldsymbol{\xi}}_i\right) \quad \forall \hat{\boldsymbol{\xi}}_j \in \mathcal{A},\ \hat{\boldsymbol{\xi}}_i \neq \hat{\boldsymbol{\xi}}_j\right\}\end{aligned}$$

which is a set defined by Ξ and $|\mathcal{A}| - 1$ additional linear constraints (and if Ξ is polyhedral then $\Xi\left(\hat{\boldsymbol{\xi}}_i\right)$ will also be polyhedral).

We will now generalize away from the cutting plane method-based reasoning and continuous problems, and we will assume throughout that the uncertainty set is closed and convex but not necessarily polyhedral. For constraint i and a static policy solution

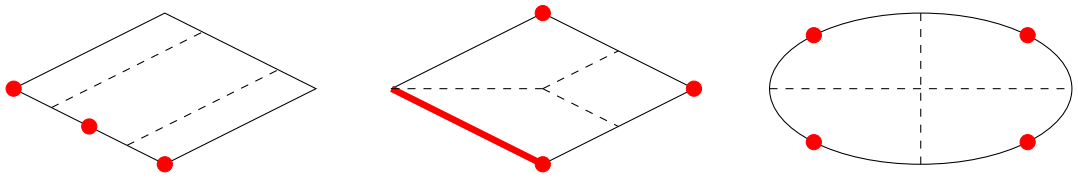

Figure 3-1: Partitioning of simple uncertainty sets using Voronoi diagrams. The first two diagrams use the polyhedral uncertainty set $\Xi_P = \left\{\boldsymbol{\xi} \,\middle|\, \left\|\left[\frac{1}{2}\xi_1, \xi_2\right]\right\|_1 \leq 1\right\}$, and the third uses an ellipsoidal set $\Xi_E = \left\{\boldsymbol{\xi} \,\middle|\, \left\|\left[\frac{1}{2}\xi_1, \xi_2\right]\right\|_2 \leq 1\right\}$. The bold points represent the active uncertain parameters $\hat{\boldsymbol{\xi}}$ used to create the partitions. In the center example the red line represents an infinite set of possible active uncertain parameters, from which we have selected one arbitrarily.

$(\bar{\mathbf{x}}^1, \bar{\mathbf{x}}^2)$ we define the set of active uncertain parameters for constraint i as

$$\mathcal{A}_i = \arg\min_{\boldsymbol{\xi}\in\Xi} \left\{ b_i\left(\boldsymbol{\xi}\right) - \mathbf{a}_i^1\left(\boldsymbol{\xi}\right) \cdot \bar{\mathbf{x}}^1 - \mathbf{a}_i^2\left(\boldsymbol{\xi}\right) \cdot \bar{\mathbf{x}}^2 \right\}, \tag{3.8}$$

which, as the set of optimal solutions for minimizing a linear function over a convex set is itself convex, may be a convex set of infinite cardinality. Note that we do not require that these $\hat{\boldsymbol{\xi}} \in \mathcal{A}_i$ correspond to constraints that have zero slack, as this may possibly never occur for a problem with integer variables. We can also define an equivalent set $\mathcal{A}_c$ for the objective epigraph constraint. The possibility that these sets are of infinite cardinality presents a problem for our proposed Voronoi diagram scheme, as the natural generalization of Voronoi diagrams for convex sets (instead of just points) can produce nonconvex partitions [Lee and Drysdale, 1981]. We propose selecting a single point from each set $\mathcal{A}_i$ using either problem-specific knowledge, or a more general technique such as taking the Chebyshev center or random selection. In Figure 3-1 we demonstrate the partitions that would be induced in a polyhedral uncertainty set for two different sets of uncertain parameters, as well as a partitioning of an ellipsoidal set.

3.2.2 Description of Method

We can now build an iterative method utilizing this partitioning scheme. The method starts by solving a static-policy version of our adaptive optimization problem to de-

termine a set of active uncertain parameters. We use these to construct a finitely-adaptive version of our problem, and solve that. This in turn produces a new set of active uncertain parameters which we can then use to partition further, ideally improving on the previous solution at each iteration.

We have already described how we collect a finite set of active uncertain parameters given a solution. The other key detail is the way we track the active uncertain parameters across iterations, and how we construct the partitions after the first iteration. We propose nested partitions: the active uncertain parameters $\mathcal{A}$ corresponding to the constraints for a partition at iteration k are used to sub-partition that partition at iteration $k+1$. We can describe the partition construction scheme in terms of a *tree* $\mathcal{T}$ of uncertain parameters, for which we will define the following sets: $Leaves\left(\mathcal{T}\right)$ is the set of leaves of the tree $\mathcal{T}$, $Children\left(\hat{\boldsymbol{\xi}}\right)$ is the set of children of $\hat{\boldsymbol{\xi}}$ in the tree $\mathcal{T}$, $Parent\left(\hat{\boldsymbol{\xi}}\right)$ is the parent of $\hat{\boldsymbol{\xi}}$ in the tree $\mathcal{T}$, and finally $Siblings\left(\hat{\boldsymbol{\xi}}\right)$ is the set of children of the parent $\hat{\boldsymbol{\xi}}$, i.e. $Siblings\left(\hat{\boldsymbol{\xi}}\right) = Children\left(Parent\left(\hat{\boldsymbol{\xi}}\right)\right)$. Each leaf of $\mathcal{T}$ corresponds to a partition of the uncertainty set and each level of the tree corresponds to an iteration of the algorithm. At each iteration we construct a partition for a leaf $\hat{\boldsymbol{\xi}}_i$ as an intersection of partitions

$$\begin{aligned}\Xi\left(\hat{\boldsymbol{\xi}}_i\right) = &\left\{\boldsymbol{\xi} \,\middle|\, \left\|\hat{\boldsymbol{\xi}}_i - \boldsymbol{\xi}\right\|_2 \leq \left\|\hat{\boldsymbol{\xi}}_j - \boldsymbol{\xi}\right\|_2 \quad \forall \hat{\boldsymbol{\xi}}_j \in Siblings\left(\hat{\boldsymbol{\xi}}_i\right)\right\} \\ &\cap \left\{\boldsymbol{\xi} \,\middle|\, \left\|Parent\left(\hat{\boldsymbol{\xi}}_i\right) - \boldsymbol{\xi}\right\|_2 \leq \left\|\hat{\boldsymbol{\xi}}_j - \boldsymbol{\xi}\right\|_2 \quad \forall \hat{\boldsymbol{\xi}}_j \in Siblings\left(Parent\left(\hat{\boldsymbol{\xi}}_i\right)\right)\right\} \\ &\vdots \\ &\cap \Xi,\end{aligned} \tag{3.9}$$

which terminates when the parent is the root node, which has no siblings and thus is equivalent to the entire uncertainty set. By adding the active uncertain parameters for partition $\Xi\left(\hat{\boldsymbol{\xi}}_i\right)$ as children of $\hat{\boldsymbol{\xi}}_i$ in $\mathcal{T}$, we create subpartitions of $\Xi\left(\hat{\boldsymbol{\xi}}_i\right)$ at the next iteration as shown in Figure 3-2.

Given this machinery our method for solving problems of the form of (3.3) proceeds as follows:

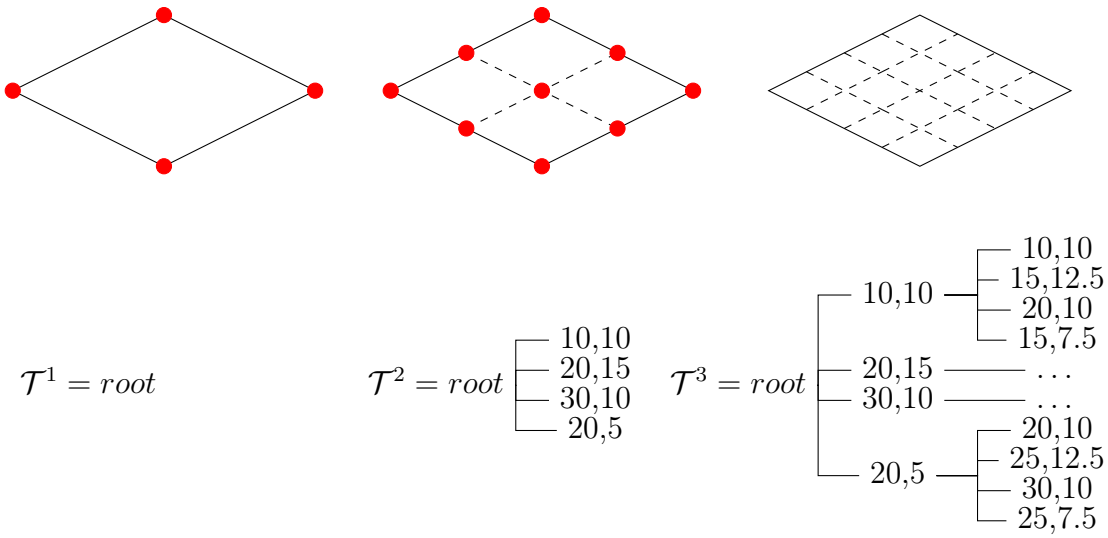

Figure 3-2: Visualization of the partitions and active uncertain parameter tree across multiple iterations. The uncertainty set is $\Xi = \left\{(\xi_1, \xi_2) \left| \frac{|\xi_1 - 20|}{10} + \frac{|\xi_2 - 10|}{5} \leq 1 \right.\right\}$, and we assume that at each iteration all the extreme points of the set/partitions are active uncertain parameters (red points) in the solved partitioned problem.

Partition-and-bound method for two-stage AMIO

1. **Initialize**. Define $\mathcal{T}^1$ to be the initial tree of uncertain parameters that consists of one root node (any $\boldsymbol{\xi} \in \Xi$). Set iteration counter $k \leftarrow 1$

2. **Solve**. Solve the following partitioned version of (3.3), with one partition for every $\hat{\boldsymbol{\xi}}_j \in \mathit{Leaves}\left(\mathcal{T}^k\right)$:

$$
\begin{aligned}
z_{alg}\left(\mathcal{T}^k\right) = \min_{\mathbf{x} \in \mathcal{X}, z} \quad & z \\
\text{subject to} \quad & \mathbf{c}^1(\boldsymbol{\xi}) \cdot \mathbf{x}^1 + \mathbf{c}^2(\boldsymbol{\xi}) \cdot \mathbf{x}_j^2 \leq z && \forall \boldsymbol{\xi} \in \Xi\left(\hat{\boldsymbol{\xi}}_j\right) \\
& \mathbf{a}_i^1(\boldsymbol{\xi}) \cdot \mathbf{x}^1 + \mathbf{a}_i^2(\boldsymbol{\xi}) \cdot \mathbf{x}_j^2 \leq b_i(\boldsymbol{\xi}) && \forall \boldsymbol{\xi} \in \Xi\left(\hat{\boldsymbol{\xi}}_j\right), i \in \mathcal{I} \\
& && \forall \hat{\boldsymbol{\xi}}_j \in \mathit{Leaves}\left(\mathcal{T}^k\right)
\end{aligned}
\tag{3.10}
$$

   where $\Xi\left(\hat{\boldsymbol{\xi}}_j\right)$ is the set defined in (3.9), and $\mathbf{x}_j^2$ is the set of recourse decisions corresponding to the partition induced by parameters $\hat{\boldsymbol{\xi}}_j$.

3. **Grow**. Initialize $\mathcal{T}^{k+1} \leftarrow \mathcal{T}^k$. For every leaf $\hat{\boldsymbol{\xi}}_j \in \mathit{Leaves}\left(\mathcal{T}^{k+1}\right)$, add children to that leaf for each $\hat{\boldsymbol{\xi}}$ in the set of active uncertain parameters $\mathcal{A}$ for the

partition $\Xi\left(\hat{\boldsymbol{\xi}}_j\right)$ and the solution to (3.10), selecting a finite subset if there is an infinite set of active uncertain parameters to choose from.

4. **Bound**. Calculate a lower bound $z_{lower}\left(\mathcal{T}^{k+1}\right)$ (see Section 3.2.3) of the fully adaptive solution, and terminate if the bound gap

$$\frac{\left(z_{alg} - z_{lower}\right)}{|z_{lower}|} \tag{3.11}$$

is less than ϵ_{gap}, or the iteration or time limit has been reached. Otherwise set $k \leftarrow k+1$ and go to Step 2.

A possibility not directly addressed by this method is the case where the initial static solution is infeasible. For example, consider the feasibility problem

$$\begin{aligned} z_{feas} = \min_{x^2} \quad & 0 \\ \text{subject to} \quad & x^2(\xi) \leq \xi + \epsilon \quad \forall \xi \in [0,1] \\ & x^2(\xi) \geq \xi - \epsilon \quad \forall \xi \in [0,1], \end{aligned} \tag{3.12}$$

which has no feasible static policy (but does have the solution $x^2(\xi) = \xi$, as well as piecewise constant solutions). If infeasibility does occur, then one potential remedy is to use problem-specific knowledge to identify a near-feasible solution or to initially relax problematic constraints so that active uncertain parameters can be collected for partitioning. Failing that, sampling (random or otherwise) of the uncertainty set can serve the same purpose. Once a feasible solution is identified the algorithm can proceed as before.

Our method is similar to that of Postek and Den Hertog [2014] for the case of two-stage problems. The primary difference is the method of constructing partitions: while our method creates multiple child partitions at each iteration, the method of Postek and Den Hertog [2014] creates only two child partitions per iteration per partition. These two child partitions are defined by a single separating hyperplane

between a pair of active uncertain parameters, with the primary selection heuristic employed in that paper being to choose the pair with the greatest distance between them. We will explore the implications of this difference further in the Section 3.3 and in the computational experiments.

We also note that, while we have focused here on mixed-integer linear optimization, other problem classes can be solved using the same approach. For example, we could consider quadratic terms in the objective, or second-order cone constraints. The only requirements we have for this method are the ability to extract active uncertain parameters after solving a partitioned problem in order to further improve the solution at the next iteration, and the ability to partition the uncertainty set.

3.2.3 Calculating Lower Bounds

Our method ideally produces successfully closer approximations to the fully adaptive problem (3.3). The most natural termination criterion then, apart from an iteration- or time-limit criterion, is to terminate when our approximation (an upper bound, as we are minimizing) is sufficiently close to the fully adaptive solution. As we do not know the fully adaptive solution, we propose using the set of uncertain parameters available at the end of iteration k to obtain a lower bound. That is, after the solution of the partitioned problem at iteration k, we grow our tree $\mathcal{T}$ and use all parameters in $\mathcal{T}$ to construct and solve a deterministic problem whose objective provides the lower bound. This is essentially the same as the "scenario based bound" of Hadjiyiannis et al. [2011], which also utilizes "binding" uncertainty realizations, much like our active uncertain parameters. We restate this bound here in the context of our method.

Proposition 1. *Consider the solution of the deterministic optimization problem*

$$
\begin{aligned}
z_{lower}\left(\mathcal{T}\right) = \min_{\mathbf{x}\in\mathcal{X},z} \quad & z & & (3.13)\\
\textit{subject to} \quad & \mathbf{c}^1\left(\hat{\boldsymbol{\xi}}_j\right)\cdot\mathbf{x}^1 + \mathbf{c}^2\left(\hat{\boldsymbol{\xi}}_j\right)\cdot\mathbf{x}_j^2 \leq z & \forall\hat{\boldsymbol{\xi}}_j \in \mathcal{T} & \\
& \mathbf{a}_i^1\left(\hat{\boldsymbol{\xi}}_j\right)\cdot\mathbf{x}^1 + \mathbf{a}_i^2\left(\hat{\boldsymbol{\xi}}_j\right)\cdot\mathbf{x}_j^2 \leq b_i\left(\hat{\boldsymbol{\xi}}_j\right) & \forall\hat{\boldsymbol{\xi}}_j \in \mathcal{T}, i\in I &
\end{aligned}
$$

where $\mathcal{T}$ is a set of uncertain parameters $\hat{\boldsymbol{\xi}} \in \Xi$. Then $z_{lower}(\mathcal{T}) \leq z_{full}$, where z_{full} is the solution to (3.3).

Proof. Proof of Proposition 1 Problem (3.13) is a relaxation of (3.3) as we only require the solution to satisfy the constraints for a finite subset of uncertain parameters. Any solution to the fully adaptive problem (3.3) has a trivial mapping to a feasible solution for (3.13) that preserves the objective function value, so $z_{lower}(\mathcal{T}) \leq z_{full}$. □

A useful property of this bound is that as the tree grows the lower bound will tend to approach the fully adaptive solution. At the same time we are improving our upper bound by decreasing the size of each partition, and thus closing the bound gap from both sides. Care must be taken if solving (3.13) approximately: for example if (3.13) is a MIO problem and we allow the solver to terminate when the gap between the best known integer solution and the continuous relaxation falls below a tolerance, as is common in practice, we will obtain a solution that is larger than the true lower bound or even the fully adaptive solution and thus understate the bound gap.

3.2.4 Worked Example: Two-stage Inventory Control Problem

Example 1. We now demonstrate our approach by solving a two-stage inventory control problem. We must satisfy all of the uncertain demand, and there are three ways to order stock to do so: ordering any amount here-and-now at a unit cost of 50, ordering a fixed "lot" of size 25 at a unit cost of 60, and ordering a fixed lot of size 25 at a unit cost of 75. Only one lot of each size may be ordered, but the decision to order them can be made after the demand is known. We must pay a high unit cost of 65 for the storage and disposal of any of the remaining stock after the demand is

satisfied. We can formulate this as a two-stage AMIO problem

$$\begin{aligned} \min \quad & z & & (3.14)\\ \text{subject to} \quad & 50x^1 + 65I^2(\xi) + 1500y_A^2(\xi) + 1875y_B^2(\xi) \leq z & \forall \xi \in \Xi \\ & I^2(\xi) \geq 0 & \forall \xi \in \Xi \\ & x^1 \geq 0,\ y_A^2(\xi), y_B^2(\xi) \in \{0,1\} & \forall \xi \in \Xi, \end{aligned}$$

where x^1 is the here-and-now continuous ordering decision, $y_A^2(\xi)$ and $y_B^2(\xi)$ are the wait-and-see binary ordering decisions, $I^2(\xi) = x^1 - \xi + 25y_A^2(\xi) + 25y_B^2(\xi)$, and ξ is the uncertain demand that is drawn from the uncertainty set $\Xi = \{\xi \,|5 \leq \xi \leq 95\}$. We will initialize our tree $\mathcal{T}^1$ (step 1) with a root $\hat{\xi} = 50$, leading to a problem with a single partition (the entire set) and with y_A^2 and y_B^2 unable to vary with ξ (static). Solving this single-partition version of (3.14) (step 2) gives us the initial solution $z = 10600$, $x^1 = 95$, $y_A^2 = 0$, and $y_B^2 = 0$. The active uncertain parameter value for the objective constraint is when the demand is low ($\hat{\xi} = 5$, which leaves 90 units of stock) and the worst-case for the non-negative inventory constraint is when the demand is high ($\hat{\xi} = 95$, leaving no stock). We grow the tree (step 3) to obtain $\mathcal{T} = \{50 : \{5, 95\}\}$, which we then use to determine a lower bound on the fully adaptive solution ($z_{lower} = 5625$) for an initial gap of approximately 88%.

We now solve the two-partition version of (3.14) at iteration 2. The first partition is

$$\begin{aligned} \Xi\left(\hat{\xi}_1 = 5\right) &= \{\xi \,|5 \leq \xi \leq 95\} \cap \{\xi \,|\|5 - \xi\|_2 \leq \|95 - \xi\|_2\} \\ &= \{\xi \,|5 \leq \xi \leq 50\} \end{aligned}$$

and the second is $\Xi\left(\hat{\xi}_2 = 95\right) = \{\xi \,|50 \leq \xi \leq 95\}$. The objective function value for this problem is substantially lower ($z = 7925$) than at iteration 1 and the stock bought initially is lower ($x^1 = 70$). For the first partition we will order neither lot ($y_{A,1}^2 = y_{B,1}^2 = 0$), but if the demand falls in the second partition we will purchase the first, cheaper lot ($y_{A,2}^2 = 1$, $y_{B,2}^2 = 0$). Once again the active un-

certain parameters lie at the extremes of the partitions, which leads to the tree $\mathcal{T}^3 = \{50 : \{5 : \{5, 50\}, 95 : \{50, 95\}\}\}$. Our lower bound is unchanged, as we were already using the values 5, 50, and 95, but our gap shrinks substantially to 41%, thanks to the reduced upper bound.

We now begin our third and final iteration, by solving the partitioned problem induced by $\mathcal{T}^3$. In the interest of space we only detail the construction of the partition for the leaf $\hat{\xi} = 50$ that is the child of $\hat{\xi} = 5$:

$$\begin{aligned}\Xi\left(\hat{\xi} = 50\right) &= \Xi \cap \{\xi \,|\|5 - \xi\|_2 \leq \|95 - \xi\|_2\} \cap \{\xi \,|\|50 - \xi\|_2 \leq \|5 - \xi\|_2\} \\ &= \Xi \cap \{\xi \,|\xi \leq 50\} \cap \left\{\xi \,\middle|\, \xi \geq 27\frac{1}{2}\right\} \\ &= \left\{\xi \,\middle|\, 27\frac{1}{2} \leq \xi \leq 50\right\}.\end{aligned}$$

The other three partitions are $\left\{\xi \,\middle|\, 5 \leq \xi \leq 27\frac{1}{2}\right\}$, $\left\{\xi \,\middle|\, 50 \leq \xi \leq 72\frac{1}{2}\right\}$, and $\left\{\xi \,\middle|\, 72\frac{1}{2} \leq \xi \leq 95\right\}$. Solving this four-partition problem produces the best solution yet ($z = 7375$). The solution purchases substantially less stock up front ($x^1 = 47\frac{1}{2}$), and we can best describe the values of y_A and y_B as piecewise constant functions of ξ, i.e.,

$$y_A^2(\xi) = \begin{cases} 0, & 5 \leq \xi < 72\frac{1}{2}, \\ 1, & 72\frac{1}{2} \leq \xi \leq 95, \end{cases} \tag{3.15}$$

and

$$y_B^2(\xi) = \begin{cases} 0, & 5 \leq \xi < 27\frac{1}{2}, \\ 1, & 27\frac{1}{2} \leq \xi \leq 95, \end{cases} \tag{3.16}$$

respectively. Growing the tree with the new active uncertain parameters leads to a new lower bound of $z_{lower} = 5912\frac{1}{2}$, resulting in a final gap of 26%. Although it cannot be determined at this iteration, our upper bound is very good as the fully adaptive solution has an objective of $z_{full} = 7250$ (as determined by exhaustively applying this method).

3.2.5 Incorporating Affine Adaptability

In the introduction we discussed the popularity and success of affine adaptability, also known as linear decisions rules. We can express this type of adaptability in the context of our two-stage problem (3.3) with the substitution

$$\mathbf{x}^2(\boldsymbol{\xi}) = \mathbf{F}\boldsymbol{\xi} + \mathbf{g}, \tag{3.17}$$

where $\mathbf{F}$ and $\mathbf{g}$ are now the decision variables instead of $\mathbf{x}^2$ itself. There are two problems with substituting (3.17) directly into (3.3). The first is that if $\mathbf{a}_i^2(\boldsymbol{\xi})$ and $\mathbf{c}^2(\boldsymbol{\xi})$ are not constant with respect to the uncertain parameters then we will have quadratic products of uncertain parameters, which is not generally tractable [Ben-Tal et al., 2004]. The second problem is that if a variable $x_j^2(\boldsymbol{\xi})$ is integer then $\mathbf{F}_j$ must necessarily be zero for any non-trivial uncertainty set and so there is no adaptability.

We will thus separate our second stage decisions into two sets: continuous decisions $\mathbf{x}^2$ and integer decisions $\mathbf{y}^2$, and apply our affine substitution only to the continuous decisions. This leads to an affine adaptability form of the fully adaptive problem (3.3):

$$\begin{aligned}
\min_{\mathbf{F},\mathbf{g},\mathbf{x}^1,\mathbf{y},z} \quad & z & & (3.18)\\
\text{subject to} \quad & \mathbf{c}^1(\boldsymbol{\xi})\cdot\mathbf{x}^1 + \mathbf{c}_x^2\cdot(\mathbf{F}\boldsymbol{\xi}+\mathbf{g}) + \mathbf{c}_y^2(\boldsymbol{\xi})\cdot\mathbf{y}^2(\boldsymbol{\xi}) \le z & \forall \boldsymbol{\xi}\in\Xi & \\
& \mathbf{a}_i^1(\boldsymbol{\xi})\cdot\mathbf{x}^1 + \mathbf{a}_{x,i}^2\cdot(\mathbf{F}\boldsymbol{\xi}+\mathbf{g}) + \mathbf{a}_{y,i}^2(\boldsymbol{\xi})\cdot\mathbf{y}^2(\boldsymbol{\xi}) \le b_i(\boldsymbol{\xi}) & \forall \boldsymbol{\xi}\in\Xi,\ i\in\mathcal{I} & \\
& \mathbf{F}\in\mathbb{R}^{n_x\times n_\xi}, \mathbf{g}\in\mathbb{R}^{n_x}, \mathbf{y}\in\mathbb{Z}^{n_y}. &
\end{aligned}$$

A key observation with respect to our finite partitioning method is that $\mathbf{F}$ and $\mathbf{g}$ are themselves decisions that can be deferred until the uncertainty has been realized, and so they too are second stage variables. This allows us to combine finite adaptability and affine adaptability, as we can associate a different affine policy $(\mathbf{F}_j, \mathbf{g}_j)$ for each partition. Doing so produces a piecewise linear policy: for example, for two partitions

we would have

$$\mathbf{x}^2(\boldsymbol{\xi}) = \begin{cases} \mathbf{F}_1\boldsymbol{\xi} + \mathbf{g}_2, & \boldsymbol{\xi} \in \Xi\left(\hat{\boldsymbol{\xi}}_1\right), \\ \mathbf{F}_2\boldsymbol{\xi} + \mathbf{g}_2, & \boldsymbol{\xi} \in \Xi\left(\hat{\boldsymbol{\xi}}_2\right), \end{cases} \tag{3.19}$$

for continuous second stage decisions, and a piecewise constant policy

$$\mathbf{y}^2(\boldsymbol{\xi}) = \begin{cases} \mathbf{y}_1^2, & \boldsymbol{\xi} \in \Xi\left(\hat{\boldsymbol{\xi}}_1\right), \\ \mathbf{y}_2^2, & \boldsymbol{\xi} \in \Xi\left(\hat{\boldsymbol{\xi}}_2\right), \end{cases} \tag{3.20}$$

for integer decisions. We demonstrate combining affine adaptability with finite adaptability in our mulitstage lot sizing computational example in Section 3.5.2, including a comparison against the method of Postek and Den Hertog [2014] which can be used to create a similar combination.

3.3 Analysis of the Partition-and-Bound Method

In this section, we characterize the partition-and-bound method proposed above from a theoretical point of view. In particular, we seek to understand how the algorithm approaches the ideal fully adaptive solution, and to what degree the number of partitions grows at each iteration.

3.3.1 Convergence Properties

We now aim to understand how $z_{alg}\left(\mathcal{T}^k\right)$ varies with the iteration count k.

Proposition 2. *The upper bound $z_{alg}\left(\mathcal{T}^k\right)$ will never increase as k increases.*

Proof. Proof of Proposition 2 This follows from the "nested" nature of our partitions. Consider the solution of the partitioned problem at iteration k, e.g. $\bar{\mathbf{x}}^1, \bar{\mathbf{x}}_1^2, \bar{\mathbf{x}}_2^2, \ldots$. At iteration $k+1$ we will further subpartition each of these partitions, resulting in new second stage variables. We can construct a feasible solution for iteration $k+1$ by setting the value of each of these new variables for each subpartition to the same value as the decision variables associated with their respective parent partition. This

new solution is feasible by construction and has the same objective value, so we will at least match the objective of iteration k at iteration $k+1$. $\square$

This property has computational implications, as it allows us to provide an initial feasible solution at iteration $k+1$ using the solution at iteration k. This can allow integer optimization solvers to truncate portions of the search tree faster, and reach lower integrality gaps earlier.

We have established that we will never get worse, and it is also possible to identify families of problems (in particular, problems with only continuous decision variables) where we will smoothly converge to the fully adaptive solution. Instead of identifying these, we instead provide the following counterexample that demonstrates that there does not exist a general guarantee that we will improve the bound gap for any finite number of iterations, even if we start at far from the lower bound, for linear AMIO problems. To demonstrate this, consider the parametric family of problems

$$\begin{aligned} z(\epsilon) = \min_{x^2 \in \{0,1\}, y^2 \in \{0,1\}, z} \quad & z & & \\ \text{subject to} \quad & x^2(\xi) + y^2(\xi) \leq z & \forall \xi \in [0,1] & \\ & x^2(\xi) \geq \frac{\epsilon - \xi}{\epsilon} & \forall \xi \in [0,1] & \\ & y^2(\xi) \geq \frac{-\epsilon + \xi}{\epsilon} & \forall \xi \in [0,1], & \end{aligned} \tag{3.21}$$

where $\epsilon \in [0,1]$. This problem has a fully adaptive solution of $z = 1$ and

$$x^2(\xi) = \begin{cases} 1, & 0 \leq \xi \leq \epsilon, \\ 0, & \epsilon < \xi \leq 1, \end{cases} \qquad y^2(\xi) = \begin{cases} 0, & 0 \leq \xi \leq \epsilon, \\ 1, & \epsilon < \xi \leq 1. \end{cases}$$

Note that the static policy sets both $x^2 = y^2 = 1$, and thus $z = 2$. The initial bound gap is 1 as any random $\hat{\xi}$ used to calculate the lower bound will correctly identify it to be $z_{lower} = 1$. Both variables share the same "breakpoint" of ϵ, which means a simple two-partition finitely adaptive solution could match the fully adaptive solution, if only such a partition could be identified.

Proposition 3. *The partition-and-bound method will never find a solution equivalent to the fully adaptive solution for* (3.21)*, or decrease the bound gap, unless* $\epsilon = q \times 2^{-p}$ *for some* $p, q \in \mathbb{Z}^+$.

Proof of Proposition 3. First note that the two extreme points of each partition are always the active uncertain parameters (the low end from the second constraint, and the high end from the third). Thus at the beginning of iteration k we will have 2^{k-1} partitions of width 2^{1-k}. For all partitions $\Xi(\xi) = [a, b]$ such that $b < \epsilon$ the corresponding decision variable values are $x^2 = 1$ and $y^2 = 0$, with the corresponding lower bound of 1 on the objective z. For all partitions such $a > \epsilon$ we have decision variable values $x^2 = 0$ and $y^2 = 1$, also with the corresponding lower bound of 1 on the objective z. The remaining partition then has the property that $a \leq \epsilon \leq b$. If the inequalities are strict, then we must set both $x^2 = y^2 = 1$, and thus do not improve over the static policy. However if $\epsilon = a$ or $\epsilon = b$ then we can match the fully adaptive solution because the "breakpoint" is correct. As a and b are always of the form $q \times 2^{-p}$ for nonnegative integers p and q, this condition can only be satisfied if ϵ is also drawn from this family. Any other choice of ϵ, such as $1/3$, will never be matched for any finite number of iterations. □

This result also applies to the method of Postek and Den Hertog [2014], using similar reasoning. While this result is a negative one, we believe that such structures are uncommon in problems of interest, and may be avoidable by making different modeling choices when they do occur.

3.3.2 Growth in Partitioning Scheme

For a problem with m uncertain constraints and an uncertain objective the number of partitions with our method is at most $(m+1)^{k-1}$ at iteration k (and may be much lower if the same uncertain parameter is active for multiple constraints). In contrast, the proposal of Postek and Den Hertog [2014] will have at most 2^{k-1}, but may require more iterations to achieve a solution of the same quality. Here we show by example

that the growth rate of the size of the partitioning scheme is of secondary concern to the relationship between computational effort and quality of solution for many cases.

Consider the following example problem, which was presented in Bertsimas and Goyal [2010]:

$$\begin{aligned} z(n) = \min_{\mathbf{x}^2 \geq 0, z} \quad & z \\ \text{subject to} \quad & \mathbf{e} \cdot \mathbf{x}^2(\boldsymbol{\xi}) \leq z \quad \forall \boldsymbol{\xi} \in \Xi \\ & \mathbf{x}^2(\boldsymbol{\xi}) \geq \boldsymbol{\xi} \quad \forall \boldsymbol{\xi} \in \Xi, \end{aligned} \tag{3.22}$$

where $\Xi = \{\boldsymbol{\xi} \,|\sum_{i=1}^{n} \xi_i \leq 1, \boldsymbol{\xi} \geq 0\}$. The initial static policy has an objective value of n compared to the fully adaptive policy objective value of 1. The upper bound provided by our method in the second iteration is $\frac{n+1}{2}$, which is obtained using n partitions. On the other hand, the method of Postek and Den Hertog [2014] will have 2^{k-1} partitions at iteration k. If we assume that the time to solve this problem is proportional to the number of partitions then in the same time it takes to solve the second iteration for our method (n time units), we can do approximately $\log_2(n)$ iterations of Postek and Den Hertog [2014]'s method. However we can show (Appendix A.1) that it requires n iterations of that method (and thus 2^{n-1}) partitions to achieve the same objective as our method, demonstrating the need to focus on the solution quality versus time rather than iteration or partition counts.

In the above example all partitions had the same objective contribution, i.e., the overall objective was equal to the objective value for each partition. This is not necessarily always the case, and it is trivial to find cases where this doesn't occur. Recall that the motivation for our method, as presented in Theorem 1, was to identify active uncertain parameters that were "binding" the solution and to separate them. A similar approach can be taken with partitions themselves: if a partition's objective is not restricting the overall objective (the partition is not *active*), then we will not sub-partition it at the next iteration. To identify these partitions we need to encourage a solver to find the best possible solution for each partition. We can achieve this by

adding a very small term to the objective, e.g.,

$$
\begin{aligned}
z_{alg}\left(\mathcal{T}^k\right) = \min_{\mathbf{x}\in\mathcal{X},z} \quad & z + \epsilon \sum_j \tilde{z}_j \\
\text{subject to} \quad & \tilde{z}_j \leq z \\
& \mathbf{c}^1(\boldsymbol{\xi})\cdot\mathbf{x}^1 + \mathbf{c}^2(\boldsymbol{\xi})\cdot\mathbf{x}_j^2 \leq \tilde{z}_j \qquad \forall \boldsymbol{\xi} \in \Xi\left(\hat{\boldsymbol{\xi}}_j\right) \\
& \mathbf{a}_i^1(\boldsymbol{\xi})\cdot\mathbf{x}^1 + \mathbf{a}_i^2(\boldsymbol{\xi})\cdot\mathbf{x}_j^2 \leq b_i(\boldsymbol{\xi}) \qquad \forall \boldsymbol{\xi} \in \Xi\left(\hat{\boldsymbol{\xi}}_j\right), i \in \mathcal{I} \\
& \qquad\qquad\qquad\qquad\qquad\qquad \forall \hat{\boldsymbol{\xi}}_j \in \mathit{Leaves}\left(\mathcal{T}^k\right),
\end{aligned}
\tag{3.23}
$$

where $\tilde{z}_j$ is the objective for partition j and we say a partition is *active* if $\tilde{z}_j = z$.

Consider the following variation on (3.22) that weights one of the components of the decision above the rest:

$$
\begin{aligned}
z(M,n) = \min_{\mathbf{x}^2\geq 0,z} \quad & z \\
\text{subject to} \quad & (M-1)x_1^2 + \mathbf{e}\cdot\mathbf{x}^2(\boldsymbol{\xi}) \leq z \qquad \forall \boldsymbol{\xi} \in \Xi \\
& \mathbf{x}^2(\boldsymbol{\xi}) \geq \boldsymbol{\xi} \qquad \forall \boldsymbol{\xi} \in \Xi,
\end{aligned}
\tag{3.24}
$$

where the uncertainty set is the same as in the previous example. The static policy now has an objective value of $M+n-1$ compared to the fully adaptive solution's M. If we now apply our partitioning scheme once, the partition corresponding to $i = 1$ will have objective $M + \frac{n-1}{2}$ and all other partitions will have the smaller objective $\frac{M+n}{2}$. At the next iteration this is further reduced to $M + \frac{n-1}{4}$ for the active partition and $\frac{M+n+2}{4}$ for all other partitions. Consider the specific case of $M = n = 10$: the initial static solution has objective 19. After partitioning we have one partition with objective 14.5 and the remainder have objective 10, which is equal to the fully adaptive objective. Thus, we should only add children parameters for the active partition, as further subpartitions of the inactive partitions will not have any further effect on the overall objective. If we continue with further iterations we find that the number of partitions grows linearly with the number of iterations in this case. We implement this approach in our computational experiments in Section 3.5.

3.4 Partition-and-Bound Method for Multistage AMIO

We now generalize our partitioning method described above for two-stage AMIO to the full multistage case of (3.1). In the interests of clarity, the order of events in our model is as follows: The here-and-now decision $\mathbf{x}^1$ are made before any uncertainty is realized, and then the first stage uncertain parameters $\boldsymbol{\xi}^1$ are revealed. We make the decision $\mathbf{x}^2$ with the benefit of knowing $\boldsymbol{\xi}^1$, but with no other knowledge of the uncertainty parameters to be revealed later except what is implied by knowing $\boldsymbol{\xi}^1$. This then proceeds until we make the decision $\mathbf{x}^T$ with the benefit of knowing $\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^{T-1}$, and the final uncertain parameters $\boldsymbol{\xi}^T$ are then revealed (if applicable). Dependencies between uncertain parameters both within and across time stages can be modeled in the uncertainty set with constraints.

While this model of decision making across time is simple enough to state, modeling these *nonanticipativity restrictions* - that a decision made now cannot be made by using exact knowledge of the future - is the primary complication that we address in this section as we extend our approach to the multistage case. We note that enforcing these restrictions is trivial for some classes of adaptability. For example, for affine adaptive policies we can restrict the policy to be a function of only the appropriate components of $\boldsymbol{\xi}$. It is less obvious how to proceed with finite adaptability: The following example demonstrates how a naive application of the method developed in Section 3.2 will either produce overly-optimistic solutions if we fail to recognize a need for nonanticipativity constraints, or will produce pessimistic solutions where the choice of partitioning scheme and nonanticipativity constraints combine to result in less adaptability than we may like.

Example 2. Consider a three-stage ($T = 3$) version of the inventory problem of Section 3.2.4. The problem proceeds exactly as before, except there are two rounds of demand, two rounds of continuous order decisions, and two rounds of lot ordering decisions. The holding cost is paid on the stock remaining after both rounds of

demand and all parameters are the same as before, resulting in the three-stage AMIO

$$\begin{aligned}
\min \quad & z \\
\text{subject to} \quad & 50x^1 + 65I^2\left(\xi^1\right) + 1500y_A^2\left(\xi^1\right) + \\
& 1875y_B^2\left(\xi^1\right) + 50x^2\left(\xi^1\right) + 65I^3\left(\xi^1,\xi^2\right) + \\
& 1500y_A^3\left(\xi^1,\xi^2\right) + 1875y_B^3\left(\xi^1,\xi^2\right) \leq z \qquad \forall \boldsymbol{\xi} \in \Xi \\
& I^2\left(\xi^1\right), I^3\left(\xi^1,\xi^2\right), x^1, x^2\left(\xi^1\right) \geq 0 \qquad \forall \boldsymbol{\xi} \in \Xi \\
& y_A^2\left(\xi^1\right), y_B^2\left(\xi^1\right), y_A^3\left(\xi^1,\xi^2\right), y_B^3\left(\xi^1,\xi^2\right) \in \{0,1\} \quad \forall \boldsymbol{\xi} \in \Xi,
\end{aligned} \tag{3.25}$$

where

$$\begin{aligned}
I^2\left(\xi^1\right) &= x^1 - \xi^1 + 25y_A^2\left(\xi^1\right) + 25y_B^2\left(\xi^1\right), \\
I^3\left(\xi^1,\xi^2\right) &= I^2\left(\xi^1\right) + x^2\left(\xi^1\right) - \xi^2 + 25y_A^3\left(\xi^1,\xi^2\right) + 25y_B^3\left(\xi^1,\xi^2\right),
\end{aligned}$$

and $\Xi = \{5 \leq \xi^1, \xi^2 \leq 95\}$. We begin by solving the static policy version of this problem (no partitions), and obtain a solution with objective value 27050 and three active uncertain parameters: $\hat{\boldsymbol{\xi}}_1 = (5,5)$ (for the objective constraint), $\hat{\boldsymbol{\xi}}_2 = (95,5)$ (for the non-negativity constraint on I^2) and $\hat{\boldsymbol{\xi}}_3 = (95,95)$ (for the non-negativity constraint on I^3). We now construct partitions exactly as before, ignoring the multistage nature of the uncertain parameters, resulting in the three partitions

$$\begin{aligned}
\Xi\left(\hat{\boldsymbol{\xi}}_1 = (5,5)\right) &= \left\{\boldsymbol{\xi} \,\middle|\, 5 \leq \xi^1 \leq 50, \xi^1 + \xi^2 \leq 100\right\}, \\
\Xi\left(\hat{\boldsymbol{\xi}}_2 = (95,5)\right) &= \left\{\boldsymbol{\xi} \,\middle|\, 50 \leq \xi^1 \leq 95, 5 \leq \xi^2 \leq 50\right\}, \\
\Xi\left(\hat{\boldsymbol{\xi}}_3 = (95,95)\right) &= \left\{\boldsymbol{\xi} \,\middle|\, 5 \leq \xi^1 \leq 95, \xi^1 + \xi^2 \geq 100\right\},
\end{aligned}$$

which are visualized in Figure 3-3. As this is a multistage problem we now consider nonanticipativity. We proceed by creating a set of variables for each partition, for example x_1^2, x_2^2, and x_3^2 for the second-stage decision $x^2\left(\xi^1\right)$. Recall that the interpretation of finite adaptability is that if the realized value of $\boldsymbol{\xi}$ falls in $\Xi\left(\hat{\boldsymbol{\xi}}_1\right)$, then we will select x_1^2 as the implemented decision, and similarly for the other partitions.

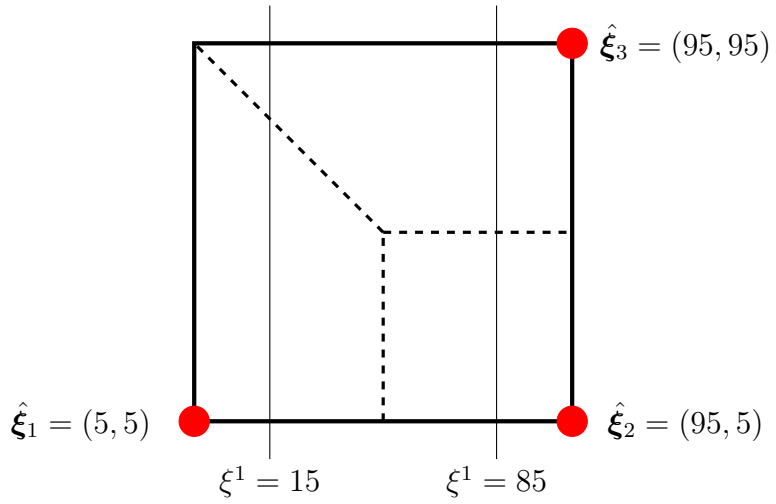

Figure 3-3: Partitioning of the uncertainty set in Example 2 using the two-stage partitioning scheme without any adjustments for the multistage case. The thin solid lines for $\xi^1 = 15$ and $\xi^1 = 85$ demonstrate that knowing both components ξ^1 and ξ^2 is required to fully identify a partition, requiring the addition of nonanticipativity constraints that remove many of the benefits of partitioning scheme.

With this in mind, we consider the case when $\xi^1 = 15$. This value of ξ^1 may fall in either $\Xi\left(\hat{\boldsymbol{\xi}}_1\right)$ or $\Xi\left(\hat{\boldsymbol{\xi}}_3\right)$, but we cannot determine which decision to implement without knowing ξ^2, which is yet to occur. We must then add the nonanticipativity restriction that $x_1^2 = x_3^2$. Similar logic applies to $\xi^1 = 85$, which may be in either $\Xi\left(\hat{\boldsymbol{\xi}}_2\right)$ or $\Xi\left(\hat{\boldsymbol{\xi}}_3\right)$, which requires us to add the restriction $x_2^2 = x_3^2$. These restrictions combine to leave us with no adaptability at all in our second-stage decisions ($x_1^2 = x_2^2 = x_3^2$). The third-stage variables are not affected by these problems as, by that time, we know both ξ^1 and ξ^2 and can thus unambiguously identify the partition and the decision we will implement.

3.4.1 Multistage Partitions and Nonanticipativity Constraints

A naive application of our existing partitioning scheme will result in overly conservative solutions after the application of nonanticipativity constraints. We thus seek a minimal modification of our existing scheme that retains as much of the benefit of partitioning as possible when these constraints are enforced. We will again have a partition for each leaf of our tree $\mathcal{T}$, and we will use the same tree operators and

terminology we established previously. The changes are in how we convert the leaves into partitions, and are coupled with how we identify the required nonanticipativity constraints.

Our scheme is very similar to the two-stage version presented previously, but with restrictions at each layer of the tree to only certain elements of the uncertain parameters. In particular, the partition induced by a leaf $\hat{\boldsymbol{\xi}}_i$ is defined to be

$$\begin{aligned}\Xi\left(\hat{\boldsymbol{\xi}}_i\right) = \quad & \left\{\boldsymbol{\xi} \;\middle|\; \left\|\hat{\boldsymbol{\xi}}_i^{t_{i,j}} - \boldsymbol{\xi}^{t_{i,j}}\right\|_2 \leq \left\|\hat{\boldsymbol{\xi}}_j^{t_{i,j}} - \boldsymbol{\xi}^{t_{i,j}}\right\|_2 \quad \forall \hat{\boldsymbol{\xi}}_j \in Siblings\left(\hat{\boldsymbol{\xi}}_i\right)\right\} \\ & \cap \left\{\boldsymbol{\xi} \;\middle|\; \left\|Parent\left(\hat{\boldsymbol{\xi}}_i\right)^{t'_{i,j}} - \boldsymbol{\xi}^{t'_{i,j}}\right\|_2 \right. \\ & \qquad \left. \leq \left\|\hat{\boldsymbol{\xi}}_j^{t'_{i,j}} - \boldsymbol{\xi}^{t'_{i,j}}\right\|_2 \quad \forall \hat{\boldsymbol{\xi}}_j \in Siblings\left(Parent\left(\hat{\boldsymbol{\xi}}_i\right)\right)\right\} \\ & \cdots \cap \Xi.\end{aligned}$$

We define $t_{i,j}$ for a pair of uncertain parameters $\hat{\boldsymbol{\xi}}_i$ and $\hat{\boldsymbol{\xi}}_j$ as $\arg\min_t \left\{\hat{\boldsymbol{\xi}}_i^t \neq \hat{\boldsymbol{\xi}}_j^t\right\}$, that is the first time stage for which the uncertain parameters differ. $t'_{i,j}$ is the same but for the one level up the tree, i.e., the first time stage t such that $Parent\left(\hat{\boldsymbol{\xi}}_i\right)^t \neq \hat{\boldsymbol{\xi}}_j^t$ for $\hat{\boldsymbol{\xi}}_j^t \in Siblings\left(Parent\left(\hat{\boldsymbol{\xi}}_i\right)\right)$, and so on.

We must also present a complementary scheme for enforcing nonanticipativity. We claim the following for two partitions of the uncertainty set $\Xi\left(\hat{\boldsymbol{\xi}}_i\right)$ and $\Xi\left(\hat{\boldsymbol{\xi}}_j\right)$ induced by the leaf uncertain parameters $\hat{\boldsymbol{\xi}}_i$ and $\hat{\boldsymbol{\xi}}_j$.

Proposition 4. *If there exists* $\boldsymbol{\psi} = \left(\boldsymbol{\psi}^1, \ldots, \boldsymbol{\psi}^T\right) \in \Xi\left(\hat{\boldsymbol{\xi}}_i\right)$ *and* $\boldsymbol{\phi} = \left(\boldsymbol{\phi}^1, \ldots, \boldsymbol{\phi}^T\right) \in \Xi\left(\hat{\boldsymbol{\xi}}_j\right)$ *such that* $\boldsymbol{\psi}^s = \boldsymbol{\phi}^s \; \forall s \in \{1, \ldots, t-1\}$*, and at least one of* $\boldsymbol{\psi} \in int\left(\Xi\left(\hat{\boldsymbol{\xi}}_i\right)\right)$ *and* $\boldsymbol{\phi} \in int\left(\Xi\left(\hat{\boldsymbol{\xi}}_j\right)\right)$ *holds, then we must enforce nonanticipativity constraints for the corresponding decisions at time stage* t *as the two partitions can not be distinguished with the uncertain parameters realized by that time stage. Otherwise, we do not need to enforce any restrictions at time stage* t *for this pair.*

Proof of Proposition 4. If the condition holds, then there exists a partial realization of the uncertain parameters which could lie in two different partitions, based on known information, so we must restrict the decisions for those partitions to be the same. If the equality condition holds, but only for a point on the boundary of both partitions,

then both decisions are valid and interchangeable and we gain no benefit from future knowledge separating the two anyway. If the equality condition does not hold then the partitions are distinguishable and no constraint is required. □

Given this condition, we can now propose a simple routine for determining which constraints to add. For the sake of exposition we will focus on polyhedral uncertainty sets, although a similar routine could be constructed for more general sets. In particular, for each pair of leaf parameters $\hat{\boldsymbol{\xi}}_i$ and $\hat{\boldsymbol{\xi}}_j$ and their corresponding induced partitions, expressed as

$$\Xi\left(\hat{\boldsymbol{\xi}}_i\right) = \{\boldsymbol{\xi} \in \Xi \,|\mathbf{F}\boldsymbol{\xi} \leq \mathbf{f}\} \quad \text{and} \quad \Xi\left(\hat{\boldsymbol{\xi}}_j\right) = \{\boldsymbol{\xi} \in \Xi \,|\mathbf{G}\boldsymbol{\xi} \leq \mathbf{g}\},$$

we can formulate the optimization problem

$$\begin{aligned} z(t) = \max_{\boldsymbol{\psi},\boldsymbol{\phi},\alpha\geq 0,\beta\geq 0} \quad & \alpha + \beta \\ \text{subject to } & F\boldsymbol{\psi} + \alpha\mathbf{e} \leq \mathbf{f} \\ & G\boldsymbol{\phi} + \beta\mathbf{e} \leq \mathbf{g} \\ & \boldsymbol{\psi}^s = \boldsymbol{\phi}^s \quad \forall s \in \{1, \dots t-1\}, \end{aligned} \tag{3.26}$$

the solution to which informs us if nonanticipativity constraints are necessary at time stage t. If the problem is infeasible then they are not required, as the first two of constraints can always be satisfied but the third may not necessarily be satisfiable. If there is a feasible solution but the objective value is 0 then all feasible solutions are on the boundaries of the uncertainty sets, which does not require the constraints. If the objective value is positive then we must enforce the nonanticipitavity constraints as there is an interior point for the partitions that is identical up to time t.

Finally, we present a simple alternative rule by which the need for nonanticipativity constraints can be determined. This rule is not a general method for enforcing nonanticipativity in finite adaptability as it relies on the particular structure produced by our partitioning scheme. Given two partitions $\Xi\left(\hat{\boldsymbol{\xi}}_i\right)$ and $\Xi\left(\hat{\boldsymbol{\xi}}_j\right)$, the rule for whether we need to enforce the nonanticipativity constraints $\mathbf{x}_i^t = \mathbf{x}_j^t$ is as follows:

1. Let $\hat{\boldsymbol{\xi}}_A \leftarrow \hat{\boldsymbol{\xi}}_i$ and $\hat{\boldsymbol{\xi}}_B \leftarrow \hat{\boldsymbol{\xi}}_j$.

2. If $\hat{\boldsymbol{\xi}}_A$ and $\hat{\boldsymbol{\xi}}_B$ **are** siblings, then let $t^{A,B} = \arg\min_s \left\{\hat{\boldsymbol{\xi}}_A^s \neq \hat{\boldsymbol{\xi}}_B^s\right\}$. If $t > t^{A,B}$, then we **do not** need to enforce the nonanticipativity constraints. If $t \leq t^{A,B}$, then we **do**.

3. Otherwise, if $\hat{\boldsymbol{\xi}}_A$ and $\hat{\boldsymbol{\xi}}_B$ **are not** siblings, let $\hat{\boldsymbol{\xi}}_A \leftarrow Parent\left(\hat{\boldsymbol{\xi}}_A\right)$ and $\hat{\boldsymbol{\xi}}_B \leftarrow Parent\left(\hat{\boldsymbol{\xi}}_B\right)$ and go to Step 2.

This rule is simple to implement, but can be overconservative as it may declare nonanticipativity constraints as required when they are not needed. Despite this shortcoming, it could be used to quickly to determine the cases where the constraints are not required before the above optimization approach can definitively determine whether they are truly required. We provide proof of the correctness of the rule, as well as an example demonstrating overconservatism, in Appendix A.2.

3.4.2 Multistage Method and Implementation

In the interests of space we will not present the multistage method in full, as it suffices to simply take the definition of the method for the two-stage case, but use the multistage partitioning scheme and add nonanticipativity constraints wherever they are required. We can also extend our lower bound approach to the multistage case, in particular

$$
\begin{aligned}
z_{lower}\left(\mathcal{T}\right) = \min_{\mathbf{x}\in\mathcal{X},z} \quad & z & & (3.27)\\
\text{subject to} \quad & \sum_{t=1}^{T} \mathbf{c}^t\left(\hat{\boldsymbol{\xi}}_j\right)\cdot\mathbf{x}_j^t \leq z & \forall\hat{\boldsymbol{\xi}}_j \in \mathcal{T}\\
& \sum_{t=1}^{T} \mathbf{a}_i^t\left(\hat{\boldsymbol{\xi}}_j\right)\cdot\mathbf{x}_j^t \leq b_i\left(\hat{\boldsymbol{\xi}}_j\right) & \forall\hat{\boldsymbol{\xi}}_j \in \mathcal{T}, i \in I\\
& \mathbf{x}_j^t = \mathbf{x}_k^t & \forall\hat{\boldsymbol{\xi}}_j, \hat{\boldsymbol{\xi}}_k \in \mathcal{T}, \forall t : \hat{\boldsymbol{\xi}}_j^{1,\ldots,t-1} = \hat{\boldsymbol{\xi}}_k^{1,\ldots,t-1},
\end{aligned}
$$

where the final set of constraints enforces the nonanticipativity restrictions. If two samples $\hat{\boldsymbol{\xi}}_j$ and $\hat{\boldsymbol{\xi}}_k$ are identical up to and including time $t-1$, then the corresponding

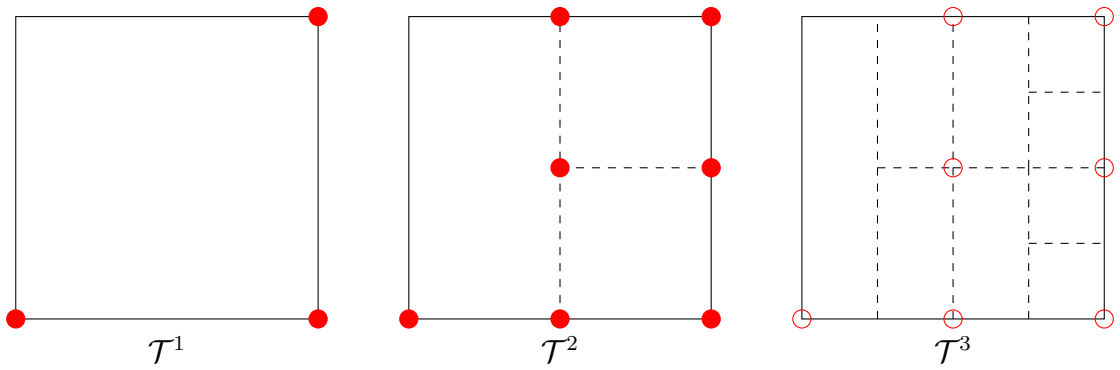

Figure 3-4: Partitioning of the uncertainty set in Example 2 using the multistage partitioning scheme. Note that at the second iteration we have two distinct second-stage decisions, unlike in Figure 3-3 where we had only one. When we partition again we grow to four distinct second-stage decisions, and nine distinct third-stage decisions.

decisions must be restricted to be equal at time t as we would not have any extra information to inform our decision. For the edge case $t = 1$ we enforce that all decisions are the same, as no uncertainty has been revealed yet.

We provide a worked example of applying the multistage algorithm to the lot sizing problem in Appendix A.3. This example demonstrates the substantial difference between our method and the method of Postek and Den Hertog [2014], which must employ various heuristics to avoid becoming overconservative.

3.5 Computational Experiments

The goal of this section is to demonstrate that, for problems of interest from the literature, the methods described in this chapter are practical and valuable. To do so we consider a two-stage binary capital budgeting problem (Section 3.5.1) and a mulit-stage lot sizing problem (Section 3.5.2). All methods were implemented in the Julia programming language using the JuMP [Lubin and Dunning, 2015, Dunning et al., 2015] and JuMPeR (Chapter 6) packages, with Gurobi 6.0.4 used as the solver [Gurobi Optimization Inc., 2016].

3.5.1 Capital Budgeting

Our first example is the two-stage capital budgeting problem described by Hanasusanto et al. [2015]. We must decide which of N projects to pursue to maximize profits, but are constrained by a known budget B. The cost c_i and profit r_i for each project i are affine functions of uncertain parameters $\boldsymbol{\xi}$,

$$c_i(\boldsymbol{\xi}) = \left(1 + \frac{1}{2}\boldsymbol{\Phi}_i \cdot \boldsymbol{\xi}\right) c_i^0 \qquad \text{and} \qquad r_i(\boldsymbol{\xi}) = \left(1 + \frac{1}{2}\boldsymbol{\Psi}_i \cdot \boldsymbol{\xi}\right) r_i^0.$$

We have the option of pursuing a project either before or after these uncertain parameters are observed. If we pursue a project after they are observed then we generate only a fraction $\theta \in [0, 1)$ of the profit - a penalty for delaying the decision. This can be expressed as the adaptive binary optimization problem

$$\begin{aligned}
\max_{z,\mathbf{x}} \quad & z & \\
\text{subject to} \quad & \mathbf{r}(\boldsymbol{\xi}) \cdot \left(\mathbf{x}^1 + \theta \mathbf{x}^2(\boldsymbol{\xi})\right) \geq z & \forall \boldsymbol{\xi} \in \Xi \\
& \mathbf{c}(\boldsymbol{\xi}) \cdot \left(\mathbf{x}^1 + \mathbf{x}^2(\boldsymbol{\xi})\right) \leq B & \forall \boldsymbol{\xi} \in \Xi \\
& \mathbf{x}^1 + \mathbf{x}^2(\boldsymbol{\xi}) \leq \mathbf{1} & \forall \boldsymbol{\xi} \in \Xi \\
& \mathbf{x}^1, \mathbf{x}^2(\boldsymbol{\xi}) \in \{0,1\}^N & \forall \boldsymbol{\xi} \in \Xi,
\end{aligned}$$

where $\mathbf{x}^1$ is the first-stage decision to pursue now, and $\mathbf{x}^2$ is the second-stage decision to pursue later. The uncertainty set is the hypercube $\Xi = \left\{\boldsymbol{\xi} \,\middle|\, \boldsymbol{\xi} \in [-1, 1]^4\right\}$. We used the same distributions for the parameters as Hanasusanto et al. [2015], and set Gurobi's integrality gap tolerance to 1% for all instances. All robust optimization problems were solved by reformulation to a deterministic problem.

We analyze our results from both the perspective of the decrease in gap versus total time, but also by "improvement" versus total time, where we use the same definition of improvement as Hanasusanto et al. [2015]:

$$\frac{z_{alg}\left(\mathcal{T}^k\right) - z_{alg}\left(\mathcal{T}^1\right)}{z_{alg}\left(\mathcal{T}^1\right)}.$$

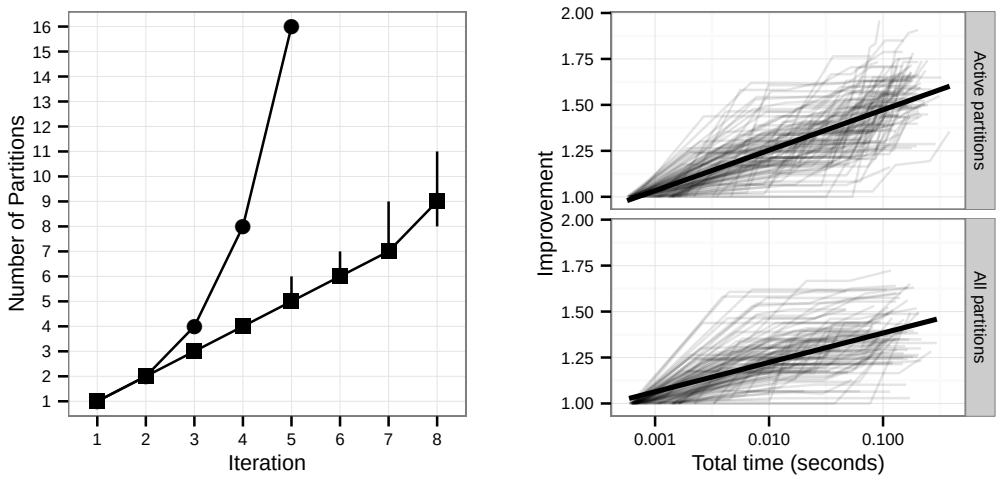

Figure 3-5: Left: median number of partitions versus iteration number if all partitions are subpartitioned (circles) or only active partitions are (squares). The top of the vertical line is the 90^{th} percentile, and the bottom is the 10^{th} percentile; there is no variation when all partitions are subpartitioned. Right: improvement versus time for the two approaches. Each semi-transparent line represents the improvement across time for a single instance, and the solid black line is the line of best fit. Partitioning only on active partitions delivers better quality solutions on average than partitioning all partitions for the same computation time.

Before comparing results with Hanasusanto et al. [2015] for larger problem sizes we first tried to understand the effects of only subpartitionining active partitions, which we define for this problem as the partitions that have an objective value that is within 10^{-2} of the worst-case objective value. As can be seen in Figure 3-5, for 100 instances of size $N = 5$ we see that while subpartitioning all partitions leads to the number of partitions growing exponentially, focusing on only the active partitions leads to a near-linear increase in partition count. As discussed in Section 3.3.2, more important than partition count is the solution quality versus time. The right side of Figure 3-5 demonstrates this clearly, as four and eight iterations of each approach both take approximately 0.1 seconds but the active partition-only approach delivers superior solutions on average.

For the purpose of comparison we considered 50 random instances of each size $N \in \{5, 10, 20, 30\}$, and in Figure 3-6 we plot improvement and gap versus time for each instance. We found that there is little variability across instances in performance

for $N \geq 10$, and that as N increases a pattern emerges where substantial initial improvement is made, before a multiple-iteration stall, followed by further improvement. It seems reasonable to suggest that this pattern is driven by the ratio of N to the fixed dimension of the uncertainty set, which is why the behaviour for $N = 5$ differs substantially from the rest.

For instances of size $N = 30$ Hanasusanto et al. [2015] found an improvement of approximately 65% with their two-partition solution, 90% with their three-partition solution, and 110% with their four-partition solution. However, even with only $N = 20$ almost all instances failed to solve to optimality using their method within 2 hours. In contrast we are able to consistently achieve an 80% improvement for instances of size $N = 30$ in under 1 second (which corresponds to a 30% bound gap). We suspect that this drastic difference in performance arises from the structure of the integer optimization problems being solved by the two methods. Our approach has a very simple block structure (one block per partition) not too dissimilar from the deterministic equivalent, whereas Hanasusanto et al. [2015] must introduce "big-M" constraints to allow for the partition structure to be included in the optimization, creating a computationally difficult problem. The two approaches are complementary, however: the method of Hanasusanto et al. [2015] would be preferable if having the best possible or a "simple" solution was important, such as when the second-stage decisions must be implemented as-is and we cannot reoptimize. Additionally, while it was not the case in this problem, if there is only uncertainty in the objective function coefficients then there are good theoretical reasons to suggest that the method of Hanasusanto et al. [2015] will perform better.

3.5.2 Multistage Lot Sizing

The multistage lot sizing problem described in Bertsimas and Georghiou [2015] is a challenging AMIO problem that features a mix of continuous and binary decisions across multiple time stages. We have used smaller versions of this problem throughout this chapter as a motivating example. The full formulation is provided in Bertsimas and Georghiou [2015], but we repeat the description here for clarity. Our goal is to

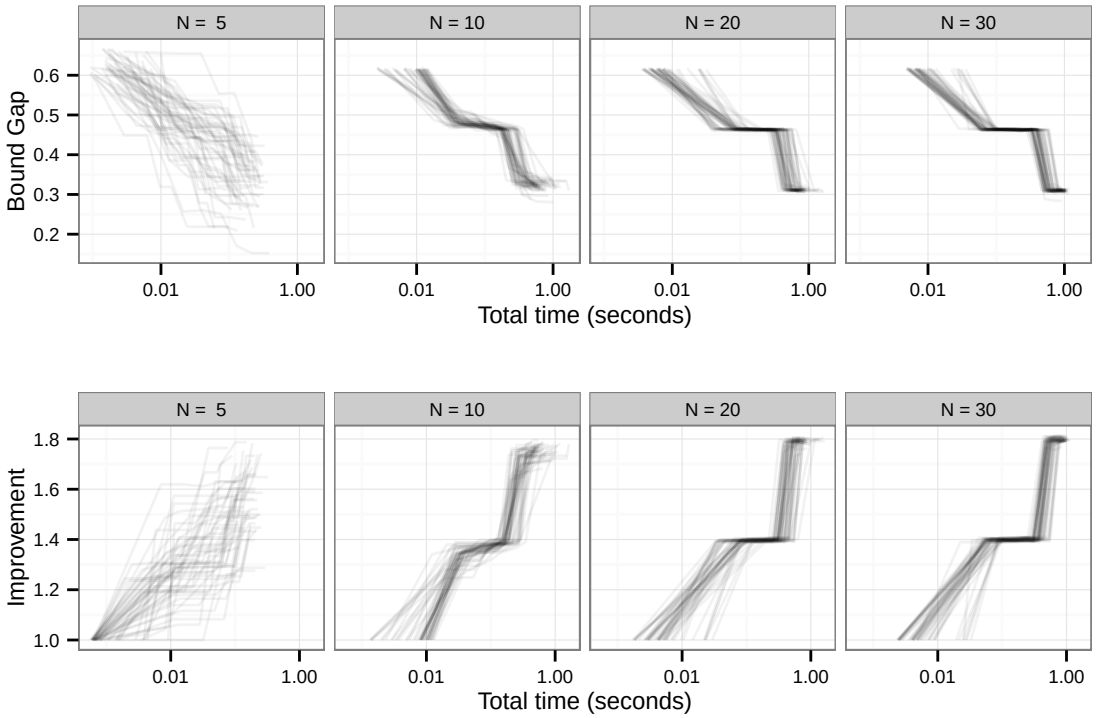

Figure 3-6: Improvement and bound gap versus time for 50 instances of size $N \in \{5, 10, 20, 30\}$. Each instance is represented by a semi-transparent line, and darker areas correspond to many overlapping lines.

minimize the cost of stock ordered plus holding costs, subject to the need to satisfy uncertain demand. We must decide x^t, the (continuous, non-negative) amount of stock ordered at time stage t for delivery at time stage $t+1$ at a unit cost of c_x, and y_n^t, a binary decision indicating whether to order a fixed amount q_n of stock for immediate delivery at time t at a unit cost of c_n. There are N different lot sizes to select from. We do not allow stock levels to become negative, and we impose a unit holding cost c_h for any stock remaining at the end of stage t. The uncertainty set is a box uncertainty set $\Xi = \{\boldsymbol{\xi} \,|\xi^1 = 1,\ l^t \leq \xi^t \leq u^t \quad \forall t \in \{2, \ldots, T\}\}$. For the continuous decisions $\mathbf{x}$ we start with an affine policy instead of a static policy, leading to a piecewise affine policy after partitioning (as described in Section 3.2.5). In order to aid understanding of the implementation of our method for this problem we provide a worked example in Appendix A.3.

The problem size may grow rapidly, proportional to T^k where k is the iteration count, if our method is applied naively. We address this by only subpartitioning the active partitions. Additionally many of the constraints only contain some of the uncertain parameters and as a result the set of active uncertain parameters is a set of infinite cardinality. To select a single active uncertain parameter per constraint we randomly weight the components and solve a small optimization problem to find a random extreme point of the set. We again use reformulation to convert the robust optimization problems to deterministic MIO problems. Gurobi's integrality gap tolerance was set to 1% for all instances, and the time limit was set to $20T$ seconds for instances with T time stages.

We implemented both our method and the method of Postek and Den Hertog [2014], and evaluated them over 50 randomly generated instances for $T \in \{4, 6, 8, 10\}$ for $N = 3$ (with results for $N = 2$ being qualitatively similar). The method of Postek and Den Hertog [2014] has multiple heuristics and parameters that can be selected: we used the same as were used in that paper's computational experiments, with the exception of the lower bound calculation where we used all found active uncertain parameters (which is what our method does). To make the running times of the two methods roughly equivalent we used 5 iterations of the method of Postek and

Method		T = 4	6	8	10
Our method (2 iter.)	Gap (%)	13.0	10.3	11.6	14.9
	Time (s)	0.0	0.5	7.7	108.6
Our method (3 iter.)	Gap (%)	11.4	9.3	11.3	14.9
	Time (s)	0.2	2.0	52.4	309.3
Postek and Den Hertog [2014]	Gap (%)	11.5	14.1	15.7	15.7
	Time (s)	0.4	1.6	10.8	77.8
Bertsimas and Georghiou [2015]	Gap (%)	17.2	34.5	37.6	-
	Time (s)	3381	9181	28743	-

Table 3.1: Comparison of our method (with two or three iterations) versus the methods of Postek and Den Hertog [2014] and Bertsimas and Georghiou [2015] for the multistage lot sizing problem. "Gap" is calculated as described in Bertsimas and Georghiou [2015], not as in this chapter. Time for our method and Postek and Den Hertog [2014] is total time to solve all upper and lower bound problems up to and including that iteration; both gap and time are medians for these two methods. Times for Bertsimas and Georghiou [2015] are obtained from a different computer but with roughly comparable performance, on instances from the same family.

Den Hertog [2014], and 3 iterations of our method. We also sought to compare our results with Bertsimas and Georghiou [2015], who use a different definition of a bound gap than us:

$$\frac{z_{alg}\left(\mathcal{T}^k\right) - z_{lower}\left(\mathcal{T}^k\right)}{2\left(z_{alg}\left(\mathcal{T}^k\right) + z_{lower}\left(\mathcal{T}^k\right)\right)}. \tag{3.28}$$

We have summarized the results in two ways. Firstly in Table 3.1 we compare our method with the methods of Postek and Den Hertog [2014] and Bertsimas and Georghiou [2015]. Our method finds superior solutions in only a fraction of the time of Bertsimas and Georghiou [2015], suggesting that our method of approximating the optimal piecewise affine policy is more efficient than trying to directly optimize the "pieces". Comparisons with Postek and Den Hertog [2014] are hard to make in this table format, as to be fair we must compare total computation time versus solution quality, not iteration count version solution quality. We visualize the full comparison, with quality measured by bound gap and improvement, in Figure 3-7. Here we can see that the curves for our method in general lie close to or below the curves for Postek and Den Hertog [2014] across all instance sizes, suggesting that in this setting our method will produce better solutions on average for the same amount of computational time invested.

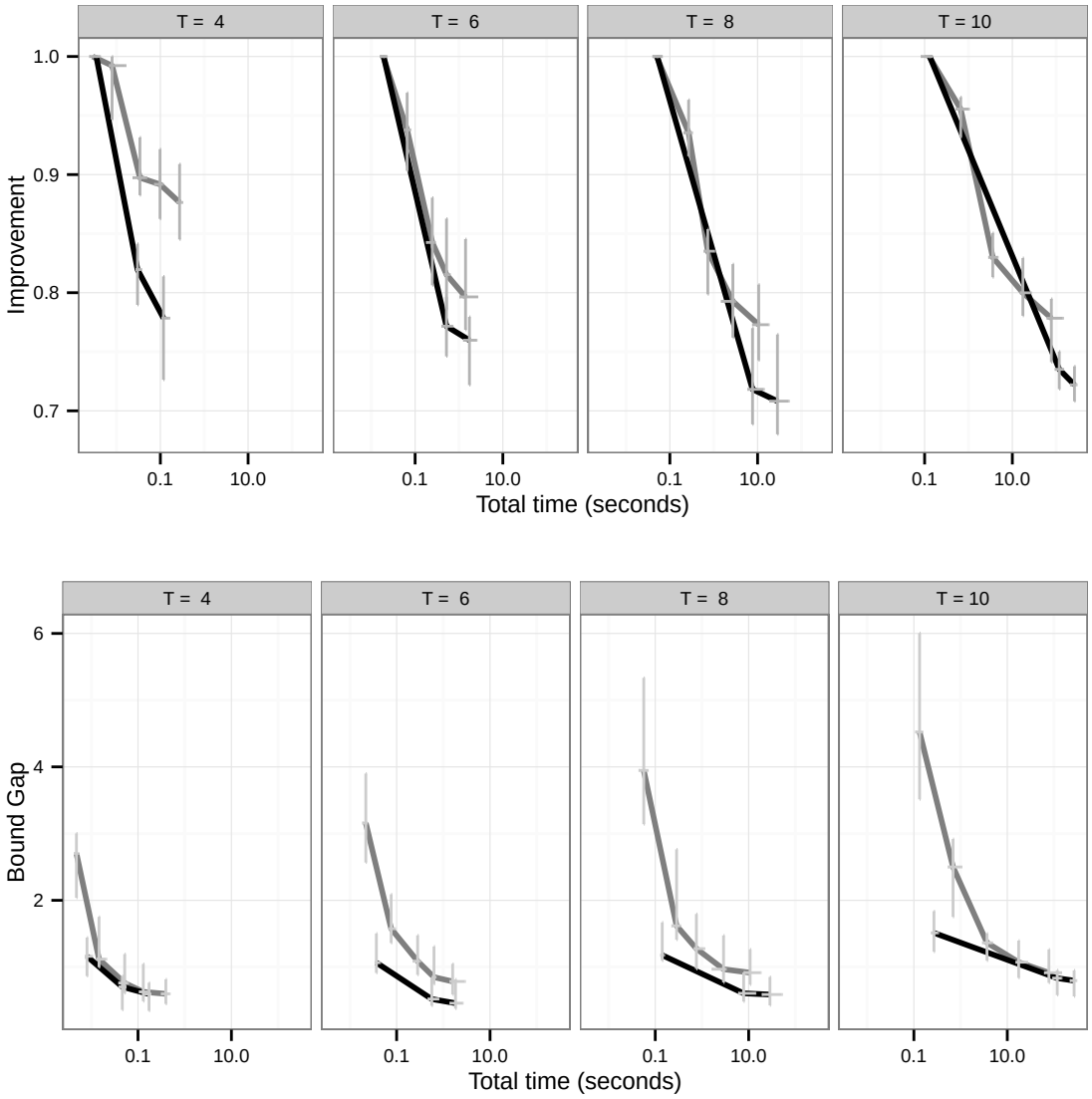

Figure 3-7: Time versus improvement and bound gap for the multistage lot sizing problem. The black line and dark grey line correspond to our method and the method of Postek and Den Hertog [2014] respectively. All quantities are medians over 50 instances at each iteration, with the light grey error bars showing the 40^{th} and 60^{th} percentiles. The black curve is generally below the dark grey curve, indicating that our method obtains better results for the same amount of computational effort.

3.6 Concluding Remarks

In this chapter, we presented a novel partition-and-bound method for solving adaptive mixed integer optimization problems. We use finite partitioning to approximate the fully adaptable solution, and then use information from the solution of the finitely partitioned problem to further improve the partitioning and calculate lower bounds. The particular Voronoi diagram-like partitioning scheme we employ is inspired by the observation that, under some assumptions, a finite partitioning of the uncertainty set can only improve the solution quality if the partitioning separates regions of the uncertainty set that correspond to binding constraints. We find that our method delivers high-quality solutions for the time spent, even as the problem scales in both number of time stages and in the number of decision variables. The time per iteration does increase with the number of time stages, although we observed that the number of partitions created typically grows much slower than the worst-case by focusing on active partitions. Our method should be viewed as one option among many for AMIO. For example if "good" solutions are required quickly, our method excels. If a near-fully adaptive solution is truly needed, other approaches may be more appropriate. We do note that, in a situation where we can reoptimize our decision at each time stage, perfectly identifying the fully adaptive solution may be unnecessary as the only important decision is the decision that must be made here-and-now.

Finally, we can draw analogies with the branch-and-bound technique for integer optimization. In branch-and-bound we use the linear relaxation of an integer optimization problem to provide a lower bound on the true integer solution, use the information obtained from the solution to branch on fractional variables to obtain upper bounds, and repeat until we are satisfied with the bound gap. In integer optimization the development of new cutting planes and presolve techniques has led to substantial improvements in the time to tighten these bounds gaps, and we believe there is exciting future work in developing new techniques for AMIO as extensions of this partition-and-bound approach.

Chapter 4

The Price of Flexibility

4.1 Introduction

Process flexibility is an important component of the strategies used by many manufacturing firms to manage uncertain product demand. Jordan and Graves [1995] define it as a firm's ability to "build different types of products in the same manufacturing plant or on the same production line at the same time." Flexibility enables, for example, a firm to utilize surplus capacity that may exist at some plants to produce products that have higher-than-expected demand that would otherwise exceed the capacity of any single plant. Many authors have related tales of both the successes of flexibility, and the failures of inflexibility. For example, Biller et al. [2006] report that a failure by Chrysler to keep up with demand, despite having underutilized capacity at some plants, lead to an estimated loss of $240M in pretax profit. Mak and Shen [2009] relay media accounts that the Ford Motor Company made a $485M investment in 2002 to increase flexibility at two of their plants and to add flexibility at most of their engine and transmission plants worldwide; Chou et al. [2010] report that similar initiatives have been undertaken by both GM and Nissan.

The more products each plant is capable of producing, the more flexible the production system becomes. With a "full" flexibility *design* each plant can produce all products, while under a "dedicated" flexibility design each plant produces a single product (as visualized in Figure 4-1). While a full flexibility design places a firm in

the best position to respond to uncertain demand, it is normally unrealistic from both a cost and management perspective for many firms to implement such a design. The natural goal then is to reduce the *price of flexibility* while realizing as many of the benefits of flexibility as possible, given considerations of profitability and feasibility.

Contributions

The primary contribution of this chapter is a group of methods based on *adaptive robust optimization* that reduce the price of flexibility by optimizing for *profitable* flexibility designs in unbalanced, nonhomogenous manufacturing systems, where the number of plants and products may differ, the plants may have varying capacities, the products may have varying marginal profit by plant, and the costs for flexibility may vary between plants. We demonstrate through simulations that our method can reduce the price by 30% or more versus standard flexibility designs described in the literature, with negligible effects on revenues. A secondary contribution are theoretical results related to the application of *Pareto efficiency* to adaptive robust optimization. In particular, we demonstrate that Pareto efficient solutions can be found with little additional computational effort, and show through computation that considering Pareto efficiency improves the designs produced by our methods.

Previous work

Designs with a low price of flexibility will typically be those in which each plant produces a small number of products and thereby gains most of the benefit of full flexibility at only a fraction of the cost. Jordan and Graves [1995] represents one of the first comprehensive studies into the effectiveness of partial flexibility designs. In particular, they seek to explore the effect of designs on expected sales and capacity utilization, and offer a convincing demonstration that "chaining" is a key property of effective designs. If process flexibility is viewed as a network of links between plants and products, then a chain design is one in which a "cycle" can be traced between any product and any plant by following these links. One such design would be a "long chain" design in which, for example, Plant A produces both Product 1 and

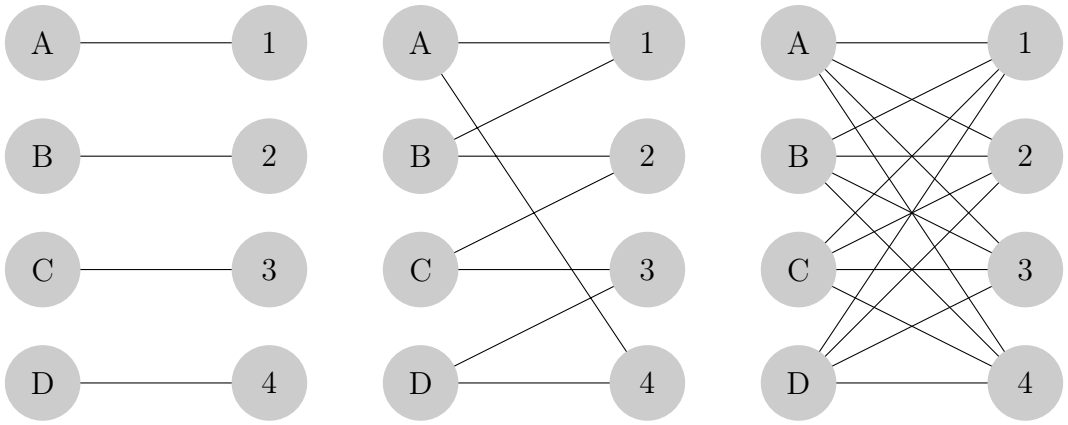

Figure 4-1: Examples of process flexibility designs, between plants (A,B,C,D) and products (1,2,3,4). Left: "dedicated" – each plant produces one product, and each product is produced by one plant. Center: "long chain" – each plant produces two products, and each product is produced by two plants. Right: "full" – each plant produces all products.

Product 4, Plant B produces Product 1 and Product 2, and so on (as shown in the center of Figure 4-1). Jordan and Graves [1995] quantify both analytically and by simulation that such designs are capable of approximating the performance of a full flexibility design with substantially fewer links. However, they do not address the problem of determining a design for any specific firm or setting, where we would expect that the costs of the adding flexibility at each plant may vary widely, or may not even be possible for some pairs of plant and product.

Since the work of Jordan and Graves [1995] many researchers have attempted to analytically explain the observed effectiveness of long chain and other partial flexibility designs, mainly in *balanced* homogeneous systems (i.e., systems where the number of plants and products are the same, all plants have equal capacities and all products face identical demand distributions). For example, in terms of average performance, Chou et al. [2010] provide theoretical performance guarantees for long chain designs under some distributions, including two-point, uniform and normal distributions. Simchi-Levi and Wei [2012] proved that, in a balanced and symmetrical system with exchangeable random demand, the long chain design maximizes the expected maximum sales among all "2-regular" flexibility designs. More recently, Wang and Zhang [2015] established a lower bound on the ratio between the expected sales

for a long chain design and a full flexibility design when all products share an identical demand distribution. Worst-case performance of the long chain and other sparse designs for balanced and homogeneous systems has also been discussed by Wei and Simchi-Levi [2015].

Many authors have proposed guidelines to identify effective partial flexibility designs, including Chou et al. [2010], Deng and Shen [2013], Chou et al. [2011], and Chen et al. [2014]. These consist of heuristics, and assume homogeneity in relevant parameters such as identical demand distributions. To the best of our knowledge, the only paper addressing the problem of finding optimal flexibility designs in unbalanced, nonhomogeneous manufacturing systems is by Mak and Shen [2009]. The authors propose a stochastic programming approach, with demand modeled as belonging to a known distribution from which a finite set of scenarios can be drawn. The resulting model is computationally challenging to solve, so in order to find feasible solutions they propose a relaxation that sacrifices solution quality and requires a complex implementation. They demonstrate that, despite the relaxation and in some settings, they are able to produce designs (a) identical to long chain designs, and (b) considerably better than long chain designs in settings where a "generic" long chain design has a high price, primarily due to the use of high and nonhomogeneous costs.

Structure

We have structured the chapter as follows:

- In Section 4.2, we model the process flexibility design problem as a robust adaptive optimization problem, where the demand for each product is treated as an uncertain parameter of the model, and the objective is to maximize profits and thereby reduce the price of flexibility. We present both a *max-min* robust objective variant and a *relative* robust objective variant, and detail how our model can be reformulated as a computationally tractable deterministic optimization problem solvable with off-the-shelf optimization software. We provide guidance on various ways the uncertain demand can be modeled within our optimization model depending on the risk preferences of management and available data.

Finally we demonstrate the extensibility and generality of our model, including how it can incorporate uncertainty in other parameters (e.g., disruptions in capacity) and restrictions on capabilities (e.g., restricting plants to producing no more than n products).

- In Section 5.4, we introduce the notion of *Pareto efficiency* to adaptive robust optimization, and apply it to manufacturing process flexibility. We develop a theoretical result that demonstrates how we can obtain flexibility designs that perform well in both "worst-case" and "average-case" demand scenarios for little extra computational effort. We present a new notion of *relaxed-Pareto efficiency* that allows us to further explore the trade-offs between worst-case and average-case performance according to the preferences of a decision maker.

- In Section 4.4, we first provide computational evidence that we can solve instances of practical size quickly on commodity hardware (e.g., instances with 15 plants and 15 products in one hour). We next provide insights derived from comparing the relative merits of different flexibility designs as viewed through our optimization model, which qualitatively reproduce some of the insights provided by Jordan and Graves [1995] and others. We then quantify to what degree Pareto efficient designs improve over initial designs, and then compare these Pareto efficient designs with designs obtained by a stochastic programming model under a variety of parameter choices. We finally compare the designs produced by our method with alternatives such as long chains and find substantial cost savings with little impact on revenue, and thus higher profits.

4.2 Process Flexibility as Adaptive Robust Optimization

In this section, we present an optimization model that can be used to reduce the price of flexibility by optimizing for a profitable design that is both robust with respect to demand uncertainty and can incorporate nonhomogenous capacities, costs, and

revenues. We consider systems with n plants, each with a capacity $c_i \geq 0$, and with m products. We model the demand for the products as an uncertain vector $\mathbf{d} = (d_1, \ldots, d_m)$. As the demand is not known with certainty, we treat the problem of flexibility design as a two-stage process: the design itself must be decided *here-and-now*, but the production decision for how much of the demand for each product to satisfy from each plant is a *wait-and-see* decision that is deferred until after the demand is realized.

Multiple techniques have been developed in the optimization literature for modeling uncertainty in optimization problems. Here we have elected to take a *robust optimization* (RO) perspective, where we model the uncertain demand vector as being drawn from a known *uncertainty set* $\mathcal{U}$. Instead of assigning a probability distribution over the demand scenarios, as is often done in the stochastic programming literature, we will instead optimize with respect to the *worst-case realization* of the demand over the uncertainty set (for a review of RO, please refer to the survey by Bertsimas et al. [2011a]). Our model thus represents an *adaptive robust optimization* (ARO) approach, with a $\max - \min - \max$ objective: we initially select a flexibility design (max), "nature" adversarially selects a demand scenario from the uncertainty set (min), and finally we select a production plan to maximize profits (max) and reduce the price of flexibility.

4.2.1 Adaptive Robust Optimization Model

Perhaps the most natural objective is to maximize total profit. Let $i \in \{1, \ldots, n\}$ index over the plants, and $j \in \{1, \ldots, m\}$ index over the products. We can enable the flexibility for plant i to produce product j in exchange for a one-time investment cost of B_{ij}. We define T_{ij} to be the production cost for manufacturing one unit of product j from plant i, and define p_j to be the revenue for selling one unit of product

j. The ARO formulation for this objective is

$$\max_{\mathbf{x},\mathbf{y}(\mathbf{d})} \min_{\mathbf{d}\in\mathcal{U}} \sum_{i,j} (p_j - T_{ij})\, y_{ij}(\mathbf{d}) - \sum_{i,j} B_{ij} x_{ij} \tag{1}$$

$$\text{subject to} \quad \sum_j y_{ij}(\mathbf{d}) \le c_i \qquad \forall i,\ \forall \mathbf{d} \in \mathcal{U} \tag{4.1a}$$

$$\sum_i y_{ij}(\mathbf{d}) \le d_j \qquad \forall j,\ \forall \mathbf{d} \in \mathcal{U} \tag{4.1b}$$

$$0 \le y_{ij}(\mathbf{d}) \le d_j x_{ij} \qquad \forall i, j,\ \forall \mathbf{d} \in \mathcal{U} \tag{4.1c}$$

$$\mathbf{x} \in \{0,1\}^{n\times m},$$

where x_{ij} are binary decisions for the flexibility design (whether to enable plant i to produce product j) that must be made here-and-now, and $y_{ij}(\mathbf{d})$ are the continuous wait-and-see production decisions (how much of the demand for product j to satisfy from plant i) that can be made after $\mathbf{d}$ is known. Note that we moved the inner maximization problem over the production decisions to outside the minimization over the uncertainty set: in this perspective, we are optimizing here-and-now over a space of possible functions $\mathbf{y}(\mathbf{d})$. These two representations are equivalent, but the representation we have selected here simplifies the discussion of computational details in Section 4.4. The objective (1) is the marginal profit from selling the products less the costs of the flexibility design. Constraint (4.1a) restricts the total amount of production at each plant i to be no more than the the plant's capacity. Constraint (4.1b) restricts the total amount produced and sold for each product j to be no more than the demand for that product. Finally constraint (4.1c) forces the production of product j from plant i to be zero if plant i is unable to produce product j (that is, if $x_{ij} = 0$).

4.2.2 Relative Profit ARO Model

While the absolute profit objective function of (4.1) is easy to understand, it may not be the best choice when coupled with a RO model of uncertainty for reducing the price of flexibility. Flexible designs excel in a setting where unused capacity at

some plants can be used to satisfy higher-than-expected product demand. However, the objective function of (4.1) does not value such designs particularly highly. As evidence of this, consider fixing the flexibility design and determining the worst-case demand scenario, which leaves only the marginal profit component of the objective function. We can observe that the worst-case scenario will be one in which demands are low, as this minimizes total profit and penalizes designs that add flexibility:

Example 1. Pessimism of model (4.1): Consider a problem with $n = 5$ plants and $m = 5$ products. We will assume that all plants are identical, as are all products. The profit p_j for all products is 1, and already includes production costs (i.e., $T_{ij} = 0$). We assume that the dedicated flexibility design is already in place ($B_{ii} = 0$), and we are interested in whether it is worth adding extra flexibility at a cost $B_{ij} = 5$. The uncertainty set restricts each d_j to the range $[50, 150]$ with a restriction that no more than two of the products can deviate from their "mean" demand of 100. We set the capacity of each plant c_i to be equal to 110, slightly more than the mean demand.

If we add no additional flexibility, our objective function simplifies to

$$\min_{\mathbf{d}\in\mathcal{U}} \sum_{i,j=1}^{5} y_{ij}(\mathbf{d}) = \sum_{k=1}^{5} \min\{c_k, d_k\}. \tag{4.2}$$

The worst-case scenario is one in which demand is as low as possible, giving a profit of 400. Consider now a "long chain" design, such that plant 1 produces products 1 and 2, plant 2 produces product 2 and 3, and so on. This requires adding flexibility to each of the 5 plants at a total cost of 25. The worst-case demand is again the case where demand is low, e.g. $\mathbf{d} = [50, 50, 100, 100, 100]$, so profit for the long chain design will be 375 (worse than a dedicated design). Indeed, for these parameters any flexibility design with more flexibility than the dedicated design will never have a better worst-case profit that the dedicated design as, in the low demand scenario, no benefit is realized from the flexibility but we still pay a price for the flexibility.

We thus propose a modification to the objective function (1) that will make the model less "pessimistic", and produce solutions that we would expect to perform well

in both worst-case scenarios and more "average" scenarios as well. In particular, we propose to optimize for a worst-case *relative* measure of profit, where we normalize the profits for our design by *total revenue*:

$$\max_{\mathbf{x},\mathbf{y}(\mathbf{d})} \min_{\mathbf{d}\in\mathcal{U}} \frac{\sum_{i,j} (p_j - T_{ij})\, y_{ij}(\mathbf{d}) - \sum_{i,j} B_{ij} x_{ij}}{\sum_j p_j d_j}. \tag{4.3}$$

The total revenue can be viewed as an optimistic estimate of what profits could be achieved, and so by normalizing by total revenue the worst-case scenario is no longer the low-demand scenario. This objective function is similar to a "competitive ratio" or "relative regret" objective function. Simchi-Levi and Wei [2015] consider a similar competitive ratio criterion for assessing flexibility designs in manufacturing, while in the field of network revenue management Ball and Queyranne [2009] consider competitive ratios and Perakis and Roels [2010] consider robust regret objective functions. While total revenue is not the only possible measure that profits can be normalized by, it has the benefit of being interpretable, in the same units as profit, and being linear in the demand (with positive implications for computational tractability).

Our ARO model for manufacturing flexibility with a relative profit objective function can thus be expressed as

$$\begin{aligned}
\max_{\mathbf{x},\mathbf{y}(\mathbf{d}),z} \quad & z & & \\
\text{subject to} \quad & \sum_{i,j} (p_j - T_{ij})\, y_{ij}(\mathbf{d}) - \sum_{i,j} B_{ij} x_{ij} \geq \sum_j p_j d_j z & \quad & \forall \mathbf{d} \in \mathcal{U} \\
& \sum_j y_{ij}(\mathbf{d}) \leq c_i & & \forall i,\ \forall \mathbf{d} \in \mathcal{U} \\
& \sum_i y_{ij}(\mathbf{d}) \leq d_j & & \forall j,\ \forall \mathbf{d} \in \mathcal{U} \\
& 0 \leq y_{ij}(\mathbf{d}) \leq d_j x_{ij} & & \forall i, j,\ \forall \mathbf{d} \in \mathcal{U} \\
& \mathbf{x} \in \{0,1\}^{n\times m}, & &
\end{aligned} \tag{4.4}$$

where we have converted the ratio objective implied by (4.3) into a linear uncertain constraint by using an epigraph formulation with auxiliary variable z. In our prelim-

inary experiments, this formulation usually produced better designs than (4.1), so we will focus on (4.4) throughout the chapter.

Example 2. Example 1 with a relative regret objective: We now revisit the previous problem, using model (4.4). Considering first the dedicated design (no additional flexibility), we note that the worst-case demand scenarios had the form $\mathbf{d} = (50, 50, 100, 100, 100)$ – however, under the new objective function we would have a ratio of 1. If we consider scenarios such as $\mathbf{d} = (150, 150, 100, 100, 100)$ instead, we obtain a ratio of $(2 \times 110 + 3 \times 100)/600 \approx 0.867$ (as the denominator is not limited by capacity), which is the worst-case scenario for this design. The long chain design in the low demand scenario obtains a ratio of $375/400 \approx 0.938$, worse than the dedicated design. However, the worst-case demand scenario for the ratio objective is again the high-demand scenario. By using the power of flexibility, we can satisfy $5 \times 110 = 550$ out of the total demand of 600, achieving a ratio of $(550 - 25)/600 = 0.875$, which is superior to the ratio for the dedicated design. We would thus prefer the long chain design for these parameters.

We will now discuss how to solve problem (4.4), before addressing the selection of the uncertainty set $\mathcal{U}$ (Section 4.2.4) and possible extensions (Section 4.2.5).

4.2.3 Solving the ARO Model

To this point we have deferred discussing in detail how to obtain a solution to problem (4.4). The key difficulty that must be addressed is that the optimal wait-and-see decision $\mathbf{y}(\mathbf{d})$ may have a complex relationship with demand $\mathbf{d}$, making the optimization problem challenging to solve. Ben-Tal et al. [2004] have shown that, even for the case where the design $\mathbf{x}$ is fixed and we thus have only continuous decisions, this problem is hard in both the computational complexity sense and in practice. A popular remedy, first applied to the problem of ARO by Ben-Tal et al. [2004], is to drastically simplify the problem by restricting the space of admissible adaptive continuous decisions to those that are affine with respect to the uncertain parameters $\mathbf{d}$. This simplification is referred to as *affine adaptability* or *linear decision rules* in

the literature, and has been applied successfully in application areas such as project management [Chen et al., 2007] and supply chains [Ben-Tal et al., 2005]. It also been shown from a theoretical point of view [Bertsimas et al., 2010, Bertsimas and Goyal, 2012, Bertsimas and Bidkhori, 2015] that, in many cases, affine adaptability is optimal or a good approximation of the fully adaptive ideal.

We now apply the concept affine adaptability to our model. Let

$$y_{ij}(\mathbf{d}) = \mathbf{Q}_{ij}^T \mathbf{d} + q_{ij}, \tag{4.5}$$

where $\mathbf{Q}_{ij} \in \mathbb{R}^m$ and $q_{ij} \in \mathbb{R}$ are the decision variables that we will now optimize over. We can substitute (4.5) into (4.4) to obtain the problem

$$\begin{aligned}
\max_{\mathbf{x},\mathbf{Q},\mathbf{q},z} \quad & z & & & (4.6)\\
\text{subject to} \quad & \sum_{i,j} (p_j - T_{ij}) \left(\mathbf{Q}_{ij}^T \mathbf{d} + q_{ij}\right) - \sum_{i,j} B_{ij} x_{ij} \geq \sum_j p_j d_j z & & \forall \mathbf{d} \in \mathcal{U} \\
& \sum_j \left(\mathbf{Q}_{ij}^T \mathbf{d} + q_{ij}\right) \leq c_i & & \forall i,\ \forall \mathbf{d} \in \mathcal{U} \\
& \sum_i \left(\mathbf{Q}_{ij}^T \mathbf{d} + q_{ij}\right) \leq d_j & & \forall j,\ \forall \mathbf{d} \in \mathcal{U} \\
& 0 \leq \mathbf{Q}_{ij}^T \mathbf{d} + q_{ij} \leq d_j x_{ij} & & \forall i, j,\ \forall \mathbf{d} \in \mathcal{U} \\
& \mathbf{x} \in \{0,1\}^{n \times m}, & &
\end{aligned}$$

which can be viewed as a relaxation of (4.4), and thus the objective value will be an upper bound on the fully adaptive problem's objective value. In return for this approximation we obtain a robust optimization problem with constraints that are linear with respect to the decision variables for fixed values of $\mathbf{d}$, and are linear with respect to $\mathbf{d}$ for fixed values of the decision variables. As a result, we are able to reformulate (4.6) to a deterministic mixed-integer optimization with convex constraints for many choices of $\mathcal{U}$. For example, if $\mathcal{U}$ is a polyhedron, then the resulting reformulation is a mixed-integer linear optimization problem, and if $\mathcal{U}$ is ellipsoidal, then the reformulation will be a mixed-integer second-order cone optimization problem. In

Section 4.4.1 we provide computational evidence that (4.6) can be easily solved on commodity hardware for instances of realistic size.

4.2.4 Selecting the Uncertainty Set

One of the key modeling decisions that we have deferred to this point is the selection of the uncertainty set of possible demand scenarios. There are many possible choices, but key considerations include

- whether the resulting reformulation of (4.6) is tractable,
- what data we have available about the uncertain demand, and
- how robust we seek to be against uncertainty in the demand.

Given the wide variety of choices available we will focus here on polyhedral *budget* uncertainty sets based on the uncertainty set described in Bertsimas and Sim [2004]. Other alternatives include sets such as the ellipsoidal set presented in Ben-Tal and Nemirovski [1999] and "data-driven" sets such as those described in Bertsimas et al. [2013a].

The first uncertainty set we will describe, the *budget* uncertainty set, is defined by the parameters $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$. In particular, for each product j we have a nominal (or average) demand parameter μ_j and a deviation parameter σ_j. Both these parameters can be either estimated from data (for example, a sample mean and standard deviation), or from some other source (such as estimates by a domain expert). The uncertainty set is then the polyhedron

$$\mathcal{U} = \{\mathbf{d} \mid d_j = \mu_j + \sigma_j z_j \ \forall j \in \{1, \ldots, m\},\ \|\mathbf{z}\|_1 \leq \Gamma,\ \|\mathbf{z}\|_\infty \leq 1\}, \tag{4.7}$$

where Γ is the *budget of uncertainty*. When $\Gamma = 0$ the uncertainty set reduces to the nominal demand scenario, i.e., $\mathcal{U} = \{\boldsymbol{\mu}\}$, and when $\Gamma = m$ the set reduces to the "hypercube" $\mathcal{U} = \{\mathbf{d} | \mu_j - \sigma_j \leq d_j \leq \mu_j + \sigma_j\}$. We can interpret Γ (when Γ is an integer) as controlling the number of products whose demand can vary from the

nominal case. While Bertsimas and Sim [2004] provide some theoretical guidance on how one can select Γ, in practice the best approach is to solve the problem for multiple values of Γ and select amongst the solutions based on some metric of interest, e.g., performance of the optimal designs in simulation. We sketch this process, which is related to the process of cross-validation for parameter tuning in machine learning, in our computational experiments in Section 4.4.4.

The budget uncertainty set (4.7) assumes there is no correlation between products. If we have some estimation available of the covariance between products, e.g., a covariance matrix $\mathbf{\Sigma}$, then we can create a *correlated budget* uncertainty set

$$\mathcal{U} = \left\{ \mathbf{d} \;\middle|\; d_j = \mu_j + \sum_k \sigma_{jk} z_k \;\forall j \in \{1, \ldots, m\},\; \|\mathbf{z}\|_1 \leq \Gamma,\; \|\mathbf{z}\|_\infty \leq 1 \right\}. \tag{4.8}$$

As a covariance matrix is positive-definite by construction, we can easily obtain $\mathbf{\Sigma}^{\frac{1}{2}}$ by performing a Cholesky factorization. The use of this uncertainty set does not make the optimization problem significantly more difficult to solve, so if sufficient data or expertise is available to estimate $\mathbf{\Sigma}$, then we would expect higher quality and less pessimistic solutions with the correlated budget uncertainty set. We explore the effects of correlation on the relative merits of different possible designs in Section 4.4.2.

4.2.5 Extensibility of Formulation

Part of the benefit our ARO approach to flexibility is that incorporating extensions is relatively simple. For example, process flexibility puts the firm in a better position to match available capacity with variable demand, as described in the introduction. However, many firms are also interested in *operational flexibility*, which in the context of this problem could refer to the possibility of allocating extra capacity at each plant after the demand is realized.

To incorporate operational flexibility in our model we will define $s_i(\mathbf{d})$ to be the amount of additional extra capacity we will allocate at plant i as a function of the uncertain demand $\mathbf{d}$. Let the cost of additional capacity be h_i per unit. We can now

extend (4.4) to include this new decision:

$$\max_{\mathbf{x},\mathbf{y}(\mathbf{d}),\mathbf{s}(\mathbf{d}),z} \quad z \tag{4.9}$$

$$\begin{aligned}
\text{subject to} \quad & \sum_{i,j} (p_j - T_{ij})\, y_{ij}(\mathbf{d}) - \sum_{i,j} B_{ij} x_{ij} - \sum_i h_i s_i(\mathbf{d}) \geq \sum_j p_j d_j z && \forall \mathbf{d} \in \mathcal{U} \\
& \sum_j y_{ij}(\mathbf{d}) \leq c_i + s_i(\mathbf{d}) && \forall i,\ \forall \mathbf{d} \in \mathcal{U} \\
& \sum_i y_{ij}(\mathbf{d}) \leq d_j && \forall j,\ \forall \mathbf{d} \in \mathcal{U} \\
& 0 \leq y_{ij}(\mathbf{d}) \leq d_j x_{ij} && \forall i, j,\ \forall \mathbf{d} \in \mathcal{U} \\
& \mathbf{x} \in \{0,1\}^{n \times m},
\end{aligned}$$

where $\mathbf{s}(\mathbf{d})$ appears in the objective function constraint and in the constraint that controls the maximum amount of demand satisfied by plant i. As with (4.4), we can replace $\mathbf{s}(\mathbf{d})$ with an affine adaptability approximation, for example

$$s_i(\mathbf{d}) = \mathbf{R}_i^T \mathbf{d} + r_i, \tag{4.10}$$

which we can then substitute into (4.9) as we did with $\mathbf{y}$.

A partial list of possible extensions that can be incorporated into this model includes:

- *capacity disruption*, where $\mathbf{c}$ is itself uncertain, and the production decisions are a function of both the realized demand and the realized available capacity, e.g., $\mathbf{y}(\mathbf{d}, \mathbf{c})$.

- *cost uncertainty*, where we allow for uncertainty in the cost of implementing a process flexibility design, e.g., $\mathbf{B}$ is an uncertain parameter. This naturally leads to a model in which we have the option of implementing process flexibility both before ($\mathbf{x}$) or after ($\mathbf{x}(\mathbf{B})$) the uncertain costs $\mathbf{B}$ are realized, where the wait-and-see design incurs a penalty to reflect increased costs for having to implement the flexibility faster to respond to the demand.

- *side constraints*, that allow for capturing more real-life restrictions. For example, linear constraints that could be added that enforce restrictions such as "plant 1 cannot produce both products 1 and 2", "the total number of products that any one plant can produce must not exceed 3", or "each product must be produced by two plants".

4.3 Pareto Efficiency in Manufacturing Operations

Our model is aimed at reducing the price of flexibility by finding flexibility designs that maximize relative profitability with respect to the worst-case demand scenarios contained within our uncertainty set. We do not explicitly consider performance under more average-case scenarios when optimizing, but ideally our design would also perform well across a range of realizations. This notion is closely related to the concept of *Pareto efficiency*: we wish to find designs that are not *dominated* by any another design. In this case, that means designs that are optimal in the worst-case, but there does not exist another design that is also optimal in the worst-case, but is better for some (or many) "average" scenarios. This concept was first applied in the context of robust optimization setting by Iancu and Trichakis [2013], who consider the general RO problem

$$z^* = \max_{\mathbf{x} \in X} \min_{\mathbf{d} \in \mathcal{U}} f(\mathbf{d}, \mathbf{x}), \tag{4.11}$$

where z^* is the optimal objective value, and X^* is the set of all optimal solutions (that is, all solutions whose worst-case objective value is equal to z^*). A solution $\mathbf{x} \in X^*$ is said to be a *Pareto robustly optimal solution* if there is no other $\tilde{\mathbf{x}} \in X^*$ such that $f(\mathbf{d}, \tilde{\mathbf{x}}) \geq f(\mathbf{d}, \mathbf{x})$ for all $\mathbf{x} \in \mathcal{U}$ and $f(\tilde{\mathbf{d}}, \tilde{\mathbf{x}}) > f(\tilde{\mathbf{d}}, \mathbf{x})$ for some $\tilde{\mathbf{d}} \in \mathcal{U}$. In other words, a Pareto robustly optimal solution is a robustly optimal solution, and there is not other solution which is better under all demand scenarios and strictly better for at least one demand scenario.

In this section, we extend the work of Iancu and Trichakis [2013] to two-stage ARO problems, and provide a tractable method for finding these Pareto robustly optimal solutions. We then define a notion of *relaxed Pareto efficiency* that allows us

to systematically trade-off worst-case performance for average case performance. By doing so, we can obtain superior designs which further reduce the price of flexibility.

4.3.1 Pareto Efficiency for Two-stage Problems

In order to motivate the need to consider Pareto efficiency, consider the following example that demonstrates that giving consideration to demand scenarios other than the worst-case can improve the performance of the final chosen designs:

Example 3. Consider a setting with five plants and five products. We set the price $p_j = 1$ for all products, and set $T_{ij} = 0$. Assume that a dedicated design is already in place, and we are interested in whether it is worth adding flexibility ($B_{ij} = 9$). We will use a budget uncertainty set, with $\mu_j = 100$, $\sigma_j = 50$, and $\Gamma = 3$.

If we now solve the ARO model (4.6), we find that there are multiple optimal solutions. In particular, the dedicated design (i.e., no additional flexibility) and a long chain design both obtain an optimal objective function value of 0.7, making the correct choice of design ambiguous. To distinguish between the two, we can consider other demand scenarios and the objective function values of the two designs under those scenarios. For example:

- A "mixed" demand scenario $\mathbf{d}' = (50, 150, 75, 125, 100)$: long chain design value is 0.91, and the dedicated design value is 0.85.
- A "low" demand scenario $\mathbf{d}' = (50, 50, 75, 100, 125)$: long chain design value is approximately 0.89, and the dedicated design value is approximately 0.94.

In this example, we were able to enumerate the worst-case optimal designs, and then evaluate their performance individually by fixing demand scenarios and reoptimizing. We now formalize this process by first considering the generic two-stage ARO problem

$$z^* = \max_{\mathbf{x}, \mathbf{y}(\mathbf{d})} \min_{\mathbf{d} \in \mathcal{U}} f(\mathbf{d}, \mathbf{x}, \mathbf{y}(\mathbf{d})) \tag{4.12}$$
$$\text{subject to } (\mathbf{x}, \mathbf{y}(\mathbf{d})) \in X(\mathbf{d}) \quad \forall \mathbf{d} \in \mathcal{U},$$

which is a generalization of (4.11), and where z^* is the optimal objective value.

Definition 1. A *Pareto optimal adaptive solution* is an optimal solution $(\mathbf{x}, \mathbf{y}(\mathbf{d}))$ to (4.12) such that there is no other optimal solution $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}(\mathbf{d}))$ such that $f(\mathbf{d}, \tilde{\mathbf{x}}, \tilde{\mathbf{y}}(\mathbf{d})) \geq f(\mathbf{d}, \mathbf{x}, \mathbf{y}(\mathbf{d}))$ for all $\mathbf{d} \in \mathcal{U}$ and there does not exist a $\tilde{\mathbf{d}} \in \mathcal{U}$ such that $f(\tilde{\mathbf{d}}, \tilde{\mathbf{x}}, \tilde{\mathbf{y}}(\tilde{\mathbf{d}})) > f(\tilde{\mathbf{d}}, \mathbf{x}, \mathbf{y}(\tilde{\mathbf{d}}))$.

In the above example both designs are Pareto optimal adaptive solutions, as neither dominates the other across all demand scenarios.

We use affine adaptability to create a tractable relaxation of the "fully" adaptive original ARO problem. We can restate (4.12) by substituting in an affine policy (4.5) to obtain

$$\begin{aligned} z^* = \max_{\mathbf{x},\mathbf{Q},\mathbf{q}} \min_{\mathbf{d}\in\mathcal{U}} \; & f(\mathbf{d}, \mathbf{x}, \mathbf{Q}, \mathbf{q}) \\ \text{subject to } & (\mathbf{x}, \mathbf{Q}, \mathbf{q}) \in X(\mathbf{d}) \quad \forall \mathbf{d} \in \mathcal{U}, \end{aligned} \tag{4.13}$$

and thus obtain a new definition for this specific case:

Definition 2. A *Pareto optimal affine adaptive solution* is an optimal solution $(\mathbf{x}, \mathbf{Q}, \mathbf{q})$ to (4.13) such that there is no other optimal solution $(\tilde{\mathbf{x}}, \tilde{\mathbf{Q}}, \tilde{\mathbf{q}})$ such that $f(\mathbf{d}, \tilde{\mathbf{x}}, \tilde{\mathbf{Q}}, \tilde{\mathbf{q}}) \geq f(\mathbf{d}, \mathbf{x}, \mathbf{Q}, \mathbf{q})$ for all $\mathbf{d} \in \mathcal{U}$ and there does not exist a $\tilde{\mathbf{d}} \in \mathcal{U}$ such that $f(\tilde{\mathbf{d}}, \tilde{\mathbf{x}}, \tilde{\mathbf{Q}}, \tilde{\mathbf{q}}) > f(\tilde{\mathbf{d}}, \mathbf{x}, \mathbf{Q}, \mathbf{q})$.

The affine policy substitution reduces the problem to nearly the same definition as provided by Iancu and Trichakis [2013], with the exception that in this model both the constraints and the objective function contain uncertain parameters.

We can now describe a general method to find a Pareto optimal affine adaptive solution. We first solve the ARO problem (4.6) to obtain the worst-case objective

function value z^*. We then solve a second optimization problem

$$\begin{aligned}
\max_{\mathbf{x},\mathbf{Q},\mathbf{q}} \quad & \sum_{i,j} (p_j - T_{ij}) \left(\mathbf{Q}_{ij}^T \hat{\mathbf{d}} + q_{ij} \right) - \sum_{i,j} B_{ij} x_{ij} && (4.14) \\
\text{subject to} \quad & \sum_{i,j} (p_j - T_{ij}) \left(\mathbf{Q}_{ij}^T \mathbf{d} + q_{ij} \right) - \sum_{i,j} B_{ij} x_{ij} \geq \sum_j p_j d_j z^* && \forall \mathbf{d} \in \mathcal{U} \\
& \sum_j \left(\mathbf{Q}_{ij}^T \mathbf{d} + q_{ij} \right) \leq c_i && \forall i,\ \forall \mathbf{d} \in \mathcal{U} \\
& \sum_i \left(\mathbf{Q}_{ij}^T \mathbf{d} + q_{ij} \right) \leq d_j && \forall j,\ \forall \mathbf{d} \in \mathcal{U} \\
& 0 \leq \mathbf{Q}_{ij}^T \mathbf{d} + q_{ij} \leq d_j x_{ij} && \forall i, j,\ \forall \mathbf{d} \in \mathcal{U} \\
& \mathbf{x} \in \{0,1\}^{n \times m},
\end{aligned}$$

where $\hat{\mathbf{d}}$ is a feasible demand scenario drawn from $\mathcal{U}$. The first constraint ensures that a solution to this problem is no worse in the worst-case than the solution to (4.6), and the objective function ensures that we are not dominated by another design for the selected demand scenario. We claim that a solution to this second problem will be a Pareto optimal affine adaptive solution.

To understand why solving (4.14) is sufficient, consider the optimization problem

$$\begin{aligned}
\max_{\mathbf{x},\mathbf{Q},\mathbf{q}} \quad & \sum_{k=1}^{m} \left[\sum_{i,j} (p_j - T_{ij}) \left(\mathbf{Q}_{ij}^T \mathbf{d}_k + q_{ij} \right) - \sum_{i,j} B_{ij} x_{ij} \right] && (4.15) \\
\text{subject to} \quad & \sum_{i,j} (p_j - T_{ij}) \left(\mathbf{Q}_{ij}^T \mathbf{d} + q_{ij} \right) - \sum_{i,j} B_{ij} x_{ij} \geq \sum_j p_j d_j z^* && \forall \mathbf{d} \in \mathcal{U} \\
& \sum_j \left(\mathbf{Q}_{ij}^T \mathbf{d} + q_{ij} \right) \leq c_i && \forall i,\ \forall \mathbf{d} \in \mathcal{U} \\
& \sum_i \left(\mathbf{Q}_{ij}^T \mathbf{d} + q_{ij} \right) \leq d_j && \forall j,\ \forall \mathbf{d} \in \mathcal{U} \\
& 0 \leq \mathbf{Q}_{ij}^T \mathbf{d} + q_{ij} \leq d_j x_{ij} && \forall i, j,\ \forall \mathbf{d} \in \mathcal{U} \\
& \mathbf{x} \in \{0,1\}^{n \times m},
\end{aligned}$$

where $\mathbf{d}_1, \ldots, \mathbf{d}_{m+1}$ are affinely independent demand scenarios in $\mathcal{U} \subset \mathbb{R}^m$.

Theorem 2. *Optimal solutions to Problem* (4.15) *are Pareto optimal affine adaptive*

solutions.

Proof of Theorem 2. First, we note that the first constraint of (4.15) is sufficient to constrain us to only robustly optimal affine solutions, as we can be no worse than z^* under any scenario. We now proceed to prove it is Pareto optimal by contradiction. Let $(\mathbf{x}, \mathbf{Q}, \mathbf{q})$ be an optimal solution, but not a Pareto optimal one. Therefore, there exists an optimal solution $(\tilde{\mathbf{x}}, \tilde{\mathbf{Q}}, \tilde{\mathbf{q}})$ that has an objective function value greater than or equal to the objective function value for $(\mathbf{x}, \mathbf{Q}, \mathbf{q})$ for all $\mathbf{d} \in \mathcal{U}$, and is strictly greater for at least one scenario. Equivalently, the inequality

$$\sum_{i,j} (p_j - T_{ij}) \left(\mathbf{Q}_{ij}^T \mathbf{d} + q_{ij}\right) - \sum_{i,j} B_{ij} x_{ij} \leq \sum_{i,j} (p_j - T_{ij}) \left(\tilde{\mathbf{Q}}_{ij}^T \mathbf{d} + \tilde{q}_{ij}\right) - \sum_{i,j} B_{ij} \tilde{x}_{ij} \tag{4.16}$$

holds for all $\mathbf{d} \in \mathcal{U}$. However, as we are optimizing for (4.15), we know that the following $m+1$ equalities must hold for $\mathbf{d}_1, \ldots, \mathbf{d}_{m+1}$ for an optimal solution:

$$\begin{aligned}
\sum_{i,j} (p_j - T_{ij}) \left(\mathbf{Q}_{ij}^T \mathbf{d}_1 + q_{ij}\right) - \sum_{i,j} B_{ij} x_{ij} &= \sum_{i,j} (p_j - T_{ij}) \left(\tilde{\mathbf{Q}}_{ij}^T \mathbf{d}_1 + \tilde{q}_{ij}\right) - \sum_{i,j} B_{ij} \tilde{x}_{ij} \\
\sum_{i,j} (p_j - T_{ij}) \left(\mathbf{Q}_{ij}^T \mathbf{d}_2 + q_{ij}\right) - \sum_{i,j} B_{ij} x_{ij} &= \sum_{i,j} (p_j - T_{ij}) \left(\tilde{\mathbf{Q}}_{ij}^T \mathbf{d}_2 + \tilde{q}_{ij}\right) - \sum_{i,j} B_{ij} \tilde{x}_{ij} \\
\vdots \qquad &= \qquad \vdots \\
\sum_{i,j} (p_j - T_{ij}) \left(\mathbf{Q}_{ij}^T \mathbf{d}_{m+1} + q_{ij}\right) - \sum_{i,j} B_{ij} x_{ij} &= \sum_{i,j} (p_j - T_{ij}) \left(\tilde{\mathbf{Q}}_{ij}^T \mathbf{d}_{m+1} + \tilde{q}_{ij}\right) - \sum_{i,j} B_{ij} \tilde{x}_{ij}
\end{aligned}$$

Recall that any $\mathbf{d} \in \mathcal{U} \subset \mathbb{R}^m$ can be generated by $m+1$ affinely independent vectors, for example $\mathbf{d}_1, \ldots, \mathbf{d}_{m+1}$. Since the two solutions $(\mathbf{x}, \mathbf{Q}, \mathbf{q})$ and $(\tilde{\mathbf{x}}, \tilde{\mathbf{Q}}, \tilde{\mathbf{q}})$ take the same value for $m+1$ affinely independent points in $\mathcal{U}$, and the objective function is an affine function of $\mathbf{d}$, they must take the same value for all other points in $\mathcal{U}$ as linear combination of $\mathbf{d}_1, \cdots, \mathbf{d}_{m+1}$. Therefore, the inequality (4.16) cannot be strict for any $\mathbf{d} \in \mathcal{U}$ and we obtain a contradiction. □

Using Theorem 2 we can then obtain the following result, which justifies our method for finding Pareto solutions:

Corollary 1. *Optimal solutions to Problem* (4.14) *are Pareto optimal affine adaptive solutions.*

Proof of Corollary 1. Any interior point $\hat{\mathbf{d}} \in \mathcal{U}$ can be written as a linear combination of $m+1$ affinely independent demand scenarios in $\mathcal{U}$. Therefore, the optimization problem (4.14) reduces to (4.15). □

Solving (4.14) is no more difficult than solving the initial optimization problem, and may in fact be easier as a optimal solution to (4.6) is a feasible solution to (4.14). In Section 4.4.3, we quantify the degree to which the simulation performance of Pareto optimal designs improves over initial designs, and how often these improvements occur.

4.3.2 Relaxed Pareto Efficiency

The applicability of the above approach rests on there being multiple designs with the same worst-case performance to select among. In practice, this may not occur due to idiosyncrasies of a given instance. For example, consider the Example 3 but with slightly perturbed costs $B_{ij} = 9.5$. This leads to a worst-case objective value for the long chain design of approximately 0.696, while the dedicated design has a value of 0.700. On this basis, the long chain design is not a robustly optimal solution, leaving the dedicated design as the only option that our model will consider.

To address this problem we suggest a simple, pragmatic extension to our method for finding Pareto optimal designs. In particular, instead of searching for a non-dominated solution amongst all robust optimal solutions, we will instead search amongst all "α-robust" optimal solutions, where $\alpha \in (0, 1]$ is the degree to which we are willing to relax the requirement that we are no worse in the worst-case. For-

mally, we wish to find solutions to the problem:

$$\max_{\mathbf{x},\mathbf{Q},\mathbf{q}} \sum_{i,j} (p_j - T_{ij}) \left(\mathbf{Q}_{ij}^T \hat{\mathbf{d}} + q_{ij} \right) - \sum_{i,j} B_{ij} x_{ij} \tag{4.17}$$

$$\begin{aligned}
\text{subject to } & \sum_{i,j} (p_j - T_{ij}) \left(\mathbf{Q}_{ij}^T \mathbf{d} + q_{ij} \right) - \sum_{i,j} B_{ij} x_{ij} \geq \sum_j p_j d_j \alpha z^* && \forall \mathbf{d} \in \mathcal{U} \\
& \sum_j \left(\mathbf{Q}_{ij}^T \mathbf{d} + q_{ij} \right) \leq c_i && \forall i,\ \forall \mathbf{d} \in \mathcal{U} \\
& \sum_i \left(\mathbf{Q}_{ij}^T \mathbf{d} + q_{ij} \right) \leq d_j && \forall j,\ \forall \mathbf{d} \in \mathcal{U} \\
& 0 \leq \mathbf{Q}_{ij}^T \mathbf{d} + q_{ij} \leq d_j x_{ij} && \forall i, j,\ \forall \mathbf{d} \in \mathcal{U} \\
& \mathbf{x} \in \{0,1\}^{n \times m},
\end{aligned}$$

where $\hat{\mathbf{d}}$ is a given demand scenario in the interior of $\mathcal{U}$, and z^* is the optimal objective value to (4.6). We will refer to the designs produced by this optimization as *relaxed Pareto optimal designs*. We present computational evidence that this relaxation can lead to higher-quality designs than non-relaxed Pareto-efficient designs in Section 4.4.3, reducing the price of flexibility even further.

4.4 Insights from Computations

In this section, we detail the performance of the designs produced by model (4.6). Before doing so, we first establish that model (4.6) is *tractable*, which we define as allowing us to quickly solve instances of practical size on commodity hardware (Section 4.4.1). In particular, we show that we can obtain near-provably-optimal solutions for instances with 15 plants and 15 products in about an hour on a laptop computer. We derive insight into the solutions produced by our model by considering the case of homogeneous parameters. We find the results match our intuition and observations made previously in the literature (Section 4.4.2).

We compare the initial designs produced by model (4.6) with their Pareto-efficient counterparts, and show that Pareto-efficient designs usually improve over the initial design for little extra effort (Section 4.4.3). Finally we investigate how our designs

perform when the true distribution of the demand is substantially different from the distribution we had estimated (Section 4.4.4). We compare different Pareto efficient designs for various values of Γ in (4.7), as well as with designs produced by a stochastic programming model similar to that presented in Mak and Shen [2009]. We find that, despite optimizing for a worst-case objective, we improve slightly over the stochastic programming designs when there is no difference between the estimated and true distribution, and improve substantially when there is.

Finally, we demonstrate in Section 4.4.5 that our overall approach produces solutions that substantially reduce the price of flexibility compared to standard designs like the long chain design, while maintaining the same level of revenue.

Throughout this section we will primarily use two summary statistics to evaluate simulation performance: the arithmetic mean of the measured quantity, and the *Conditional Value at Risk* [Bertsimas and Brown, 2009] at the 10% level ("$CVaR_{10\%}$"), which we take as the arithmetic mean of our metric over the worst 10% of scenarios. The use of $CVaR$ allows us to capture worst-case performance without being unduly affected by the most extreme cases.

4.4.1 Tractability of Adaptive Formulation

As described in Section 4.2, model (4.6) is an ARO problem with binary here-and-now variables and continuous wait-and-see variables. As we are using a polyhedral "budget" uncertainty set and affine adaptability, we can reformulate this problem to obtain a robust mixed-integer linear optimization problem that can be solved with any general-purpose integer optimization solver. To understand the tractability of our approach requires a brief description of the *branch-and-bound* method these solvers employ. In branch-and-bound a continuous relaxation of the problem is solved, giving an upper bound z_{bound} on the problem (as we are maximizing). This bound is tightened progressively by creating subproblems by branching on fractional variables. The best integer solution found so far is stored (with value $z_{integer}$ providing a lower bound), and the usual termination criterion, other than a time limit, is to stop this

branching process when the *bound gap*, defined as

$$\frac{z_{bound} - z_{integer}}{z_{integer}}, \tag{4.18}$$

falls below a tolerance τ. The rate at which the two bounds change depends on the structure of the problem. For example, a good integer solution may be found quickly but it may take a very long time to prove optimality of the solution. In Figure 4-2, we visualize the progress of these bounds for an instance of the flexibility model: we see that a "good" integer solution is found within 1 second, and a near-optimal solution is found within 6 seconds, but the solver takes approximately 8 seconds to prove that it is within 1% of optimal by successive tightening of the upper bound. We also note that the initial "good" solution was actually within 3% of optimal, although given the bound known at the time we could only guarantee that it was within 13% of optimal.

While proving a design is optimal may be desirable after a model is finalized, of more immediate interest is the time to find "good" solutions. We thus consider the *true gap*

$$\frac{z^*_{integer} - z_{integer}}{z_{integer}}, \tag{4.19}$$

where $z^*_{integer}$ is the objective function value of a design that is within 1% of the optimal solution. As our goal is to find such a solution, we would like to compare the time taken to find it on average with the time it takes us to confirm it. In Figure 4-3, we plot the percentage of instances achieving a true gap of less than 1%, 2%, 5% as a function of time for instances of size $n = m = 7$, and find that 75% of instances have a true gap from optimal of less than 5% within 5 seconds, but only approximately 60% of instances have a "proved" gap less than 1% after 40 seconds. Based on this insight, in Figure 4-4 we show that we can find solutions, for instances of size $n = m = 15$, with a proved gap of less than 5% in approximately 2 hours, and find solutions with a true gap within 2% in approximately 1 hour.

All experiments were conducted on a laptop computer with an Intel Core i7 2.6GHz CPU. All models were implemented using the JuMP [Lubin and Dunning, 2015, Dunning et al., 2015] and JuMPeR (Chapter 6) packages in the Julia programming

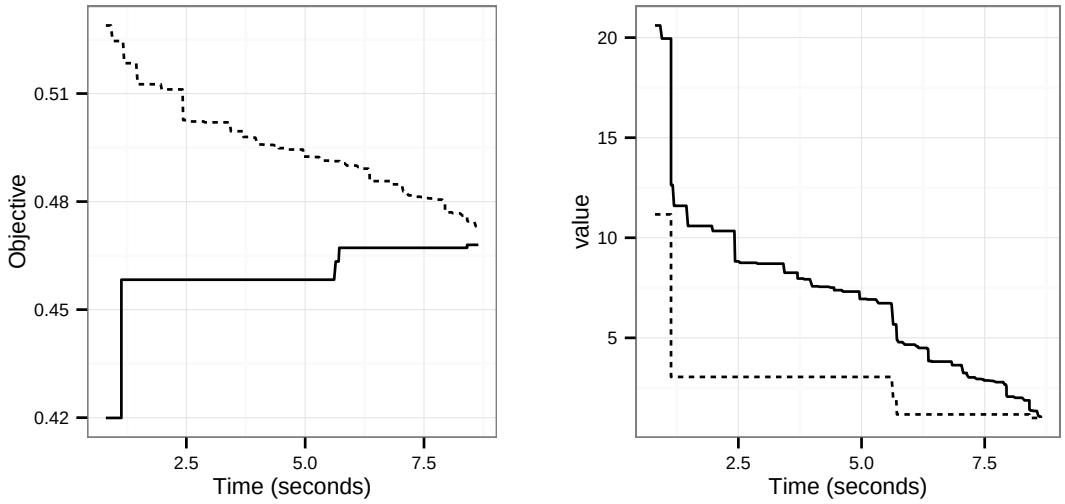

Figure 4-2: Visualization of the progression of a mixed-integer optimization solver for a random instance. Left: the objective value of the solver's best integer solution so far (solid) and the bound (dashed). Right: the bound gap reported by the solver (solid) and the bound gap between between the best integer solution so far and the bound when solver stops (dashed).

language, with Gurobi 6.0.4 used as the solver [Gurobi Optimization Inc., 2016].

4.4.2 Impact of Parameter Selections

Perhaps the most obvious flexibility design, especially in a setting where the number of plants and number of products is equal and products are relatively homogeneous in their profitability and demands, is a design in which each plant produces one product - a "dedicated" design

$$\mathcal{C}_D = \{(1,1),(2,2),\ldots,(n,n)\}\,. \tag{4.20}$$

The natural extension to this design is one in which each plant produces two products and each product is itself produced by two plants. This forms a long cycle

$$\mathcal{C}_L = \{(1,1),(1,2),(2,2),(2,3),\ldots,(n,n),(n,1)\}\,, \tag{4.21}$$

that is referred to as the "long chain" design throughout the literature. To gain an understanding of how model (4.6) values different flexibility designs, we considered these

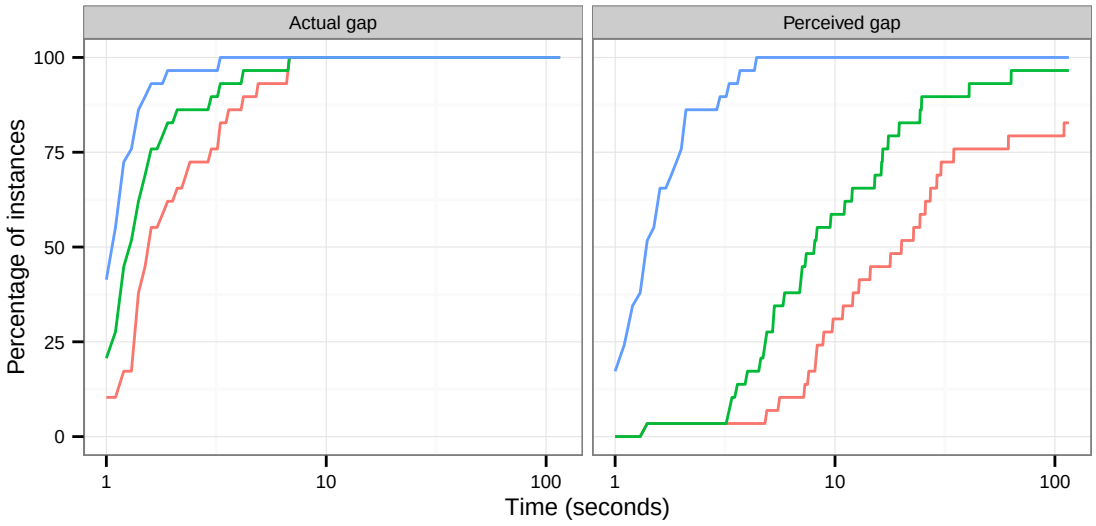

Figure 4-3: Percentage of a set of 50 random instances of size $n = m = 7$ that have a gap lower than 5%, 2%, or 1% (lines, top to bottom) at a given time. Left, the "actual gap": the gap between the best integer solution so far and the final integer solution. Right, the "perceived gap": the gap between the best integer solution so far and the bound at that point in time.

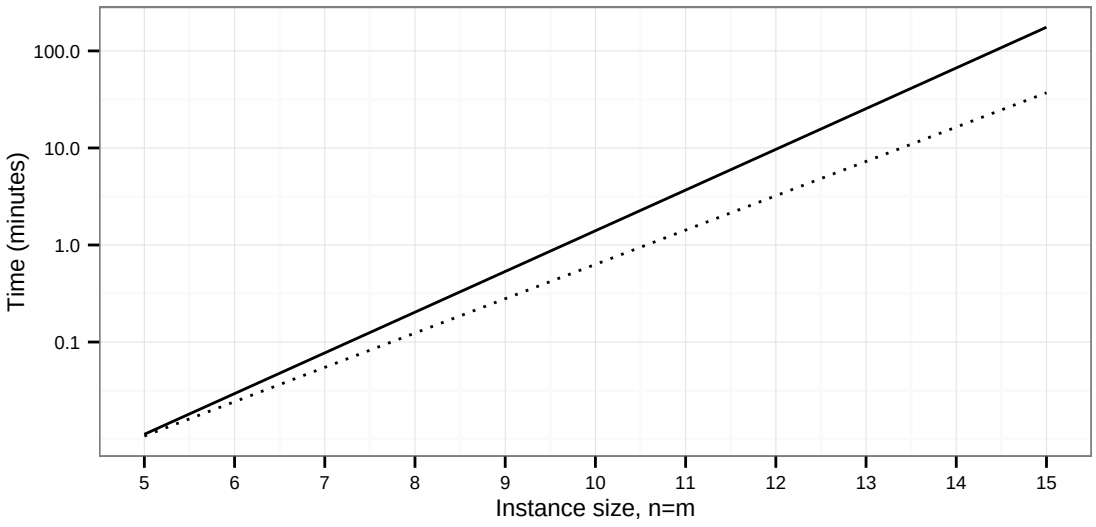

Figure 4-4: Time for 90% of instances to reach a proved bound gap of 5% (solid line), and time for 90% of instances to find a solution within 2% of the eventual integer solution (dashed line).

restricted, homogeneous settings commonly discussed in the literature and analyzed how the model values different designs under different parameter selections.

Our basic model consists of $n = 5$ plants and $m = 5$ products. We use the correlated budget uncertainty set (4.8). We will assume that the "dedicated" design is already built (or equivalently $B_{ii} = 0$), and will set all other flexibility costs to be equal, $B_{ij} = B$. Similarly, we will assume that $\mu_j = \bar{\mu}$, $\sigma_j = \bar{\sigma}$, and $p_j = \bar{p}$. We will always set the capacity equal to the mean demand ($c_i = \bar{\mu}$), and will assume without loss of generality that $T_{ij} = 0$, as it can be included in p, the revenue. Even in this homogenous setting the optimal solutions can vary in structure, but they are generally either similar to a long chain design or a dedicated design. To simplify the analysis, we will fix the design variables $\mathbf{x}$ to each of the designs in turn, and then solve for the continuous variables to determine the objective function value.

Flexibility cost versus Γ

We first consider the relationship between the cost of adding flexibility, B, with the uncertainty set parameter Γ. To do so, we fix $\bar{p} = 1$, $\bar{\mu} = 100$, and $\bar{\sigma} = 50$. We then vary $B \in \{1, \ldots, 18\}$ and $\Gamma \in \{0, 0.2, \ldots, n\}$, and record the objective function values for the long chain design and dedicated designs. In Figure 4-5, we plot the ratio of the objective values ($z_{\mathcal{C}_D}/z_{\mathcal{C}_L}$) with respect to the two varying parameters B and Γ. We first note that the maximum flexibility cost for which the long chain is better than dedicated is approximately 17. The maximum revenue that could be obtained per product in this setting is 50 ($\bar{p}\bar{\sigma}$), which matches the intuition that the cost of additional flexibility should be less than the extra profits we could possibly hope to capture by having the flexibility. As Γ varies between 0 and 1, the range of B for which the long chain solution is better increases, before decreasing again as Γ ranges from 1 to 5. This again matches intuition: if the variability is high, then there is little more to gain from the long chain as we will either exhaust our capacity easily, or we will be able to satisfy each product from one plant, leaving less room for the long chain design to be beneficial.

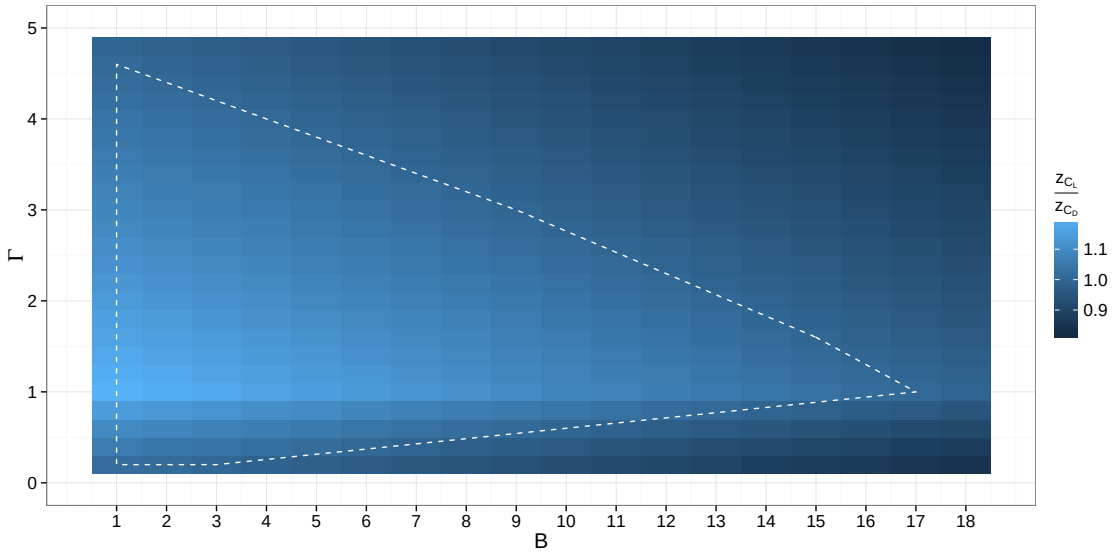

Figure 4-5: Ratio of the objective function value for a "long chain" design versus a "dedicated" design for varying flexibility costs B and uncertainty set budget Γ. The dotted line marks the parameter range in which the long chain design is better than the dedicated design.

Flexibility cost versus correlation

We also consider how correlation between products affects the design choice. Suppose the mean and standard deviation of demand for each product is again $\bar{\mu}$ and $\bar{\sigma}$, respectively, but the demand for each product has a pairwise correlation of ρ with the other products. The covariance matrix Σ thus has diagonal elements $\Sigma_{i,i} = \sigma^2$, with off-diagonals of $\rho\sigma^2$. We again fix $\bar{p} = 1$, $\bar{\mu} = 100$, and $\bar{\sigma} = 50$. We then vary $B \in \{1, \ldots, 18\}$ and $\rho \in \{0, 0.05, \ldots, 0.5\}$, with Γ fixed to 1.

In Figure 4-6, we plot the same quantity as before. In the case where $\rho = 0$ (no correlation) we return to the same setting as above. However, for any fixed B we note that as ρ increases the long chain design becomes less attractive. For values of ρ above approximately 0.4, we always prefer the dedicated design over the long chain design. This matches intuition, as at least in the limiting case where $\rho = 1$, we would expect to gain no benefit from the long chain design as there would be no variability in demand between products. This relationship was also noted by Jordan and Graves [1995] for their model as well: "... the overall benefits of flexibility [are] reduced as

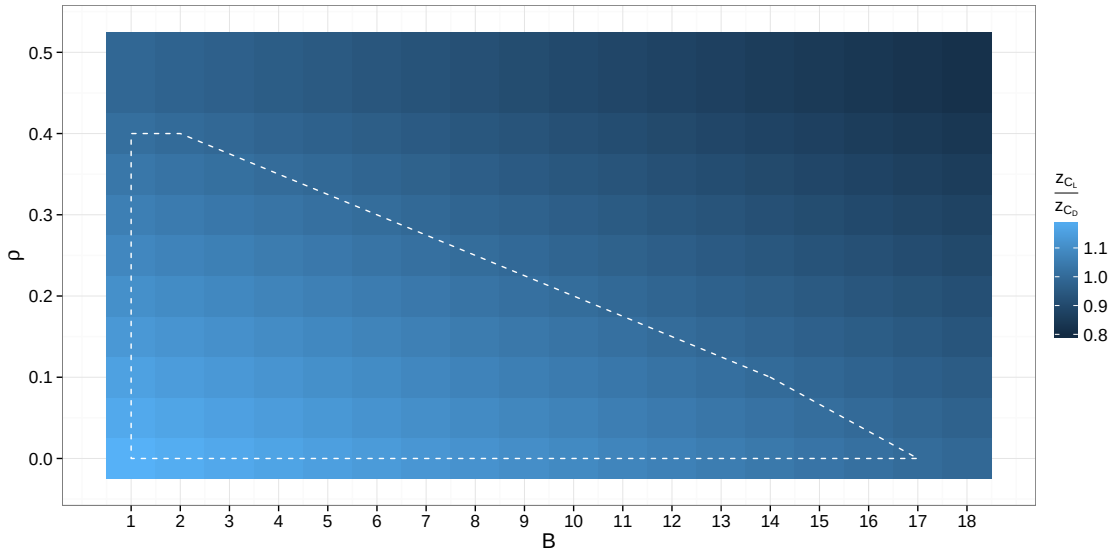

Figure 4-6: Ratio of the objective function value for a "long chain" design versus a "dedicated" design for varying flexibility costs B and pair-wise demand correlation ρ. The dotted line marks the parameter range in which the long chain design is better than the dedicated design.

product demand correlation increases."

These figures provide some insight into the designs produced by model (4.6). In particular, we see that the designs are intuitive with respect to the parameters, and that they are similar in spirit to previous observations in the literature. However, our model excels in the more general nonhomogenous setting where analytical solutions are not possible, as demonstrated in the following sections.

4.4.3 Utility of Pareto Efficient Designs

We presented the notion of Pareto efficient designs in Section 5.4. In brief we are interested in finding designs that are not only robust with respect to the objective function, but that also perform well in non-worst-case demand scenarios. To find these designs we first solve model (4.6), and then seek to find a (possibly) different design with the same worst-case objective value but with non-dominated performance for another scenario (like the nominal or mean scenario). We also presented a relaxation of the worst-case criterion, in which the "relaxed Pareto" design only needs to achieve

$\alpha \in (0, 1]$ of the worst-case objective value of the initial design: in other words, we exchange worst-case performance for possible improvements in the alternative scenario. In this section, we demonstrate how often we obtain a Pareto or relaxed-Pareto efficient design that improves over the initial (possibly Pareto) design, and to what degree these alternative designs improve over the initial designs.

We considered 100 random instances drawn from the following family of instances: $n = m = 5$, $B_{ij} \sim Uniform(50, 350)$, $B_{ii} = 0$, $p \sim Uniform(10, 20)$, $T_{ij} = 0$, $\mu_j \sim Uniform(150, 250)$, $\sigma_j = \frac{1}{2}\mu_j$, and $c_i = \mu_i$. For each instance we generated 100 demand scenarios, where the demand d_j for product j is drawn from $\max\{Normal(\mu_j, \sigma_j), 0\}$. For each instance and demand scenario a deterministic problem was solved to find the best-possible profit that could be obtained if we had clairvoyant knowledge of the demand. This naturally represents an upper bound on the profit that can be obtained by any design for a given demand scenario.

We considered two values of α: the relaxed-Pareto parameter 0.8, and the standard Pareto parameter 1, i.e., not relaxed at all. The scenario used to find a Pareto efficient design was the mean demand scenario, $d_j = \mu_j$. We used a budget uncertainty set (4.7) with either $\Gamma = 1$ or $\Gamma = 3$. For each of the instances and demand scenarios we thus have six realized profits: one for each of two Pareto efficient designs (for $\alpha = 0.8$ or 1) and the initial (possibly non-Pareto) design (solution to (4.6)), and for the two values of Γ. These profits were all normalized by the best possible profit obtained from solving the deterministic problem described above.

The results are depicted in Figure 4-7. The mean and $CVaR$ of the normalized profits across the demand scenarios for each instance are sorted from lowest to highest independently for each design. The left figure, showing the mean performance, reveals that there is little difference for the relaxed-Pareto and Pareto efficient designs in the $\Gamma = 3$ case, but that both have marginally better profitability than the initial design. For $\Gamma = 1$ we see more substantial differentiation between the designs, with the relaxed-Pareto design performing slightly better than the Pareto design. The figures for the $CVaR$ metric are similar in appearance, although the absolute magnitudes of the differences are more substantial than for the mean. For both $\Gamma = 1$ and $\Gamma = 3$

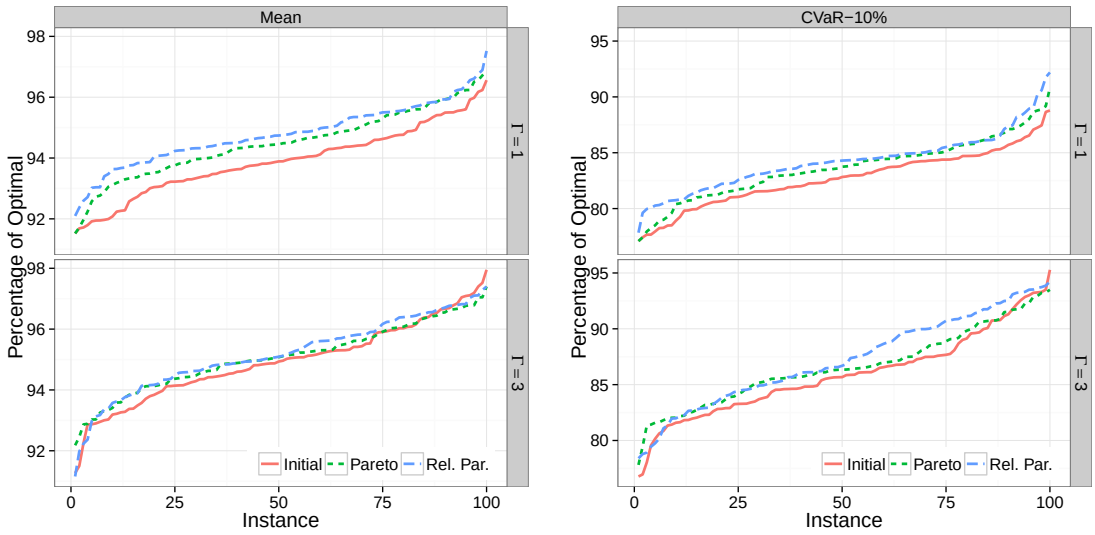

Figure 4-7: Results for evaluation of Pareto efficient designs (Section 4.4.3). Each plot displays the mean (left) or $CVaR_{10\%}$ (right) of simulated normalized profits of the relaxed-Pareto efficient design (purple), Pareto efficient design (red), and initial design (teal) for each random instance, ordered from worst to best independently for each design type. The top figures when the uncertainty set parameter Γ is set to 1, and bottom are for $\Gamma = 3$.

the relaxed-Pareto design outperforms both alternatives.

Figure 4-7 summarizes the overall performance of the designs, but does not capture on a per-instance basis how Pareto efficient designs compare to the initial designs. To address this, we constructed Table 4.1 which describes, for each of the 100 instances, whether the mean and CVaR of the Pareto efficient designs were the same, worse, or better. We note that the Pareto efficient designs are at least as good as the initial designs in approximately 70% to 95% of instances, with Pareto efficient designs being strictly better in approximately 35% to 75% of instances. The relaxed-Pareto efficient designs are, as expected, different from the initial designs for more instances, and in general they are better than the initial designs more often than the Pareto efficient designs.

4.4.4 Quality of Designs

To quantify the degree to which our designs performed in a realistic setting with both uncertainty in the data and variability in the realized demand. we considered fami-

		Mean			$CVaR_{10\%}$		
Γ	α	Worse	Same	Better	Worse	Same	Better
1	0.8	10	17	73	14	17	69
1	1.0	4	45	51	10	45	45
3	0.8	30	18	52	32	18	50
3	1.0	32	34	34	28	34	38

Table 4.1: Results for evaluation of Pareto efficient design (Section 4.4.3). For each of the 100 random instances considered, the Pareto ($\alpha = 1$) and relaxed-Pareto ($\alpha = 0.8$) efficient designs are either worse, the same, or better than the initial designs in simulation under a mean or CVaR metric of performance. The relaxed-Pareto efficient design is different from the initial design more frequently, and is better than the initial design in over half of the instances by either metric.

lies of random instances where the "true" demand distribution differs from perceived demand. We compare our designs with the designs obtained from the stochastic programming approach in Mak and Shen [2009]. In particular, we implemented a sample-average approximation (SAA) version of the model presented in Mak and Shen [2009]. The method describes a relaxation of that approach, so the results we present should be considered an upper bound on the quality of their solutions that are obtained at the cost of tractability.

We considered 100 instances drawn from the same family of instances described in Section 4.4.3. To investigate the effect of uncertainty in the data we considered three levels of perturbation of the distribution from which demand scenarios are drawn for simulation, $\eta \in \{0.0, 0.5, 0.9\}$. Given η, the mean demand $\hat{\mu}_j$ for product j is drawn from $Uniform\left((1-\eta)\,\mu_j, (1+\eta)\,\mu_j\right)$, where μ_j is the mean that is provided to the optimization model. We then generated 100 random demand scenarios for each instance and level of perturbation: that is, demand d_j is drawn from $\max\left\{Normal\left(\hat{\mu}_j, \sigma_j\right), 0\right\}$.

For each instance we calculated the relaxed-Pareto efficient design ($\alpha = 0.8$) with $\Gamma \in \{0, 1, 2, 3, 4, 5\}$, where $\Gamma = 0$ corresponds to the deterministic problem and $\Gamma = 5$ is equivalent to a "box" uncertainty set. We also calculated the SAA design with an expectation objective. As in Section 4.4.3, we considered the mean and $CVaR$ of profit normalized by the clairvoyant-optimal profit.

We first investigated the effect of Γ on design quality, and the results are displayed

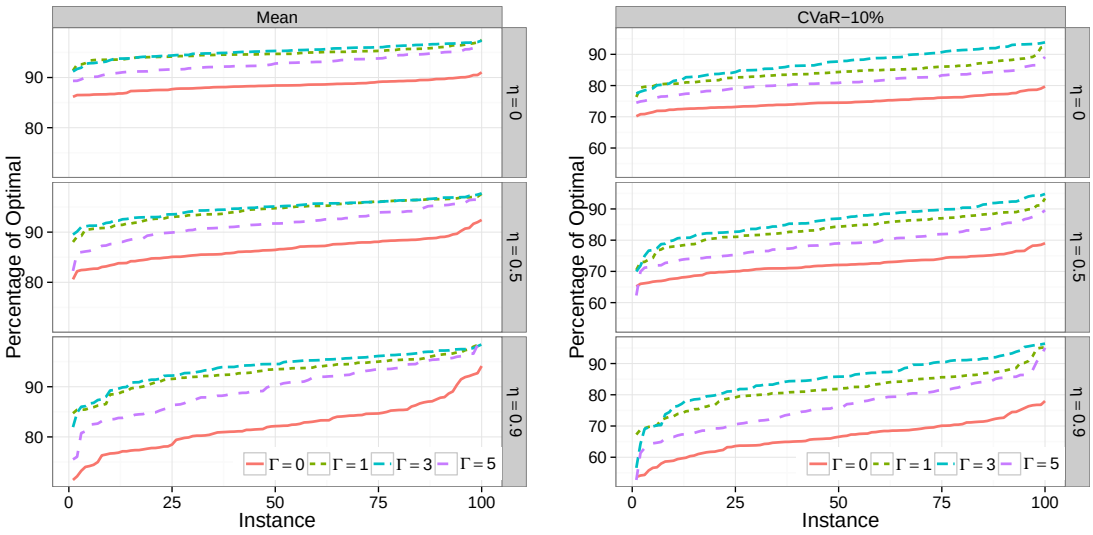

Figure 4-8: Results for evaluation of effect of Γ for different levels of perturbation η (Section 4.4.4). Each plot displays the mean (left) or $CVaR_{10\%}$ (right) of simulated normalized profits for each random instance, ordered from worst to best independently for each design type. Each line represents the designs corresponding to different values of Γ, with the perturbation increasing from top to bottom.

in Figure 4-8. To summarize, we see that by tuning Γ for an "intermediate" level of variability we can obtain good average and worst-case behaviour. In particular, we see that both $\Gamma = 0$ and $\Gamma = 5$ perform poorly in comparison to $\Gamma = 1$ and $\Gamma = 3$ at all perturbation levels by mean performance. However, the $\Gamma = 3$ design distinguishes itself from the $\Gamma = 1$ solution for $CVaR$ performance even without disruption, and performs better than all alternatives.

We then compared the $\Gamma = 3$ designs with the SAA designs in Figure 4-9. Under the no-perturbation setting ($\eta = 0$) there is little difference in mean performance between the two, and similarly for $CVaR$. However, as perturbation increases we see that our designs perform better more often than the SAA designs, with the largest differences occurring at the highest level of disruption.

4.4.5 Price of Flexibility

We established that model (4.6) is tractable (Section 4.4.1), produces intuitive designs (Section 4.4.2), that Pareto efficiency adds value (Section 4.4.3), and that the designs

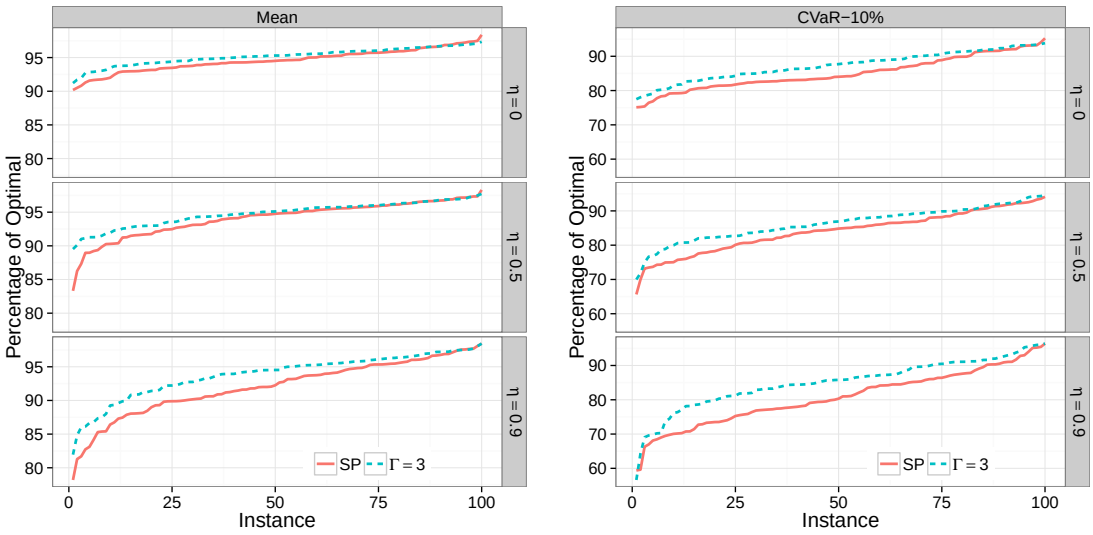

Figure 4-9: Results for the comparison of our designs with SAA designs for different levels of perturbation η (Section 4.4.4). Each plot displays the mean (left) or $CVaR_{10\%}$ (right) of simulated normalized profits for each random instance, ordered from worst to best independently for each design type. The top line represents the designs corresponding to either our $\Gamma = 3$, or the SAA stochastic programming design.

are robust against uncertainty (Section 4.4.4). We now seek to quantify how much our method reduces the price of flexibility.

We will compare the design selected by the relaxed-Pareto model (with parameter $\alpha = 0.8$) with three alternatives: no additional flexibility ("dedicated"), the classic "long chain" design, and a "three chain" design in which each plant produces three products, and vice versa. We evaluate them on 100 instances drawn from the same family of instances described in Section 4.4.3, and simulate random demand scenarios as before.

We are interested in two key metrics for each design: the "price of flexibility" (equal to the cost of the flexibility added $\sum_{ij} B_{ij}x_{ij}$), and total revenue. In the left plot of Figure 4-10 we show these metrics for one of the 100 instances. Note that the price is fixed for each design, but revenue has a distribution. We can see that in this particular instance, the distribution of revenues for the robust design and the two chain designs have a roughly similar distribution. However, the additional costs of the long chain design and the three chain design add only slightly improve revenues

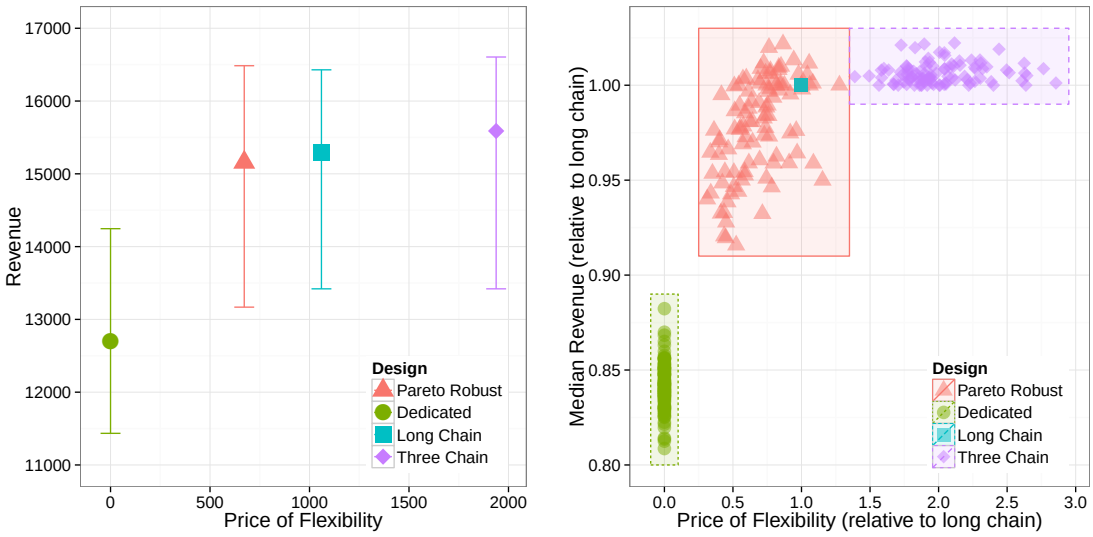

Figure 4-10: Left: comparison of the "price of flexibility" for four different flexibility designs for a representative random instance. The price is the cost of adding flexibility over the "dedicated" design, and revenue is the distribution (first quartile, median, and third quartile) of revenues over 100 demand scenarios. Right: median revenues and prices for all 100 random instances, normalized by the long chain design's median revenue and price.

obtained, which the robust design being the most profitable.

In the right plot of Figure 4-10 we summarize the median revenue and price of flexibility for all 100 instances, normalized by the corresponding revenue and price for the long chain design. We see that the dedicated design achieves approximately 85% of the revenue of the long chain design, and is the cheapest option (by definition). The robust design achieves approximately 98% of the revenue on average of the long chain design, but at only 70% of the cost. The "three chain" design is roughly double the cost of the long chain design, but improves revenues by little more than 1% point. This demonstrates that our method can effectively reduce the price of flexibility without impacting revenues, and therefore drive higher profitability.

4.5 Concluding Remarks

In this chapter, we have demonstrated that we can reduce the price of flexibility by formulating the process flexibility design problem as an adaptive robust optimization

problem. In particular, in simulation experiments our designs often obtained more than 90% of the maximum clairvoyant possible profit. They proved to be robust against considerable uncertainty in the distribution of demand, in terms of both average profit and worst-case profit. Most importantly, we showed that our designs reduce the price of flexibility by 30% versus long chain designs. This was achieved with negligible impact on revenues, and thus profitability was substantially higher for our designs.

Our approach improves over more ad-hoc approaches and heuristics as it explicitly characterizes the trade-off between flexibility and profits (or, equivalently, costs). The use of Pareto-efficient and relaxed Pareto-efficient solutions was a key part of our strategy for producing solutions that perform well in both "worst-case" and "average-case" scenarios.

Our models represent a general tool for obtaining profitable manufacturing process flexibility designs. As we utilize on established techniques such as mixed-integer optimization and robust optimization, our model can be extended to handle a variety of firm-specific constraints and additions. It thus represents a general way to reduce the price of flexibility and improve profitability in settings encountered in practice by manufacturing firms.

Chapter 5

Relative Robust and Adaptive Optimization

5.1 Introduction

Consider a parametric family of optimization problems

$$\begin{aligned} \Pi(\boldsymbol{\xi}) = \max_{\mathbf{x} \in \mathcal{X}} \quad & f(\mathbf{x}, \boldsymbol{\xi}) \\ \text{subject to} \quad & \mathbf{g}(\mathbf{x}, \boldsymbol{\xi}) \leq \mathbf{0}, \end{aligned} \tag{5.1}$$

where the objective function f and constraints $\mathbf{g}$ are functions of both the decision variables $\mathbf{x}$ and parameters $\boldsymbol{\xi}$, and where $\mathcal{X}$ captures constraints on decision variables $\mathbf{x}$ that do not involve parameters $\boldsymbol{\xi}$ such as bounds or restricting decision variables to be integer. We are interested in problems where $\boldsymbol{\xi}$ can be interpreted as a vector of *uncertain* parameters, and so $\Pi(\boldsymbol{\xi})$ can be interpreted as the *clairvoyant* or *offline* optimal objective function value for an "optimization under uncertainty" problem – in other words, it is the best objective function value that could be achieved if the value of the uncertain parameters were somehow known *a priori*. As this is not the case in practice, a decision maker must consider formulations for their problem that capture the nature of the uncertainty and their risk preferences. A popular choice in the operations research literature in the last ten years has been the *robust optimization*

(RO) formulation, i.e.,

$$\begin{aligned} \Pi_{RO} = \max_{\mathbf{x}\in\mathcal{X}} \quad & \min_{\boldsymbol{\xi}\in\Xi} f(\mathbf{x},\boldsymbol{\xi}) \\ \text{subject to} \quad & \mathbf{g}(\mathbf{x},\boldsymbol{\xi}) \leq \mathbf{0} \quad \forall \boldsymbol{\xi} \in \Xi, \end{aligned} \tag{5.2}$$

where Ξ is an *uncertainty set* that contains all the realizations of the uncertain parameters that the solution should be robust against. The solution to this problem is such that it satisfies the constraints for all values of $\boldsymbol{\xi} \in \Xi$, and there is no solution that has a strictly better objective function for all $\boldsymbol{\xi} \in \Xi$. This *absolute* objective function and formulation has some desirable properties, including that it has been shown to be both computationally practical and theoretically tractable for many problem classes and choices of the uncertainty set (see, e.g., the survey by Bertsimas et al. [2011a]). However, the formulation may not be appropriate for all applications due to its inherently conservative nature. For example, if there exists a realization $\hat{\boldsymbol{\xi}} \in \Xi$ such that $f\left(\mathbf{x},\hat{\boldsymbol{\xi}}\right)$ is small, regardless of the particular decision $\mathbf{x}$, then this realization may dominate all other considerations as we have concerned ourselves only with worst-case performance. The result may be that an optimal solution may perform poorer than we would hope on the vast majority of scenarios, even if it technically meets the optimality criterion we have set.

In this paper, we consider two alternatives to the above RO problem that have slightly different objective functions, but can have a significantly different solutions. The first alternative uses a *competitive ratio* (CR) objective function, i.e.,

$$\begin{aligned} \Pi_{CR} = \max_{\mathbf{x}\in\mathcal{X}} \quad & \min_{\boldsymbol{\xi}\in\Xi} \frac{f(\mathbf{x},\boldsymbol{\xi})}{\Pi(\boldsymbol{\xi})} \\ \text{subject to} \quad & \mathbf{g}(\mathbf{x},\boldsymbol{\xi}) \leq \mathbf{0} \quad \forall \boldsymbol{\xi} \in \Xi, \end{aligned} \tag{5.3}$$

and the second a *robust regret* (REG) objective function, i.e.,

$$\begin{aligned} \Pi_{REG} = \max_{\mathbf{x}\in\mathcal{X}} \quad & \min_{\boldsymbol{\xi}\in\Xi} f(\mathbf{x},\boldsymbol{\xi}) - \Pi(\boldsymbol{\xi}) \\ \text{subject to} \quad & \mathbf{g}(\mathbf{x},\boldsymbol{\xi}) \leq \mathbf{0} \quad \forall \boldsymbol{\xi} \in \Xi, \end{aligned} \tag{5.4}$$

where $\Pi(\boldsymbol{\xi})$ is the offline (clairvoyant) optimization problem (5.1). Both of these objective functions are defined as *relative* to the offline objective function value, in contrast to the absolute RO objective function.

Contributions: Our contributions are as follows:

- We show that the Π_{CR} and Π_{REG} problems are theoretically tractable for a large and useful class of problems when $\Pi(\boldsymbol{\xi})$ is a concave function of $\boldsymbol{\xi}$. We propose a duality-based reformulation approach in the case where the offline problem has continuous decision variables, and a cutting plane algorithm for the case where it has some discrete decision variables.

- We show that the methods used for the concave case do not extend to problems where $\Pi(\boldsymbol{\xi})$ is convex in $\boldsymbol{\xi}$, but build on previous work by showing how a practical cutting plane algorithm can be developed by exploiting the structure of the uncertainty set, in particular the popular "budget" uncertainty set of Bertsimas and Sim [2004].

- We explore combining both the absolute robust and competitive ratio objective functions to find Pareto-efficient solutions, and suggest a new way to distinguish between otherwise equivalent solutions to RO problems.

- We perform an in-depth study on three problems that cover a wide variety of problem structures, including single-stage, two-stage, and multistage adaptive RO (ARO) problems. We explore not only the properties of solutions for each of the three different objectives, but also the relative computational practicality of solving these problems as a function of problem size.

Literature review: The study of optimization problems with robust objective functions, both "absolute" and "relative", for the most part started in the early 1990s. The book by Kouvelis and Yu [1997] on "robust discrete optimization" summarizes much of the work on the topic to that point, in particular on problems where there is uncertainty only in the objective function. In that work they discuss two major notions of robust optimization: "minimax" or "absolute robust" (which most recent

literature and this paper refer to as simply "RO"), "minimax regret", which encompasses both "robust deviation" (which we refer to as robust regret), and "relative robust" (which we refer to as competitive ratio). They contend that the deviation and relative criteria are, in general, less conservative than the absolute criterion, and that these criteria tend "...to look at uncertainty as an opportunity to be exploited rather than just as a risk to be hedged against". The book mainly focuses on "minimax regret", with an emphasis on complexity results for various discrete optimization problems. Others at this time also considered applying relative robust optimization to specific problems, both discrete and continuous. For example, Averbakh and Berman [1997] consider the weighted p-center problem on a network with *interval*, or *box*, uncertainty sets on parameters in a "minimax regret" objective function, while Averbakh [2000] develops a more general approach to a class of combinatorial problems with uncertain objective function coefficients that have interval uncertainty. Vairaktarakis [2000] consider the multi-item newsvendor problem with a budget constraints, again with interval uncertainty on the uncertain demand. Mausser and Laguna [1999a] consider robust regret in general LO problems with objective function coefficient uncertainty and interval uncertainty sets. They use an iterative cutting plane method that exploits the structure of optimal solutions to the difficult subproblem of finding the worst-case realizations of the uncertain parameters for given solution. This subproblem is reformulated as a mixed-integer LO problem which is more easily solved, but remained challenging to solve at the time – to partially address this, a complementary heuristic approach is proposed by Mausser and Laguna [1999b].

A separate body of work, in which this current paper is situated, focused initially on uncertain parameters in constraints of mixed-integer convex optimization problems (which subsumes objective function uncertainty as a special case). In particular, Ben-Tal and Nemirovski [1999] and Bertsimas and Sim [2004] consider the case of *ellipsoidal* and *polyhedral* uncertainty sets respectively, and show that these problems have reformulations to deterministic problems which are tractable in both the complexity and computational sense. RO techniques were then applied to multistage optimization problems, which have both *here-and-now* and *wait-and-see* decisions.

These wait-and-see decisions are functions of the uncertain parameters revealed before the decisions must be made, but solving problems with "fully adaptive" wait-and-see decisions has been shown to be intractable in general [Ben-Tal et al., 2004]. Several proposals have been made for approximations of these wait-and-see decisions for adaptive RO (ARO) problems, including affine adaptability for continuous recourse decisions [Ben-Tal et al., 2004], and finite adaptability for both continuous and discrete recourse decisions (Bertsimas and Caramanis [2010], Chapter 3). The objective functions in the recent RO literature are usually "absolute", with some exceptions. For example, Perakis and Roels [2010] apply a "minimax regret" objective function to the classic network revenue management problem, and apply a cutting plane method specialized to their particular problem structure. Assavapokee et al. [2008] consider "deviation robust" (robust regret) objectives in two-stage MILO problems with interval uncertainty sets, and describe a complex three-level optimization problem to produce globally optimal solutions (to a tolerance). Finally, Iancu and Trichakis [2013] present a two-phase process for finding "Pareto" robust solutions: solutions that optimize for the absolute robust objective function, but are in addition not dominated by any other robust solution across the uncertainty set.

Structure:

- In Section 5.2, we discuss the case where the offline problem $\Pi(\boldsymbol{\xi})$ is the optimal objective function value of a linear optimization (LO) problem that is *concave* in $\boldsymbol{\xi}$.
- In Section 5.3, we discuss the case where $\Pi(\boldsymbol{\xi})$ is the optimal objective function value of a LO problem that is *convex* in $\boldsymbol{\xi}$.
- In Section 5.4, we combine the absolute and competitive ratio objective functions to construct a Pareto-efficient set of solutions, and contrast to the proposal of Iancu and Trichakis [2013] for selecting among RO solutions.
- In Section 5.5, we compare and contrast the different objective functions through three cases: a minimum-cost flow problem, an inventory control problem, and a facility location problem.

Problems considered: This paper does not attempt to address the fully general descriptions of Π_{CR} and Π_{REG} given in (5.3) and (5.4), respectively. Instead, we will focus on a broad class of problems that are tractable – if not in always theoretical complexity sense, then in at least in a computational sense. The particular RO problem we consider is

$$\begin{aligned} \Pi_{RO} = \max_{\mathbf{x}\in\mathcal{X}} \quad & \mathbf{f}^T\mathbf{x} + \min_{\boldsymbol{\xi}\in\Xi} \boldsymbol{\xi}^T F\mathbf{x} \\ \text{subject to} \quad & \mathbf{g}_i^T\mathbf{x} + \boldsymbol{\xi}^T G_i\mathbf{x} \leq \boldsymbol{\xi}^T\mathbf{b}_i \quad \forall \boldsymbol{\xi}\in\Xi, \forall i, \end{aligned} \tag{5.5}$$

where $\mathbf{f}$, $\mathbf{g}$, F, and G are known parameters. The structure of the Π_{CR} and Π_{REG} problems we consider are the same as Π_{RO}, except for an adjustment to the objective function by $\Pi(\boldsymbol{\xi})$ (subtracting it for regret, and dividing by it for CR). This structure includes as special cases RO problems with only uncertain objective functions. It also incorporates multistage ARO problems using affine adaptability, as a linear decision rule for a wait-and-see decision $y(\boldsymbol{\xi})$ can be expressed in this format: $y(\boldsymbol{\xi}) = \boldsymbol{\xi}^T\mathbf{g} + h$ (where $\mathbf{g}$ and h are the auxiliary variables that describe the policy). Finite adaptability for ARO problems can be incorporated with a slight modification, as the objective function in that case is the minimum of multiple objective functions of this style. We will consider, for the sake of simplicity, two different uncertainty sets commonly encountered in the literature: a polyhedral uncertainty set

$$\Xi^P = \{\boldsymbol{\xi} \mid H\boldsymbol{\xi} \leq \mathbf{h}\}, \tag{5.6}$$

and an ellipsoidal uncertainty set

$$\Xi^E = \{\boldsymbol{\xi} \mid \boldsymbol{\xi} = H\boldsymbol{\zeta} + \mathbf{h},\ \|\boldsymbol{\zeta}\|_2 \leq 1\}. \tag{5.7}$$

This includes box uncertainty sets as a special case.

Notation: We use lowercase, non-bold face symbols for scalar quantities (e.g., $z \in \mathbb{R}$) and a bold face for vectors (e.g., $\mathbf{x} \in \mathbb{R}^n$). Matrices are denoted by uppercase symbols, e.g., $A \in \mathbb{R}^{m\times n}$. We generally use Ξ to denote an uncertainty set and $\boldsymbol{\xi}$ to

denote uncertain parameters. We also use Π to refer to the optimal objective value of an optimization problem.

5.2 $\Pi(\boldsymbol{\xi})$ is a Concave Function of $\boldsymbol{\xi}$

In this section, we consider offline (clairvoyant) problems $\Pi(\boldsymbol{\xi})$ of the form

$$\begin{aligned} \Pi(\boldsymbol{\xi}) = \max_{\mathbf{x} \geq \mathbf{0}} \quad & \mathbf{c}^T \mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} = \boldsymbol{\xi}, \end{aligned} \tag{5.8}$$

which is a LO problem with a parametric right-hand-side vector. There exists an equivalent dual minimization problem that has a parametric objective function instead, i.e.,

$$\begin{aligned} \Pi^D(\boldsymbol{\xi}) = \min_{\boldsymbol{\pi}} \quad & \boldsymbol{\xi}^T \boldsymbol{\pi} \\ \text{subject to} \quad & A^T \boldsymbol{\pi} \geq \mathbf{c}, \end{aligned} \tag{5.9}$$

and, by strong duality for LO problems, $\Pi(\boldsymbol{\xi}) = \Pi^D(\boldsymbol{\xi})$. We make the assumption that $\Pi(\boldsymbol{\xi})$ is feasible for all $\boldsymbol{\xi} \in \Xi$ – if it was not, then the robust version would be infeasible over at least the same $\boldsymbol{\xi}$. Through the dual problem we can observe that $\Pi(\boldsymbol{\xi})$ is equivalent to minimizing over a finite number of linear functions (the basic feasible solutions to the dual problem), and thus $\Pi(\boldsymbol{\xi})$ is a piecewise linear *concave* function of $\boldsymbol{\xi}$ (see, e.g., Bertsimas and Tsitsiklis [1997]).

We first address the robust regret problem Π_{REG}, and present a reformulation for both polyhedral and ellipsoidal uncertainty sets. We then present the competitive ratio variant, which is a small change from the regret problem. Finally, we provide an alternative *cutting plane* approach for the case where $\Pi(\boldsymbol{\xi})$ is a mixed-integer LO (MILO) problem.

5.2.1 Reformulation of Robust Regret Problem

Recall that the objective function for Π_{REG} is

$$\max_{\mathbf{x}\in\mathcal{X}} \min_{\boldsymbol{\xi}\in\Xi} \mathbf{f}^T\mathbf{x} + \boldsymbol{\xi}^T F\mathbf{x} - \Pi(\boldsymbol{\xi}), \tag{5.10}$$

where $\mathbf{f}$ and F are data. We define the "inner" optimization problem for Π_{REG} to be

$$\Pi^I_{REG}(\mathbf{x}) = \min_{\boldsymbol{\xi}\in\Xi} \mathbf{x}^T F^T\boldsymbol{\xi} - \Pi(\boldsymbol{\xi}), \tag{5.11}$$

the tractability of which depends strongly on the nature of $\Pi(\boldsymbol{\xi})$. Note that $\mathbf{f}$ only appears in the deterministic term $\mathbf{f}^T\mathbf{x}$, which we do not include in the inner problem as it is constant with respect to $\boldsymbol{\xi}$. We are considering here just one family of possible $\Pi(\boldsymbol{\xi})$, so we can proceed by substituting (5.8) into (5.11) and rearranging terms,

$$\begin{aligned}
\Pi^I_{REG}(\mathbf{x}) &= \min_{\boldsymbol{\xi}\in\Xi} \mathbf{x}^T F^T\boldsymbol{\xi} - \max_{A\mathbf{y}=\boldsymbol{\xi},\mathbf{y}\geq\mathbf{0}} \mathbf{c}^T\mathbf{y} && \text{(5.12a)}\\
&= \min_{\boldsymbol{\xi}\in\Xi} \mathbf{x}^T F^T\boldsymbol{\xi} + \min_{A\mathbf{y}=\boldsymbol{\xi},\mathbf{y}\geq\mathbf{0}} -\mathbf{c}^T\mathbf{y} && \text{(5.12b)}\\
&= \min_{\boldsymbol{\xi}\in\Xi, A\mathbf{y}=\boldsymbol{\xi},\mathbf{y}\geq\mathbf{0}} \mathbf{x}^T F^T\boldsymbol{\xi} - \mathbf{c}^T\mathbf{y}, && \text{(5.12c)}
\end{aligned}$$

to obtain a new optimization problem. As $\Pi(\boldsymbol{\xi})$ is a concave function, $-\Pi(\boldsymbol{\xi})$ in (5.11) is a *convex* term, and thus Π^I_{REG} is a convex optimization problem (assuming Ξ is a convex set). If Ξ is polyherdal (Ξ^P) then we can also determine that Π^I_{REG} is a piecewise linear *concave* function of $\mathbf{x}$, as we are minimizing a parametric objective function as in (5.9). As the "outer" robust regret optimization problem is a maximization problem, a concave $\Pi^I_{REG}(\mathbf{x})$ is ideal for tractability.

To complete the reformulation we substitute the rearranged "inner" problem into the "outer" problem, which we can do by first formulating the dual of the inner problem. Consider the case of the polyhedral uncertainty set $\Xi^P = \{\boldsymbol{\xi} \mid H\boldsymbol{\xi} \leq \mathbf{h}\}$,

i.e.,

$$\begin{aligned} \Pi^{I}_{REG}(\mathbf{x}) = \min_{\mathbf{y}\geq\mathbf{0},\boldsymbol{\xi}} \quad & \mathbf{x}^T F^T \boldsymbol{\xi} - \mathbf{c}^T \mathbf{y} \\ \text{subject to} \quad & H\boldsymbol{\xi} \leq \mathbf{h} \\ & A\mathbf{y} = \boldsymbol{\xi}. \end{aligned} \tag{5.13}$$

We can now take the dual of (5.13) to obtain an equivalent maximization problem,

$$\begin{aligned} \Pi^{I,D}_{REG}(\mathbf{x}) = \max_{\boldsymbol{\alpha}\leq\mathbf{0},\boldsymbol{\beta}} \quad & \mathbf{h}^T \boldsymbol{\alpha} \\ \text{subject to} \quad & H^T \boldsymbol{\alpha} = F\mathbf{x} + \boldsymbol{\beta} \\ & A^T \boldsymbol{\beta} \leq -\mathbf{c}. \end{aligned} \tag{5.14}$$

As $\Pi^{I}_{REG}(\mathbf{x}) = \Pi^{I,D}_{REG}(\mathbf{x})$, we can substitute $\Pi^{I,D}_{REG}(\mathbf{x})$ into the outer problem to obtain a final formulation:

$$\begin{aligned} \Pi_{REG} = \max_{\mathbf{x}\in\mathcal{X},\boldsymbol{\alpha}\leq\mathbf{0},\boldsymbol{\beta}} \quad & \mathbf{f}^T\mathbf{x} + \mathbf{h}^T\boldsymbol{\alpha} \\ \text{subject to} \quad & F\mathbf{x} - H^T\boldsymbol{\alpha} + \boldsymbol{\beta} = \mathbf{0} \\ & A^T\boldsymbol{\beta} \leq -\mathbf{c} \\ & \mathbf{g}_i^T\mathbf{x} + \boldsymbol{\xi}^T G_i \mathbf{x} \leq \boldsymbol{\xi}^T \mathbf{b}_i \quad \forall \boldsymbol{\xi} \in \Xi^P, \forall i. \end{aligned}$$

We have thus reformulated Π_{REG} with a polyhedral uncertainty set Ξ^P as a LO problem with one auxiliary decision variable per uncertainty set constraint and uncertain parameter, and one extra constraint per decision variable and uncertain parameter – a small increase in size over the equivalent RO problem, and in the same problem class (i.e, LO-and-LO if $\mathbf{x}$ is all continuous, and MILO-and-MILO if some $\mathbf{x}$ are integer).

For completeness we also consider the case of the ellipsoidal uncertainty set

$$\Xi^E = \{\boldsymbol{\xi} \mid \boldsymbol{\xi} = H\boldsymbol{\zeta} + \mathbf{h},\ \|\boldsymbol{\zeta}\|_2 \leq 1\}. \tag{5.15}$$

The inner problem in this case is, after eliminating $\boldsymbol{\xi}$,

$$\begin{aligned} \Pi^{I}_{REG}(\mathbf{x}) = \mathbf{x}^T F^T h + \min_{\mathbf{y}\geq\mathbf{0}, \|\boldsymbol{\zeta}\|_2 \leq 1} \quad & \mathbf{x}^T F^T H \boldsymbol{\zeta} - \mathbf{c}^T \mathbf{y} \\ \text{subject to} \quad & A\mathbf{y} = H\boldsymbol{\zeta} + h, \end{aligned} \tag{5.16}$$

the dual of which is the second-order cone (SOC) problem

$$\begin{aligned} \Pi^{I,D}_{REG}(\mathbf{x}) = \mathbf{x}^T F^T h + \max_{\boldsymbol{\beta}} \quad & \mathbf{h}^T \boldsymbol{\beta} - \|H^T F\mathbf{x} + H^T \boldsymbol{\beta}\|_2 \\ \text{subject to} \quad & A^T \boldsymbol{\beta} \leq -\mathbf{c}. \end{aligned} \tag{5.17}$$

As with Ξ^P, we can now substitute (5.17) into the outer problem to obtain a second-order cone (SOC) problem with one auxiliary variable per uncertainty set constraint, one extra linear constraint per decision variable, and one SOC constraint.

Finally, we note that if $F = 0$ then the optimal set of solutions to Π_{RO} and Π_{REG} will be identical. This is a consequence of Π^{I}_{REG} no longer being a function of $\mathbf{x}$, and we can thus interpret the inner optimization problem as finding the "best-case" $\boldsymbol{\xi}$ with respect to the offline problem, which is naturally independent of the decision made. One setting this would occur in is a two-stage ARO problem with affine adaptability, where the objective function contains only terms for the "here-and-now" decisions.

5.2.2 Reformulation of Competitive Ratio Problem Π_{CR}

A similar reformulation is possible for the competitive ratio problem Π_{CR}, utilizing the same duality-based approach. For ease of exposition, we first convert the objective function of Π_{CR} into a constraint using an epigraph formulation, i.e.,

$$\begin{aligned} \Pi_{CR} = \max_{\mathbf{x}\in\mathcal{X}, 0\leq z\leq 1} \quad & z \\ \text{subject to} \quad & z \leq \frac{\mathbf{f}^T\mathbf{x} + \boldsymbol{\xi}^T F\mathbf{x}}{\Pi(\boldsymbol{\xi})} \qquad \forall \boldsymbol{\xi} \in \Xi \\ & \mathbf{g}_i^T\mathbf{x} + \boldsymbol{\xi}^T G_i \mathbf{x} \leq \boldsymbol{\xi}^T \mathbf{b}_i \quad \forall \boldsymbol{\xi} \in \Xi, \forall i. \end{aligned} \tag{5.18}$$

If we rearrange the terms to eliminate the fraction in the epigraph constraint we obtain the inequality

$$0 \leq \mathbf{f}^T\mathbf{x} + \boldsymbol{\xi}^T F\mathbf{x} - z\Pi\left(\boldsymbol{\xi}\right) \quad \forall \boldsymbol{\xi} \in \Xi, \tag{5.19}$$

which is equivalent to

$$0 \leq \mathbf{f}^T\mathbf{x} + \min_{\boldsymbol{\xi}\in\Xi} \left\{\mathbf{x}^T F^T\boldsymbol{\xi} - z\Pi\left(\boldsymbol{\xi}\right)\right\}. \tag{5.20}$$

This yields an "inner" problem very similar to Π^I_{REG}, but with a coefficient z for the $\Pi\left(\boldsymbol{\xi}\right)$ term. As this coefficient is non-negative the convexity is unchanged and, as before, we can substitute the dual of the inner problem into the outer problem, resulting in a similar reformulation. The reformulation for a polyhedral uncertainty set Ξ^P is

$$\begin{aligned}
\Pi_{CR} = \max_{\mathbf{x}\in\mathcal{X}, 0\leq z\leq 1, \boldsymbol{\alpha}\leq\mathbf{0}, \boldsymbol{\beta}} \quad & z \\
\text{subject to} \quad & 0 \leq \mathbf{f}^T\mathbf{x} + \mathbf{h}^T\boldsymbol{\alpha} \\
& F\mathbf{x} - H^T\boldsymbol{\alpha} + \boldsymbol{\beta} = \mathbf{0} \\
& A^T\boldsymbol{\beta} \leq -\mathbf{c}z \\
& \mathbf{g}_i^T\mathbf{x} + \boldsymbol{\xi}^T G_i\mathbf{x} \leq \boldsymbol{\xi}^T\mathbf{b}_i \quad \forall \boldsymbol{\xi} \in \Xi, \forall i,
\end{aligned}$$

which is structurally very similar to the reformulation for Π_{REG}. An equivalent reformulation is available for Ξ^E, using the same approach.

5.2.3 Integer Variables in $\Pi\left(\boldsymbol{\xi}\right)$

A natural extension of the case of a concave LO $\Pi\left(\boldsymbol{\xi}\right)$ is to consider a MILO $\Pi\left(\boldsymbol{\xi}\right)$ with the same structure, but with some of the decision variables restricted to integer values. If the extreme points of the feasible region in a MILO $\Pi\left(\boldsymbol{\xi}\right)$ all satisfy the integer constraints automatically, then we can effectively ignore the integer constraints and proceed as we did before. However, if this is not the case, then we must modify

our approach as we can no longer use duality in the same fashion.

We note that substitution and rearrangement of terms in (5.12) did not explicitly rely on the concavity or continuous nature of $\Pi(\boldsymbol{\xi})$. Thus, we are able to perform the same rearrangement even for a MILO $\Pi(\boldsymbol{\xi})$, to obtain a MILO inner problem for $\Xi = \Xi^P$ and a mixed-integer SOC (MISOC) inner problem for $\Xi = \Xi^E$. As we cannot obtain an equivalent dual formulation for this problem, we will instead solve the primal inner problem iteratively for different values of $\mathbf{x}$ to approximate the outer problem's objective function with a set of hyperplanes – a *cutting plane* method.

The algorithm proceeds as follows. Let $\hat{\Xi}$ be a subset of $\hat{\boldsymbol{\xi}} \in \Xi$ for which we have previously calculated $\Pi\left(\hat{\boldsymbol{\xi}}\right)$. We can then formulate and solve the following relaxation that provides an *upper bound* on the value of Π_{REG}:

$$\begin{aligned} \Pi^{UB}_{REG}\left(\hat{\Xi}\right) = \max_{\mathbf{x}\in\mathcal{X},z} \quad & z \\ \text{subject to} \quad & z \leq \mathbf{f}^T\mathbf{x} + \hat{\boldsymbol{\xi}}^T F\mathbf{x} - \Pi\left(\hat{\boldsymbol{\xi}}\right) \quad \forall \hat{\boldsymbol{\xi}} \in \hat{\Xi} \\ & \mathbf{g}_i^T\mathbf{x} + \boldsymbol{\xi}^T G_i\mathbf{x} \leq \boldsymbol{\xi}^T\mathbf{b} \quad \forall \boldsymbol{\xi} \in \Xi, \forall i \end{aligned} \tag{5.21}$$

Let the solution to this relaxation be $\mathbf{x}^*$. As $\mathbf{x}^*$ is a feasible solution to Π_{REG}, the objective function value of the inner problem $\Pi^I_{REG}(\mathbf{x}^*)$ represents a *lower bound* on the true objective function value of Π_{REG} at $\mathbf{x}^*$. We now take the optimal solution to the inner problem, $\boldsymbol{\xi}^*$, and add it to the set $\hat{\Xi}$. We can repeat this process of solving the outer relaxation and the lower bound problem until the "gap" between the lower bound and upper bound is zero, or sufficiently close enough that further effort is not justified. We can use this approach even when the outer problem has integer decision variables by using the *lazy constraint* feature supported by many MILO solvers – see Chapter 2 for a discussion of the use of this feature for solving MILO RO problems efficiently.

5.3 $\Pi(\boldsymbol{\xi})$ is a Convex Function of $\boldsymbol{\xi}$

In this section, we consider offline problems $\Pi(\boldsymbol{\xi})$ of the form

$$\begin{aligned} \Pi(\boldsymbol{\xi}) = \max_{\mathbf{x} \geq \mathbf{0}} \quad & \boldsymbol{\xi}^T \mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} = \mathbf{b}, \end{aligned} \tag{5.22}$$

which is a linear optimization problem with a parametric objective function vector (in contrast to the parameter right-hand-side vector in Section 5.2). This problem is equivalent to maximizing over a finite number of linear functions, and thus $\Pi(\boldsymbol{\xi})$ is a piecewise linear *convex* function of $\boldsymbol{\xi}$.

As it did in Section 5.2, $\Pi(\boldsymbol{\xi})$ appears in the objective function of $\Pi^I_{REG}(\mathbf{x})$ as $-\Pi(\boldsymbol{\xi})$, however in this case it is a *concave* term. As we are minimizing, the resulting optimization problem is nonconvex and thus does not have an equivalent dual reformulation or a simple solution method. Despite this, we can still substitute $\Pi(\boldsymbol{\xi})$ into (5.11) and rearrange terms as before:

$$\Pi^I_{REG}(\mathbf{x}) = \min_{\boldsymbol{\xi} \in \Xi} \mathbf{x}^T F^T \boldsymbol{\xi} - \max_{A\mathbf{y}=\mathbf{b}, \mathbf{y} \geq \mathbf{0}} \boldsymbol{\xi}^T \mathbf{y} \tag{5.23a}$$

$$= \min_{\boldsymbol{\xi} \in \Xi} \mathbf{x}^T F^T \boldsymbol{\xi} + \min_{A\mathbf{y}=\mathbf{b}, \mathbf{y} \geq \mathbf{0}} -\boldsymbol{\xi}^T \mathbf{y} \tag{5.23b}$$

$$= \min_{\boldsymbol{\xi} \in \Xi, A\mathbf{y}=\mathbf{b}, \mathbf{y} \geq \mathbf{0}} \mathbf{x}^T F^T \boldsymbol{\xi} - \boldsymbol{\xi}^T \mathbf{y}. \tag{5.23c}$$

Problem (5.23c) is a *biconvex* (and *bilinear*) optimization problem: if we fix $\mathbf{y}$, the objective function becomes linear and the problem becomes convex, and likewise if we fix $\boldsymbol{\xi}$. This problem has been studied extensively, and there exist multiple approaches for solving it of varying computational practicality – see, e.g, Gorski et al. [2007]. In particular, branch-and-bound algorithms exist that guarantee a globally optimal solution to problems like Π^I_{REG}. These algorithms would be sufficient to solve the outer problem to optimality when combined with the cutting plane approach described in Section 5.2.3, although we would not expect to be able to solve instances as large as we could if $\Pi(\boldsymbol{\xi})$ was concave. Alternatively, we could approximately solve

the inner problem by alternatively fixing one set of variables and minimizing the other. This does not guarantee an optimal solution, but maybe be sufficient to find good solutions if applied multiple times for different random starting solutions. Finally, we propose a third method that reformulates Π^I_{REG} as a MILO problem by exploiting the structure of the popular "budget" polyhedral uncertainty set, introduced by Bertsimas and Sim [2004]. This allows us to use the cutting plane method of Section 5.2.3 and utilizes the power of high-performance MILO solvers. It extends the method of Mausser and Laguna [1999a], and in terms of computational practicality, benefits from nearly two decades of improvements in MILO solvers and computers.

5.3.1 Reformulating Π^I_{REG} as a MILO problem

To perform our reformulation we rely on the following theoretical result, which we reproduce in the notation of the survey by Gorski et al. [2007], and appears to have been first shown by Geng and Huang [2000]:

Theorem 3. *Let $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ be biconvex and let $S \subset \mathbb{R}^n, T \subset \mathbb{R}^m$ be polytopes with vertex sets S^* and T^*, respectively. Then*

$$\max_{(\mathbf{x},\mathbf{y})\in S\times T} f(\mathbf{x},\mathbf{y}) = \max_{(\mathbf{x},\mathbf{y})\in S^*\times T^*} f(\mathbf{x},\mathbf{y}). \tag{5.24}$$

In other words, when optimizing a biconvex optimization problem such as $\Pi^I_{REG}(\mathbf{x})$, it is sufficient to consider solutions only on the extreme points of two polyhedra (which are Ξ and the feasible set of the offline problem).

In general the polyhedra we are interested in will have a large number of extreme points, to the extent that enumerating them is not practical. However, there is one case of practical interest for which we can use MILO modeling techniques to implicitly do so. First, we will assume that all our offline problem decision variables have known and finite upper bounds $\mathbf{u}$ such that $\mathbf{0} \leq \mathbf{y} \leq \mathbf{u}$. Second, we assume our uncertainty set is the "budget" uncertainty set

$$\Xi = \{\boldsymbol{\xi} \in \mathbb{R}^n \mid \xi_i = \mu_i + \sigma_i\gamma_i, \|\boldsymbol{\gamma}\|_\infty \leq 1, \|\boldsymbol{\gamma}\|_1 \leq \Gamma\}, \tag{5.25}$$

where $\Gamma \in \{1, \ldots, n\}$. This includes, as a special case, the "box" uncertainty set ($\Gamma = n$). The budget uncertainty set consists of n uncertain parameters, of which up to Γ of them may at any time deviate by up to σ_i from their "nominal" value μ_i.

If we substitute this uncertainty set into $\Pi^I_{REG}(\mathbf{x})$ and eliminate $\boldsymbol{\xi}$ in favor of $\boldsymbol{\gamma}$ we obtain the optimization problem

$$\begin{aligned} \Pi^I_{REG}(\mathbf{x}) = \min_{\mathbf{0} \leq \mathbf{y} \leq \mathbf{u}, \boldsymbol{\gamma}} \quad & (\boldsymbol{\mu} + \Sigma\boldsymbol{\gamma})^T (\mathbf{y} + F\mathbf{x}) \\ \text{subject to} \quad & \|\boldsymbol{\gamma}\|_\infty \leq 1 \\ & \|\boldsymbol{\gamma}\|_1 \leq \Gamma \\ & A\mathbf{y} = \mathbf{b}, \end{aligned} \tag{5.26}$$

where $\Sigma = diag(\boldsymbol{\sigma})$. The "problematic" bilinear terms, which were $\xi_i \cdot y_i$, have now been replaced by terms $\gamma_i \cdot y_i$. We note that, by Theorem 1, there exists an optimal solution to $\Pi^I_{REG}(\mathbf{x})$ on the extreme points of Ξ, which means that there exists an optimal solution such that a non-zero number of γ_i have an absolute value of 1 (either -1 or +1).

Our reformulation begins by replacing each γ_i with two auxiliary binary decision variables $\gamma_i^+, \gamma_i^- \in \{0, 1\}$, with the substitution

$$\gamma_i = \gamma_i^+ - \gamma_i^-. \tag{5.27}$$

We require auxiliary constraints

$$\gamma_i^+ + \gamma_i^- \leq 1 \quad \forall i, \tag{5.28}$$

which altogether enforces that $\gamma_i \in \{-1, 0, +1\}$. After performing this substitution, the problematic terms are now $\gamma_i^+ \cdot y_i$ and $\gamma_i^- \cdot y_i$, which are products of a continuous and a binary decision variable. We replace each of these terms with new auxiliary decision variables p_i^+ and p_i^-, respectively. These auxiliary variables are linked to the

original decision variables by the constraints:

$$\begin{aligned} & p_i^\bullet \geq 0 \\ & p_i^\bullet \geq y_i - u_i\left(1-\gamma_i^\bullet\right) \\ & p_i^\bullet \leq u_i \gamma_i^\bullet \\ & p_i^\bullet \leq y_i, \end{aligned} \tag{5.29}$$

where these constraints should be applied for both $\bullet = +$ and $\bullet = -$. We can verify that this is an exact reformulation of the original bilinear term by enumerating the feasible possibilities, of which there are only two. In the first, $\gamma^\bullet = 0$, and the constraints reduce to $\max\{0, y_i - u_i\} \leq p_i^\bullet \leq \min\{0, y_i\}$, or simply $p_i^\bullet = 0$. In the second, $\gamma^\bullet = 1$, and the constraints reduce to $\max\{0, y_i\} \leq p_i^\bullet \leq \min\{u_i, y_i\}$, or simply $p_i^\bullet = y_i$.

We can now state the final optimization MILO inner problem:

$$\begin{aligned} \Pi_{REG}^I(\mathbf{x}) = \min_{\mathbf{0}\leq\mathbf{y}\leq\mathbf{u},\boldsymbol{\gamma}^\bullet\in\{0,1\},\mathbf{p}^\bullet} \quad & \left(\boldsymbol{\mu} + \Sigma\left(\boldsymbol{\gamma}^+ - \boldsymbol{\gamma}^-\right)\right)^T F\mathbf{x} + \boldsymbol{\mu}^T\mathbf{y} + \sigma^T\left(\mathbf{p}_i^+ - \mathbf{p}_i^-\right) \\ \text{subject to} \quad & A\mathbf{y} = \mathbf{b} \\ & \|\boldsymbol{\gamma}^+\|_1 + \|\boldsymbol{\gamma}^-\|_1 \leq \Gamma, \quad \boldsymbol{\gamma}^+ + \boldsymbol{\gamma}^- \leq \mathbf{1} \\ & p_i^\bullet \leq \min\{u_i\gamma_i^\bullet, y_i\} \quad \forall i,\ \bullet \in \{+,-\}. \\ & p_i^\bullet \geq \max\{0, y_i - u_i\left(1-\gamma_i^\bullet\right)\} \quad \forall i,\ \bullet \in \{+,-\}. \end{aligned} \tag{5.30}$$

The term $\left(\boldsymbol{\mu} + \Sigma\left(\boldsymbol{\gamma}^+ - \boldsymbol{\gamma}^-\right)\right)^T F\mathbf{x}$ in the objective function is the transformation of $\boldsymbol{\xi}^T F\mathbf{x}$, while the remainder is the linearization of $\Pi(\boldsymbol{\xi})$. The objective function for $\Pi_{CR}^I(\mathbf{x})$ differs only in that we premultiply the terms corresponding to $\Pi(\boldsymbol{\xi})$ by the value of the auxiliary epigraph variable z from the outer problem, i.e., $z\boldsymbol{\mu}^T\mathbf{y}$ and $z\sigma^T\left(\mathbf{p}_i^+ - \mathbf{p}_i^-\right)$. This problem is, of course, larger than the original inner problem: it has four auxiliary variables (two binary) and ten extra constraints for every uncertain parameter. However, it can be solved with any MILO solver for which many excellent options, both commercial and open-source, are available. A natural question is whether this reformulation results in a computationally tractable method to solve the

original problem. To address this, we apply this methodology to a minimum-cost flow problem in Section 5.5.1 across a variety of problem sizes.

5.4 Combining Objective Functions – Pareto-efficient Solutions

Having established methods to solve the regret and competitive ratio problems, a natural next question is when and where to use them. In some settings the choice may be dictated by the risk preferences of the decision maker, e.g., "the absolute cost of a project may not exceed a certain limit", or "the portfolio can perform no more than a certain factor worse than the best portfolio in adverse conditions". However, there may not always be such clear guidance, and a decision maker may wish to explore the spectrum of possibilities across these objective functions. Moreover, a solution that takes both absolute and regret objective functions into consideration may be more desirable than the optimal solution for the objective functions separately. In this section, we formalize the notion and methodology to achieve these "blended" solutions.

Recall that the competitive ratio objective function is

$$f_{CR}(\mathbf{x}) = \min_{\boldsymbol{\xi} \in \Xi} \frac{f(\mathbf{x}, \boldsymbol{\xi})}{\Pi(\boldsymbol{\xi})}, \tag{5.31}$$

where $f_{CR}(\mathbf{x}) \leq 1$ for feasible $\mathbf{x}$ and is a dimensionless quantity (i.e., the numerator and denominator have the same units that will then cancel out). The absolute robust objective function is

$$f_{RO}(\mathbf{x}) = \min_{\boldsymbol{\xi} \in \Xi} f(\mathbf{x}, \boldsymbol{\xi}), \tag{5.32}$$

which is on a different magnitude scale to $f_{CR}(\mathbf{x})$, and may have dimensions (e.g., a currency, area, mass). To explore solutions that have aspects of both we need to combine the two objective functions, but mixing units and scales can lead to models that are hard to interpret and reason about. To address this, we define a new

"normalized" robust (NR) objective function :

$$f_{NR}(\mathbf{x}) = \min_{\boldsymbol{\xi}\in\Xi} \frac{f(\mathbf{x},\boldsymbol{\xi})}{\Pi_{RO}}, \tag{5.33}$$

where $f_{NR}(\mathbf{x}) \leq 1$ for feasible $\mathbf{x}$ and is a dimensionless quantity. Note that $f_{NR}(\mathbf{x}) = 1$ is obtained for any optimal solution to Π_{RO}, and is always obtainable by definition. Any sub-optimal solution $\hat{\mathbf{x}}$ to Π_{RO} will have a corresponding $\hat{\boldsymbol{\xi}}$ such that $f\left(\hat{\mathbf{x}},\hat{\boldsymbol{\xi}}\right) < \Pi_{RO}$. In contrast, there may be no solution for which $f_{CR}(\mathbf{x}) = 1$.

With these two unit-less objective functions of comparable scales defined, we can now consider combining them. We define the *Pareto-efficient robust* (PR) optimization problem to be

$$\begin{aligned} \Pi_{PR}(\lambda) = \max_{\mathbf{x}\in\mathcal{X}} \quad & \lambda \cdot f_{CR}(\mathbf{x}) + (1-\lambda)\cdot f_{NR}(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{g}(\mathbf{x},\boldsymbol{\xi}) \leq \mathbf{0} \quad \forall \boldsymbol{\xi} \in \Xi, \end{aligned} \tag{5.34}$$

the solutions of which form an *efficient frontier* of Pareto-efficient solutions, such that any improvement in one objective function comes at the expense of the other. For $\lambda = 0$ the problem is equivalent to Π_{RO}, and for $\lambda = 1$ it is equivalent to Π_{CR}. The difficulty of solving this problem is primarily driven by the competitive ratio component of the objective function, and can be solved in the same way using the same methods. This is demonstrated by considering the equivalent epigraph formulation of the problem, i.e.,

$$\begin{aligned} \Pi_{PR}(\lambda) = \max_{\mathbf{x}\in\mathcal{X}, z_{CR}, z_{NR}} \quad & \lambda z_{CR} + (1-\lambda) z_{NR}(\mathbf{x}) \\ \text{subject to} \quad & \Pi(\boldsymbol{\xi}) \cdot z_{CR} \leq f(\mathbf{x},\boldsymbol{\xi}) \quad \forall \boldsymbol{\xi} \in \Xi \\ & \Pi_{RO} \cdot z_{NR} \leq f(\mathbf{x},\boldsymbol{\xi}) \quad \forall \boldsymbol{\xi} \in \Xi \\ & \mathbf{g}(\mathbf{x},\boldsymbol{\xi}) \leq \mathbf{0} \quad \forall \boldsymbol{\xi} \in \Xi, \end{aligned} \tag{5.35}$$

where the first constraint set is the same as in Π_{CR}, and the second is the same as in the relatively tractable Π_{RO}.

The benefits of considering Pareto-efficient solutions depend strongly on the par-

ticular problem. There are not necessarily any intermediate solutions that differ substantially from the solutions for $\lambda = 0$ and $\lambda = 1$ – problems with integer variables in particular may not have much of a spectrum of possibilities, as small changes may correspond to large movements in the objective function value. The notion that there may be many equivalent robust solutions and we want to pick the "best" by a second metric was explored previously by Iancu and Trichakis [2013]. They proposed finding the worst-case absolute objective function value Π_{RO}, then selecting amongst the solutions to the RO problem that are no worse than Π_{RO} in the worst case (i.e., $f_{NR}(\mathbf{x}) = 1$) by optimizing for a particular, manually selected $\hat{\boldsymbol{\xi}} \in \Xi$. Our approach differs in that we do not pick any particular scenario, but instead provide an alternative but related objective function and thus the decision maker need only consider a single scalar parameter λ. For example, we could place a small weight on the CR objective function, i.e., $\lambda = \varepsilon$, and find an absolute robust solution that also attempts to drive up the CR objective function as high as possible without compromising the worst-case absolute performance.

5.4.1 Worked Example: Portfolio Optimization

To explore the two alternatives to "Pareto robustness", consider a simple portfolio optimization problem with n assets and uncertain returns $\mathbf{r}$. If $\mathbf{x}$ is the fraction of funds allocated to each asset, then the feasible set is $\mathcal{X} = \left\{\mathbf{x} \mid \mathbf{x} \geq 0,\ \mathbf{1}^T\mathbf{x} = 1\right\}$. We'll make the simplifying assumption that the returns on the assets are independent and identically distributed, and define the uncertainty set to be the box set

$$\mathcal{U} = \{\mathbf{r} \mid 1 \leq r_i \leq 2 \quad \forall i \in \{1, \ldots, n\}\}. \tag{5.36}$$

The offline optimization problem is then

$$\Pi(\mathbf{r}) = \max_{\mathbf{x} \in \mathcal{X}} \ \mathbf{r}^T\mathbf{x} = \max_i \ r_i. \tag{5.37}$$

Absolute robust: The absolute robust optimization problem is

$$\Pi_{RO} = \max_{\mathbf{x}\in\mathcal{X}} \min_{\mathbf{r}\in\mathcal{U}} \mathbf{r}^T\mathbf{x}, \tag{5.38}$$

which has multiple solutions. In fact, any $\hat{\mathbf{x}} \in \mathcal{X}$ is a robust optimal solution, as the worst-case realizations of the uncertain parameters are independent of the portfolio selected, i.e., $\hat{r}_i = 1\ \forall i$, with $\Pi_{RO} = 1$. While any solution is optimal, simplex-based solvers will return a solution that is one of the n extreme point of $\mathcal{X}$. We will index these solutions by k, i.e., $\mathbf{x}^{[k]}$, and they are all of the general form $x^{[k]}_{i=k} = 1$, $x^{[k]}_{i\neq k} = 0$. We can assess solutions of this form by the relative robust objective functions. Under the competitive ratio objective, the worst case realization for the solution $\mathbf{x}^{[k]}$ is the scenario $\hat{r}_{i=k} = 1$, $\hat{r}_{i\neq k} = 2$, giving an objective function value of $\frac{1}{2}$. Under the robust regret objective, the worst case realization is the same, and gives an objective function value of -1.

Relative robust: The competitive ratio optimization problem is

$$\Pi_{CR} = \max_{\mathbf{x}\in\mathcal{X}} \min_{\mathbf{r}\in\mathcal{U}} \frac{\mathbf{r}^T\mathbf{x}}{\max_i r_i}, \tag{5.39}$$

and unlike Π_{RO} it has a unique optimal solution $\hat{x}_i = \frac{1}{n}$, with value $\Pi_{CR} = \frac{n+1}{2n}$ corresponding to a worst-case realization of $\hat{r}_{i=1} = 2$, $\hat{r}_{i\neq 1} = 1$. The same solution is obtained for the regret objective function, and $\Pi_{REG} = \frac{n+1}{n} - 2$. Finally, under the absolute robust objective function, we obtain a value of 1 (as this is also a solution to Π_{RO}). Thus the solution to Π_{CR} (or Π_{REG}) is, in this case, a better solution than the solutions $\mathbf{x}^{[k]}$ described above as it is superior by relative measures, and equivalent on absolute measures.

Pareto solutions: Given that we have a multiple solutions to Π_{RO}, we are in a setting that is considered by Iancu and Trichakis [2013]. In that paper, a solution is a "Pareto robustly optimal" (PRO) solution if there is no other solution that is as-good-as-or-better across all scenarios, and there doesn't exists a scenario for which another solution is strictly better. To identify one of these PRO solutions, we can optimize for a particular $\hat{\mathbf{r}}$ amongst all robust solutions. As we established above, all

feasible solutions are robust solutions for this problem, but the set of PRO solutions is the set of solutions $\mathbf{x}^{[k]}$. To see why, consider any solution *not* of the form $\mathbf{x}^{[k]}$, which will have some index j for which $0 < x_j < 1$. If we consider any scenario $\hat{\mathbf{r}}$ (other than $\hat{\mathbf{r}} = \mathbf{1}$) then this solution is dominated by one of the solutions $\mathbf{x}^{[k]}$ that allocates all resources to the asset with the highest return in $\hat{\mathbf{r}}$.

In our proposal, instead of optimizing for any given scenario, we place a very small weight on the z_{CR} component of the objective function, i.e., $\Pi_{PR}(\lambda = \varepsilon)$. This is essentially equivalent to selecting a solution, amongst the RO solutions, with the best competitive ratio. For any value of $\lambda > 0$, we recover the CR solution $\hat{x}_i = \frac{1}{n}$. Thus not only have we avoided the need to determine a particular scenario for which we should optimize for, but we have also found a solution that is better than any PRO solution (as measured by the three objective functions). Finally, we suggest that this solution is perhaps the most intuitive, as diversification is a known and successful risk management strategy in portfolio design.

5.5 Computations and Insights

To develop an understanding for the situations that benefit from considering relative robustness, and to understand the computational difficulty of doing so, we have analyzed three cases that cover many different situations:

1. A minimum-cost flow problem (Section 5.5.1), with an uncertain objective function, continuous variables, and no recourse (single stage).

2. An inventory control problem (Section 5.5.2), a multistage problem (modeled with affine adaptability) with uncertain right-hand-side parameters and continuous variables.

3. A facility location problem (Section 5.5.3), with two time stages, an uncertain right-hand-side, and both binary and continuous variables.

All of the optimization problems were modeled with JuMP [Dunning et al., 2015] and solved with Gurobi 6.5.0 [Gurobi Optimization Inc., 2016].

5.5.1 Minimum-cost Flow

Consider the problem of sending a unit of flow from node 1 to node n across a directed graph $G = (V, E)$ at minimum cost, $V = \{1, \ldots, n\}$. All nodes, except nodes 1 and n, must conserve flow ("flow in equals flow out"). Each edge $(i, j) \in E$ has capacity $C_{i,j}$ and an uncertain per-unit flow cost $c_{i,j}$. The offline problem $\Pi(\mathbf{c})$ can be expressed as the LO problem

$$\Pi(\mathbf{c}) = \min_{\mathbf{x}} \quad \sum_{(i,j)\in E} c_{i,j} \cdot x_{i,j} \tag{5.40a}$$

$$\text{subject to} \quad \sum_{(i,n)\in E} x_{i,n} = 1 \tag{5.40b}$$

$$\sum_{(i,j)\in E} x_{i,j} = \sum_{(j,k)\in E} x_{j,k} \qquad \forall j \in V \setminus \{1, n\} \tag{5.40c}$$

$$0 \leq x_{i,j} \leq C_{i,j} \qquad \forall (i, j) \in E, \tag{5.40d}$$

where (5.40b) requires exactly one unit of flow to arrive at node n and (5.40c) enforces flow conservation everywhere except nodes 1 and n (allowing flow to be "injected" at node 1).

We performed computational experiments comparing the RO and CR versions of the minimum-cost flow problem. The RO, CR and offline problems all share the same feasible region, $\mathcal{X}$, as the uncertain parameters appear only in the objective function. We used a polyhedral "budget" uncertainty set $\mathcal{U}$ throughout this experiment:

$$\mathcal{U} = \{\mathbf{c} \mid c_{i,j} = \mu_{i,j} + \sigma_{i,j}\gamma_{i,j},\ \|\boldsymbol{\gamma}\|_\infty \leq 1,\ \|\boldsymbol{\gamma}\|_1 \leq \Gamma\}, \tag{5.41}$$

where $\Gamma \in \{1, \ldots, |E|\}$. The RO version of the problem is simply

$$\Pi_{RO} = \min_{x\in\mathcal{X}} \max_{c\in\mathcal{U}} \sum_{(i,j)\in E} c_{i,j} \cdot x_{i,j}, \tag{5.42}$$

and the CR version is

$$\Pi_{CR} = \min_{\mathbf{x}\in\mathcal{X}} \max_{c\in\mathcal{U}} \frac{\sum_{(i,j)\in E} c_{i,j} \cdot x_{i,j}}{\Pi(\mathbf{c})}. \tag{5.43}$$

As the "outer" problem is minimizing, and the uncertainty is in the objective function, we are in the setting considered in Section 5.3. In particular, we use the MILO reformulation of the "inner" problem described in Section 5.3.1, and couple this with the cutting plane method. Cutting planes were added until no epigraph constraint with an absolute violation greater than 10^{-3} could be found. We compared the solutions and computational difficulty of both the RO and CR formulations across two different graph structures.

Layered graph

The first graph structure is a "layered" pattern, with all edge capacities set to one. We have L layers, each with M nodes – the source node 1 is connected to each node in layer $l = 1$, and each node in layer $l = L$ is connected to the sink node $i = n$. Each node in each layer is connected to each node in the subsequent layer. The nominal cost μ and deviation σ for a unit of flow from node i in layer l to all nodes in layer $l + 1$ is $\mu = M + i - 1$, $\sigma = M - i + 1$ if l is even, and $\mu = 2M - i$, $\sigma = i$ if l is odd. While somewhat complex, this pattern has the property that lower nominal cost edges have higher deviation, and that there is not a simple low-deviation or low-nominal cost flow through the network. An example instance for $L = 3, M = 3$ is shown in Figure 5-1 with the cost information displayed.

To compare solutions, we fixed $L = 3, M = 3$, and set $\Gamma = L \cdot M$ (where the maximum value would be $M + (L - 1)M^2$). The RO solution and CR solutions are displayed in Figure 5-2. Both solutions "diversify" their flows: the deterministic solution will only use four edges out of 24, while the CR solution uses 16 and the RO solution uses 21. In particular, the CR solution appears to avoid edges with $(\mu, \sigma) = (5, 1)$ (using only two) and avoiding edges that lead to those edges, while the RO solution uses four of the seven edges with those costs. To understand the

impact this has on solution quality, we simulated cost vectors $\mathbf{c}$ and calculated the empirical distribution of the total cost of the respective solutions. Each $c_{i,j}$ was drawn independently from $N(\mu_{i,j}, \sigma_{i,j})$, and then clipped at 0 if the sample was negative. In total 10,000 samples of $\mathbf{c}$ were drawn, with $\Pi(\mathbf{c})$ and the total cost for each of the two solutions calculated for each sample. The resulting distribution of costs is plotted in Figure 5-3.

When considering the absolute total costs, we see that the RO solution does a better job of protecting against the very worst case scenarios (16.8 for RO versus 17.1 for CR at the 99^{th} percentile), but the two are equal by the 93^{rd} percentile. This worst-case protection comes at the detriment of the median (13.2 for CR versus 15.6 for RO) and even for "easier" scenarios (11.3 for CR versus 12.2 for RO at the 25^{th} percentile). If we consider relative total costs, we see that the CR solution is superior to the RO solution at effectively all percentiles, including the 99^{th} (3.4 versus 3.6), the median (1.7 versus 1.9), and the 25^{th} (1.5 versus 1.6). We conclude from this experiment that while the CR solution is not a strict improvement over the RO solution, unless we are concerned with the very worst-case scenarios under an absolute metric, we should prefer the competitive ratio solution for this graph layout.

Dense graph

The second graph structure is a "dense" graph, with all edge capacities set to $\frac{1}{2}$. The graph has n nodes that are connected to all other $n-1$ nodes. The costs are non-symmetric and are randomized – the nominal costs $\mu_{i,j}$ are drawn from $Uniform(0, 10)$, and the deviations $\sigma_{i,j}$ are drawn from $Uniform(0, \mu_{i,j})$. We considered the case of $n = 10$ with $\Gamma = 10$ (out of a possible maximum of 90), and compared the RO and CR solutions for one particular sample of the cost parameters, with the two solutions displayed in Figure 5-4. The solutions are qualitatively very different for the dense graph, in contrast to the layered graph. The robust solution picks two "paths", i.e., four edges, between the source and sink and assigns all flow there. The competitive ratio uses 16 edges in total in an apparent effort to "diversify" the flow over the graph.

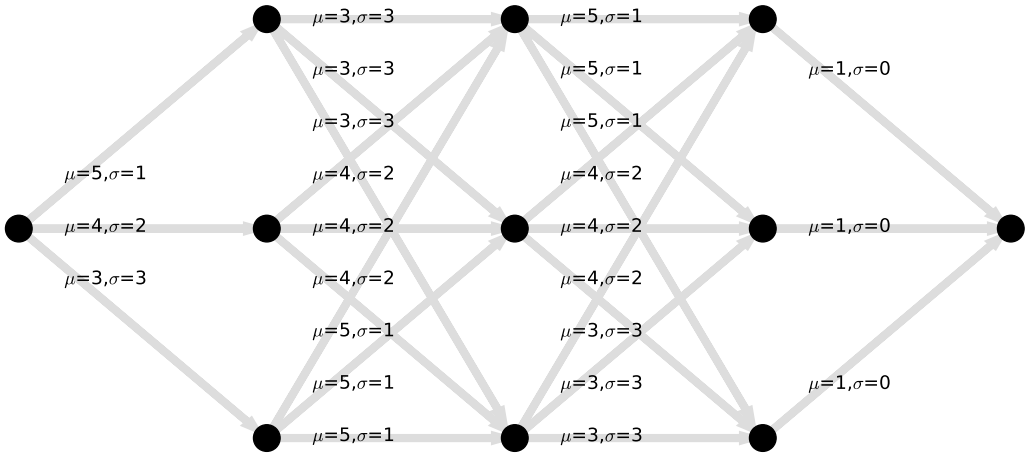

Figure 5-1: An example of a "layered" graph ($L = 3, M = 3$) used for the experiments in Section 5.5.1, with the nominal cost μ and deviation σ for each edge. All edges have capacity one, and one unit of flow must flow from the left-most node to the right-most node.

We obtained an empirical distribution of the costs using the same model as in Section 5.5.1, with the results displayed in Figure 5-5. For the absolute total cost, the RO solution is significantly better at the 99^{th} percentile (8.3 versus 9.1 for CR), but also at the median (5.5 versus 5.8). For the relative total cost, we do obtain protection from the CR solution: it is significantly better at the 99^{th} percentile with 5.6 versus 7.1 for RO. However, the two solutions are essentially the same at the 82^{nd} percentile and the RO solution is better at the median (1.8 versus 2.0). The conclusions that we draw are roughly the opposite from what was obtained from the layered graph: that the RO solution would in general be preferable unless the worst-case relative costs are critical.

Computational tractability

To estimate how tractable the competitive ratio formulation is relative to the robust formulation, we solved each formulation for 30 different instantiations of the dense graph layout for $n \in \{10, 20, 30, 40, 50\}$, with the results plotted in Figure 5-6. We also solved the deterministic problem, which is a very easy LO problem at these sizes. The

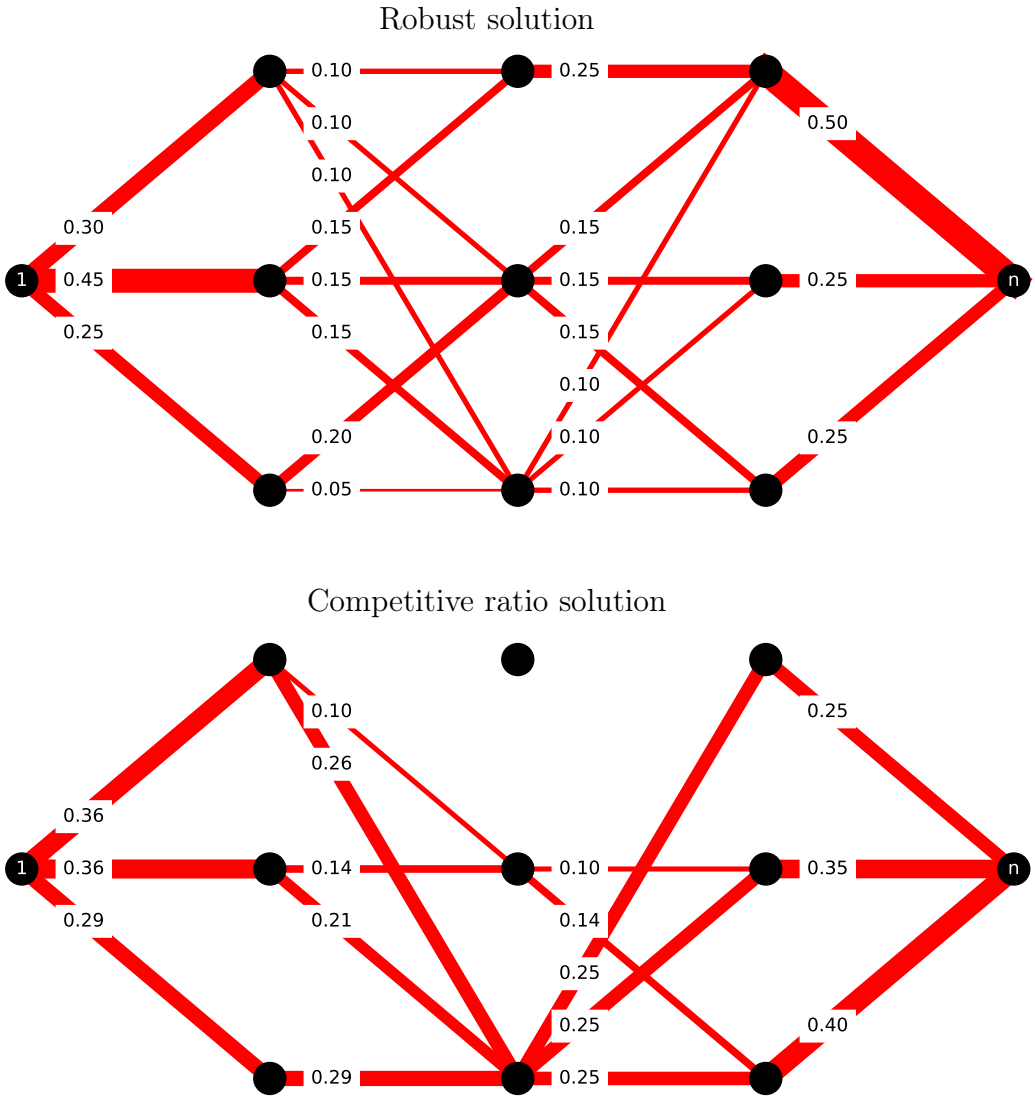

Figure 5-2: Optimal flows for a "layered" graph ($L = 3, M = 3, \Gamma = 9$) used for the experiments in Section 5.5.1, with the top image showing the robust solution, and the bottom image showing the competitive ratio solution. The thickness of the line and the text labels indicate the amount of flow on each edge - the absence of a line indicates zero flow.

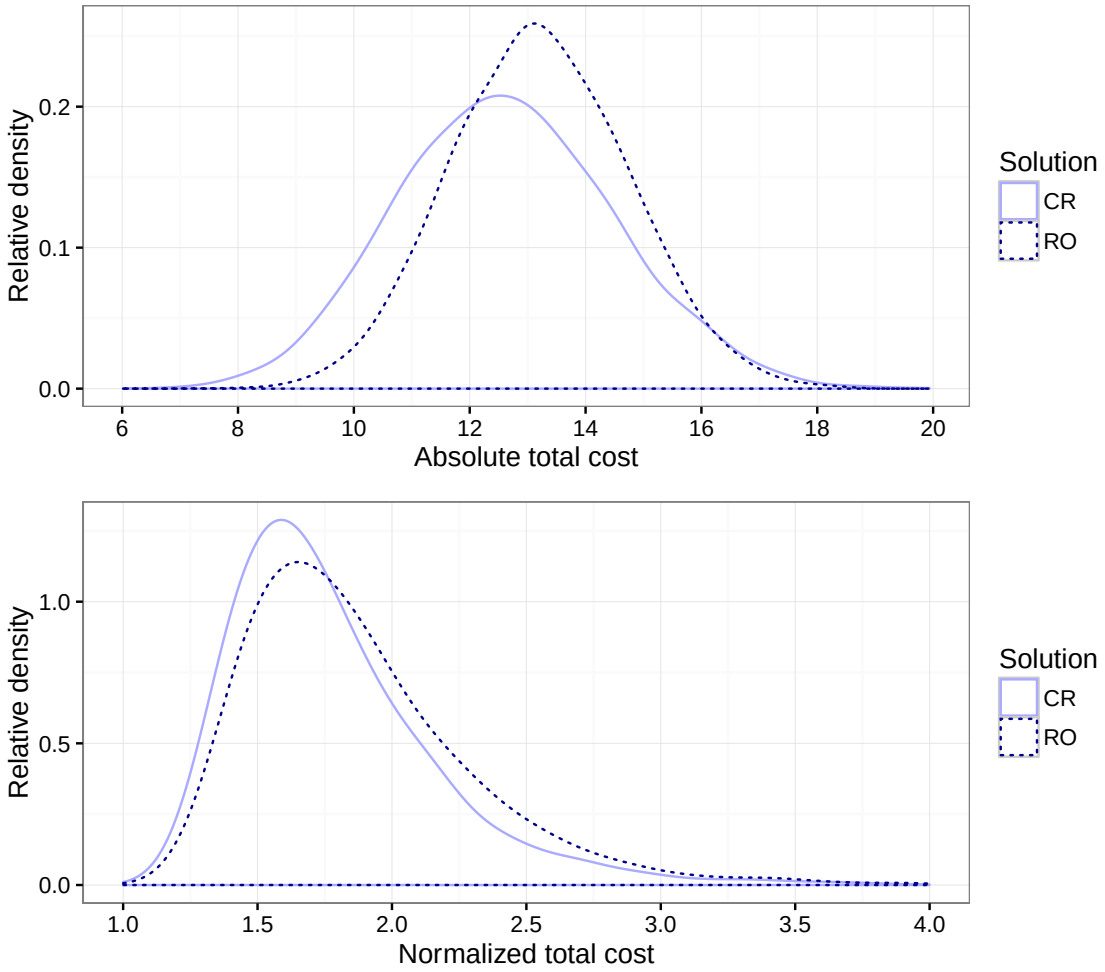

Figure 5-3: Empirical distribution of absolute total costs (top) and total cost normalized by the clairvoyant optimal total cost (bottom) obtained by simulating costs for the CR and RO solutions to a "layered" graph ($L = 3, M = 3, \Gamma = 9$) described in Section 5.5.1.

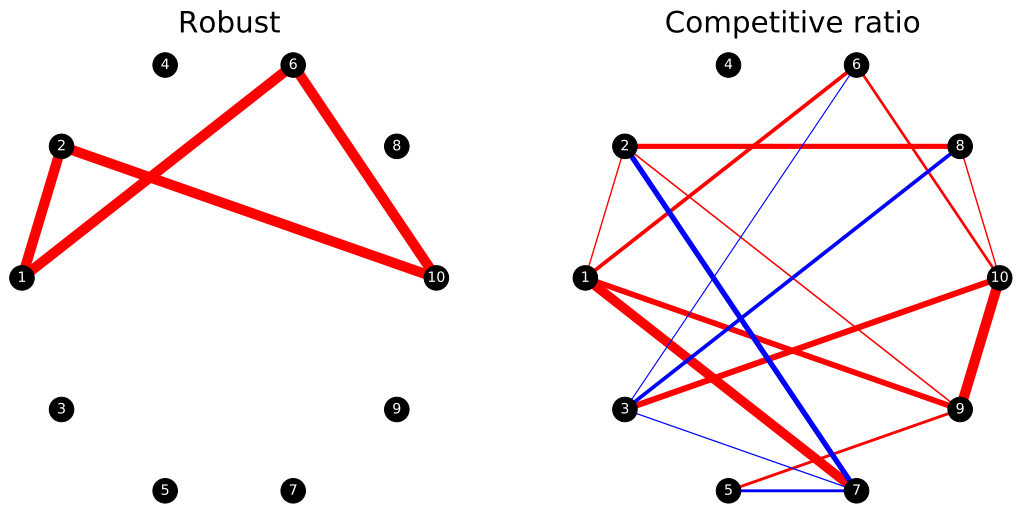

Figure 5-4: Optimal flows for a "dense" graph ($n = 10$) used for the experiments in Section 5.5.1, with the left image showing the robust solution, and the right image showing the competitive ratio solution. The thickness of the line indicate the amount of flow on each edge - the absence of a line indicates zero flow. Red flows are for edges with flow going from a lower-indexed node to a higher-indexed node, and blue flows for the opposite.

results suggest that the robust formulation is approximately 50 to 100 times slower than the deterministic formulation, and that the CR formulation is approximately 10 to 20 times slower than the robust formulation.

The median time to solve an instance of size $n = 50$ with the CR formulation was approximately 3.6 seconds, versus approximately 0.3 seconds for the RO formulation. We note that we could possibly improve on these times by using a more problem-specific method for generating good cuts, or re-using cuts if solving a series of similar problems.

5.5.2 Multistage Inventory Control

Consider a multistage inventory control problem, where the demand is an uncertain parameter and our goal is to maximize profitability over T time stages. The sequence of events, and their related costs and revenues is as follows: At the beginning of time t we have non-negative inventory I_t, for which we must pay a per-unit holding cost

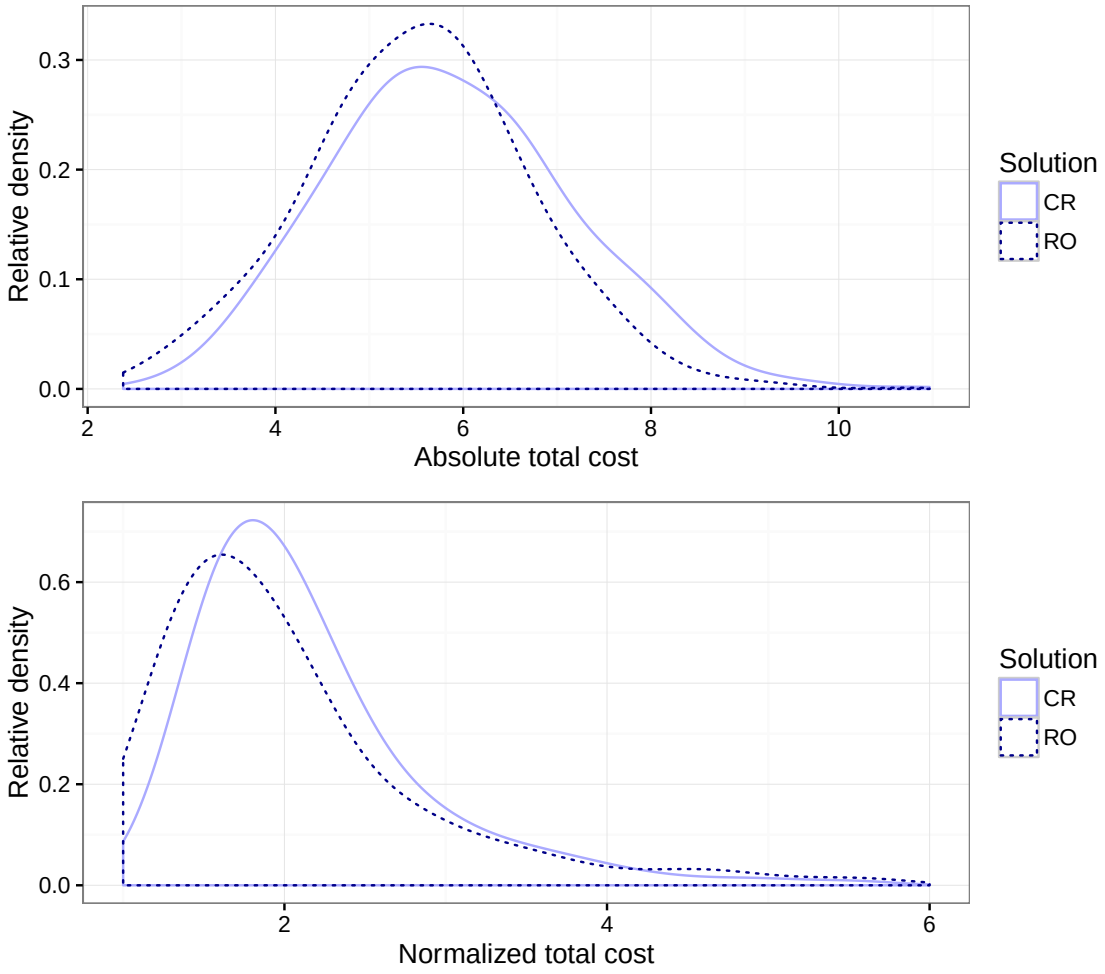

Figure 5-5: Empirical distribution of absolute total costs (top) and total cost normalized by the clairvoyant optimal total cost (bottom) obtained by simulating costs for the CR and RO solutions to a "dense" graph ($n = 10$) described in Section 5.5.1.

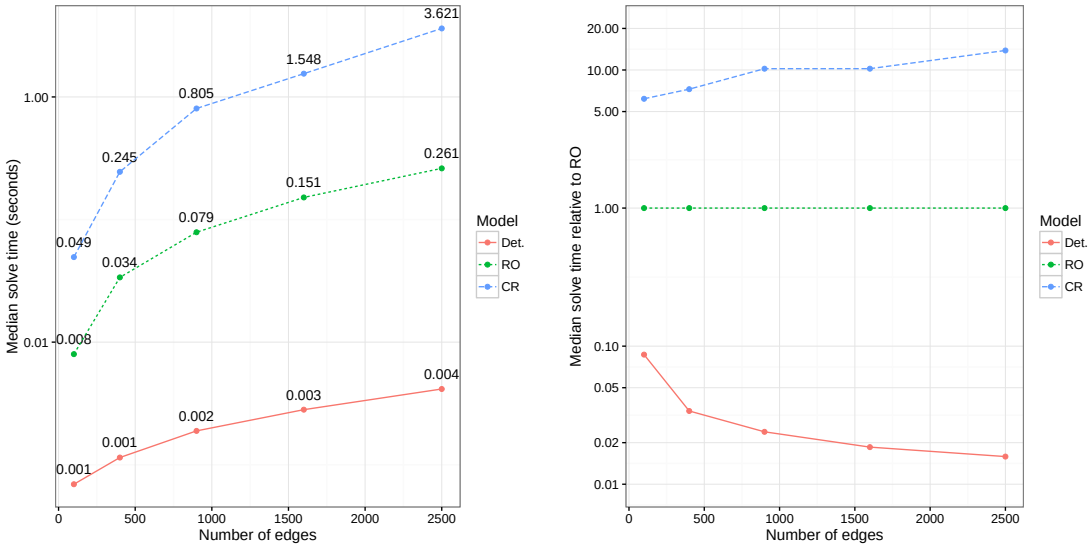

Figure 5-6: Median solve times of the minimum cost flow problem across 30 random instances of a "dense" graph ($n \in \{10, 20, 30, 40, 50\}$) used in Section 5.5.1. The left plot shows the absolute median solve time in seconds for the deterministic, robust, and competitive ratio formulations. The right plot shows the same information, normalized to the time to solve the robust formulation.

h_t. Demand d_t is then revealed, allowing us to sell $s_t \leq \min\{d_t, I_t\}$ units of stock at unit price p_t. We can then order additional stock x_t at a unit price of c_t, for delivery before the next time stage $t+1$ begins. The inventory at the beginning of time $t+1$ is thus $I_t + x_t - s_t$.

If we have an initial inventory I_1, and know the future demand $\mathbf{d}$ with certainty, then the clairvoyant problem $\Pi(\mathbf{d})$ is the LO problem

$$\begin{aligned} \Pi(\mathbf{d}) = \max_{\mathbf{x} \geq \mathbf{0}, \mathbf{s}} \quad & \sum_{t=1}^{T} \left[p_t s_t - c_t x_t - h_t \left(I_1 + \sum_{r=1}^{t-1} (x_t - s_t) \right) \right] \\ \text{subject to} \quad & 0 \leq s_t \leq \min \left\{ d_t, \; I_1 + \sum_{r=1}^{t-1} (x_t - s_t) \right\} \qquad \forall t \in \{1, \ldots, T\}. \end{aligned} \tag{5.44}$$

Note that as $I_t = I_1 + \sum_{r=1}^{t-1} (x_t - s_t)$, we do not need to explicitly include any I_t terms in our optimization problem. $\Pi(\mathbf{d})$ is a concave function of $\mathbf{d}$, and thus falls under the class of problems described in Section 5.2.

We modeled $\mathbf{d}$ as uncertain parameters drawn from a polyhedral budget uncertainty set with nominal values μ_t and deviation values σ_t. As this is a multistage

problem, we wish to model that the decisions at time t are made using all information previously known at that time, i.e., $\mathbf{x}$ and $\mathbf{s}$ are functions of $\mathbf{d}$. If we allowed "full adaptability" in how the future decisions change with the uncertain parameters, we would have an intractable optimization problem. However, an effective approximation, proposed in the RO context by Ben-Tal et al. [2004], is to substitute a *linear decision rule* for $\mathbf{x}(\mathbf{d})$ and $\mathbf{s}(\mathbf{d})$, i.e.,

$$\begin{aligned} s_t(\mathbf{d}) &= s_{t,0} + \sum_{r=1}^{t} d_r s_{t,r} \\ x_t(\mathbf{d}) &= x_{t,0} + \sum_{r=1}^{t} d_r x_{t,r}, \end{aligned} \tag{5.45}$$

where $x_{t,r}$ and $s_{t,r}$ are new auxiliary variables that characterize the policy. After making this substitution, Π_{ROB} is a robust LO problem with a polyhedral uncertainty set, and can be solved by reformulation to a deterministic LO equivalent problem. Π_{CR} and Π_{REG} can be solved either by reformulation (as $\Pi(\mathbf{d})$ has no integer variables) or using cutting planes.

To understand the quality of relative robust versus absolute robust solutions, we generated a random instance of the problem for $T = 10$, with $\mu_t \sim Uniform(20, 80)$, $\sigma_t = 0.5\mu_t$, $I_1 = 20$, $p_t \sim Uniform(0, 2)$, $c_t \sim Uniform(0, 1)$, $h_t \sim Uniform(0, 0.5)$, and $\Gamma = 3 \approx \sqrt{T}$. We then solved the problem using the Pareto formulation, with λ varied between 0 (absolute robust objective function) and 1 (competitive ratio objective function) in increments of 0.1. In Figure 5-7, we have displayed how the competitive ratio and relative robust (the absolute value normalized by Π_{RO}) components of the objective function vary with λ, as well as the overall combination of the two. The first plot reveals that we obtain a rich variation of solutions, with the CR objective function component ranging from approximately 0.43 to 0.69, and the absolute robust component ranging from approximately 85 to 123. Note that, when optimizing, we use the "normalized robust" value instead of absolute, so an absolute value of 85 corresponds to 0.69, and 123 corresponds to 1.0 (by definition). Based solely on this plot we can observe that, for example, $\lambda = 0.5$ reduces the absolute

robust objective function by less than 10 versus the RO solution, but halves the gap in CR objective between the RO and CR solutions (from 0.43 to 0.55). The second plot reveals that, for this problem, the two objective function values converge to the overall Pareto objective function as λ approaches one.

To explore how the solutions performed under simulation, we drew 10,000 samples of demand where each d_t was sampled from $Uniform\,(\mu_t - \sigma_t, \mu_t + \sigma_t)$. The resulting distribution of profits, both absolute and relative, is displayed in Figure 5-8. If we first consider the left plot, which displays absolute profits, we indeed see that $\lambda \leq 0.2$ protects against the absolute worst case, as designed. However, as we move to $0.3 \leq \lambda \leq 0.5$ we see that while the very absolute worst case does get worse, the 10^{th} percentile actually improves slightly, and the median substantially. As we approach $\lambda = 1$ the very worst case is now substantially lower than for $\lambda = 0$, but the 10^{th} is effectively unchanged and the median is much higher. When we consider the right plot, which displays relative profits, a different pattern emerges. For $\lambda = 1$ we always obtain the same relative performance, no matter what the scenario. As we decrease λ to 0.6, 0.5 and 0.4, we see a steady decrease in the 10^{th} percentile, but also the median. When we reach $\lambda = 0$, the 90^{th} percentile performance is roughly the same as the performance for $\lambda = 1$, suggesting we have got much worse by this metric.

Together, these results suggest that there is good reason to consider intermediate solutions – in particular, $\lambda = 0.6$ features an excellent trade-off. By the absolute profit metric, we are slightly worse in the very worst case than the RO solution, but better at the 10^{th} percentile and higher. By the relative profit metric, we are only slightly worse in than the CR solution at the 10^{th} percentile, roughly the same at the median, and better at the 90^{th} percentile.

5.5.3 Two-stage Facility and Stock Location

Here we consider a problem of jointly locating facilities and stock. We have n locations, $i \in \{1, \ldots, n\}$, and at each location we will experience an uncertain product demand, d_i. To meet this demand we need to stock inventory (x_i) at, and possibly build (b_i), facilities at a subset of these locations. Facilities each cost c to build, and

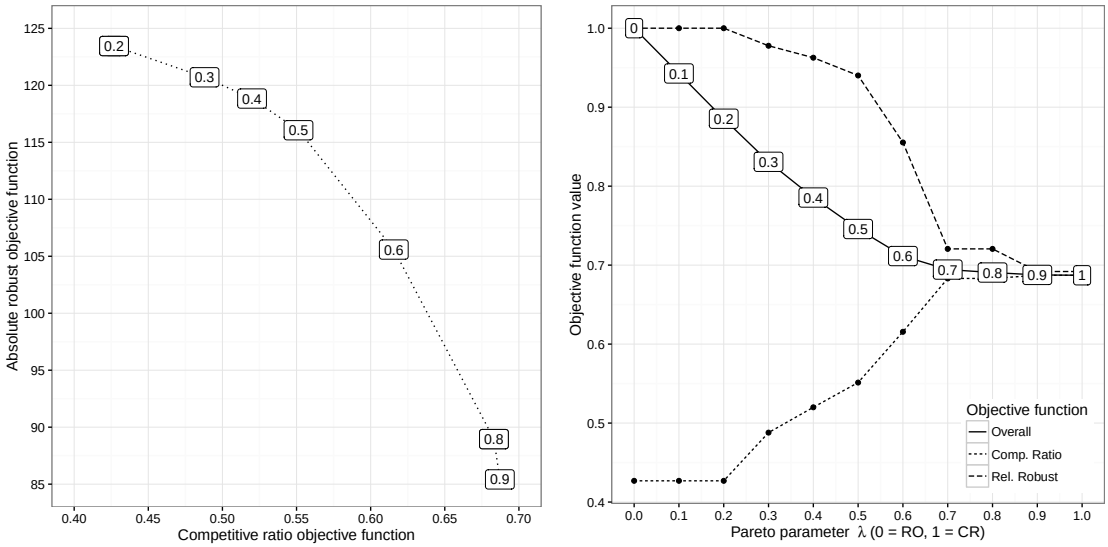

Figure 5-7: Comparisons of the value of the objective function components for the Pareto formulation of the inventory control problem (Section 5.5.2) The problem was solved for 11 values of λ between 0 and 1 inclusive. Left: the horizontal axis displays the competitive ratio (CR) component, while the vertical axis shows the absolute component, clearing displaying an efficient frontier of solutions. The values in boxes show the corresponding value of λ. Right: the horizontal axis displays the value of λ, while the CR, relative robust, and overall objective functions are displayed as lines.

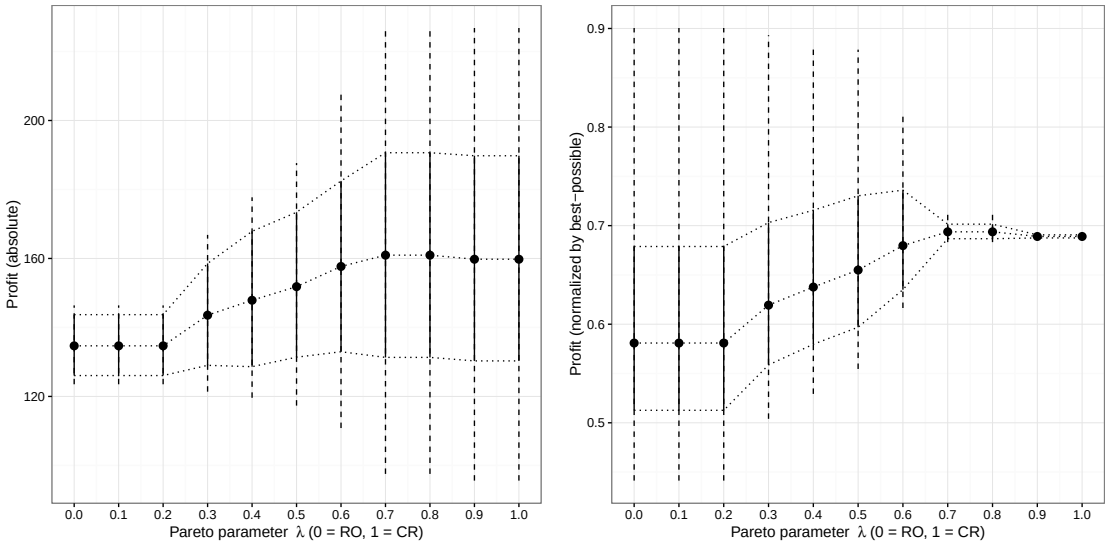

Figure 5-8: Performance in simulation of solutions for each value of λ. In both plots, the solid line indicates the 10^{th} to the 90^{th} percentile, the dashed line indicates the full range (minimum to maximum), and the solid point is the median. The dotted lines capture the values of the 10^{th},50^{th}, and 90^{th} percentiles as λ varies. Left: the distribution of absolute profits. Right: the distribution of relative profits.

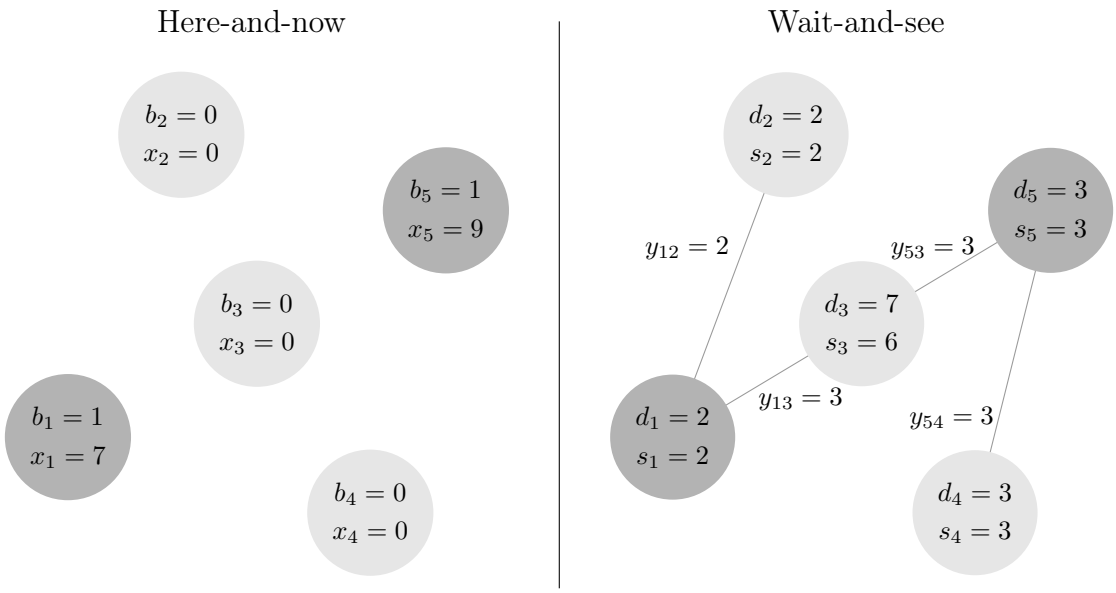

Figure 5-9: Example instance of the two-stage facility and stock location problem. Here-and-now, we must select which of the five stores we will build storage facilities ($\mathbf{b}$), and at those locations (here, $i = 1$ and $i = 5$) how much stock we will purchase ($\mathbf{x}$). The other wait-and-see decisions are made after the demand $\mathbf{d}$ is revealed. We dispatch stock from some stores to others ($\mathbf{y}$), and sell as much as possible ($\mathbf{s}$) based on demand and available stock.

both the build and ordering decisions must be made before the demand is realized. After the demand is realized, we can move stock around between locations (y_{ij}) at a per-unit cost t_{ij}. Finally for each unit of demand satisfied s_i we will obtain revenue p. An example instance and solution is visualized in Figure 5-9, where we construct and stock three facilities. In this case we considered two different settings: all facilities are constructed (which is a continuous problem), and initially no facilities are constructed (which has binary decisions). We will use the first setting to study the efficiency of reformulation versus cutting planes, and the second setting to compare the properties of the solutions for the three objective functions RO, CR, and regret.

Case 1: All facilities constructed

In the case where all the facilities are already constructed, our goal is to determine an initial allocation of stock such that we can respond to the demand as profitability as possible. The deterministic optimization problem $\Pi(\mathbf{d})$, where $\mathbf{d}$ is known, can be

written as a LO problem:

$$\begin{aligned} \Pi(\mathbf{d}) = \max_{\mathbf{x},\mathbf{s},\mathbf{y}} \quad & p\sum_i s_i - \sum_i x_i - \sum_{ij} t_{ij} y_{ij} \\ \text{subject to} \quad & s_i \leq x_i + \sum_j y_{ji} - \sum_j y_{ij} \qquad \forall i \\ & \mathbf{0} \leq \mathbf{s} \leq \mathbf{d} \\ & \mathbf{y} \geq \mathbf{0}, \mathbf{x} \geq \mathbf{0}. \end{aligned} \tag{5.46}$$

This is a maximization problem where the parameters $\mathbf{d}$ appear only in the right-hand-side of optimization problem, and so we are in the setting considered in Section 5.2. However, observe that this problem has a trivial solution that satisfies all demand, with no need to reallocate stock: $\mathbf{s}^* = \mathbf{x}^* = \mathbf{d}, \mathbf{y}^* = 0$. Thus we can write $\Pi(\mathbf{d})$ as

$$\Pi(\mathbf{d}) = p\sum_i s_i^* - \sum_i x_i^* - \sum_{ij} t_{i,j} y_{i,j}^* = (p-1)\sum_i d_i, \tag{5.47}$$

which is far simpler.

The outer problems, Π_{RO}, Π_{CR}, and Π_{REG}, are all similar in structure to the offline problem, with the key difference that $\mathbf{y}$ and $\mathbf{s}$ are "adaptive" variables whose values depend on $\mathbf{d}$. We again use a linear decision rule approximation, as in Section 5.5.2, for these two variables, i.e.,

$$\begin{aligned} y_{i,j}(\mathbf{d}) &= \sum_k y_{i,j,k} d_k + y_{i,j,0} \\ s_i(\mathbf{d}) &= \sum_k s_{i,k} d_k + s_{i,0}. \end{aligned} \tag{5.48}$$

We used a budget uncertainty set, with nominal demands μ_i, deviations σ_i, and budget $\Gamma = \sqrt{n}$.

We solved the regret version of this problem by reformulation (i.e, taking the dual of the inner problem) and by cutting planes (i.e., solving the inner problem iteratively) to compare their relative performance – see Appendix B for the derivation of the reformulation. Additionally, we solved the RO version as a point of comparison. For

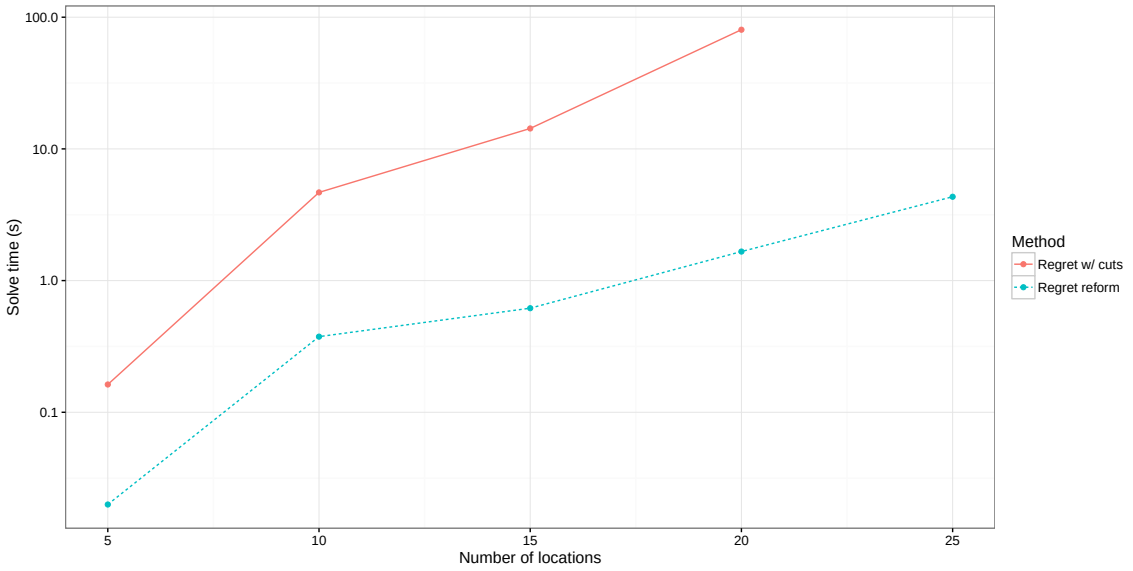

Figure 5-10: Solution times for the stock allocation problem in Section 5.5.3. The solid line shows the times for solving the regret problem with cutting planes, and the dashed line shows the times for solving the reformulation. The RO version of the problem takes the same time as the regret reformulation. The cutting plane problem for size 25 and higher took longer than 100 seconds to solve, and was terminated early.

each size $n \in \{5, 10, 15, 20, 25\}$ we created and solved 10 instances. We set $p = 3$, sampled μ_i from $Uniform(50, 150)$, and deviation σ_i was set to $0.5\mu_i$. Transport costs t_{ij} were determined by randomly locating facilities on the unit square, and setting t_{ij} to the Euclidean distance between them. We found that the reformulation was much faster than the cutting plane method, and took essentially the same amount of time to solve as the RO version of the problem (Figure 5-10). For an instance of size 20, the median solve time for the cutting plane method was approximately 80 seconds, versus 2 seconds for the reformulation. This suggests that optimizing for a relative regret objective function is essentially no harder than optimizing for an absolute robust objective function in settings such as those considered here.

Case 2: No facilities initially

If no facilities are constructed initially, then we must decide which facilities to construct jointly with the amount of stock to allocate. The deterministic optimization

problem $\Pi(\mathbf{d})$ can be written as a MILO problem:

$$\begin{aligned}
\Pi(\mathbf{d}) = \max_{\mathbf{b},\mathbf{x},\mathbf{s},\mathbf{y}} \quad & p\sum_i s_i - \sum_i x_i - c\sum_i b_i - \sum_{ij} t_{ij}y_{ij} \\
\text{subject to} \quad & s_i \leq x_i + \sum_j y_{ji} - \sum_j y_{ij} \qquad \forall i \\
& \mathbf{0} \leq \mathbf{s} \leq \mathbf{d} \\
& \mathbf{0} \leq \mathbf{x} \leq M\mathbf{b} \\
& \mathbf{y} \geq \mathbf{0},\ \mathbf{b} \in \{0,1\}^n.
\end{aligned} \tag{5.49}$$

We are again in the setting considered in Section 5.2, and as we have integer variables in $\Pi(\mathbf{d})$ we must use the cutting plane approach described in Section 5.2.3.

We compared the three objective functions (RO, competitive ratio, and regret) using an instance of size $n = 10$ generated using the same distribution as above, with the additional parameter c set to 30. To evaluate them we generated 1000 demand scenarios, where $d_i \sim Normal(\mu_i, 0.5\mu_i)$. We then solved $\Pi(\mathbf{d})$ to obtain the best-possible profit for each scenario, then for each of the three solutions we again solved $\Pi(\mathbf{d})$ with the here-and-now variables $\mathbf{x}$ and $\mathbf{b}$ fixed to determine the optimal dispatch decisions and thus the realized profit.

The three solutions were fairly similar in terms of facilities built. All three built three facilities, with two facilities in common to all three solutions. The robust solution selected a different third facility from the other two, also it was located in a similar position. The bigger difference was the amount of stock purchased in total across the facilities. The RO solution purchased only 781 units, compared with 990 for the CR solution and 1045 for the regret solution. This difference in stock purchased is possibly best explanation for the different simulation results, visualized in Figures 5-11, 5-12, and 5-13. The three figures show the distribution of absolute profits, relative profits (i.e., CR), and profit regret respectively for each of the three solutions as well as the "optimal" clairvoyant solution.

If we first consider the absolute profits (Figure 5-11), we notice that in the very worst case scenarios, all the solutions can perform quite poorly, even the optimal

solution. On the right-hand-side of the plot, we see that there are some scenarios where the optimal solution is able to profit handsomely, and none of the solutions can match it. On the left-hand-side of the plot, up to the 25^{th} percentile we see that the RO solution has indeed succeeded in protecting absolute profits, but that it fails to take advantage of "good" situations – leaving a lot of profit on the table, possibly due to a lack of stock to match demand.

When we consider competitive ratios (Figure 5-12) we see that the CR solution does indeed do best at the lower percentiles, but that both the regret and CR solutions are very similar and far superior to the robust solution from the 5^{th} percentile and higher. The gap closes again somewhat for the very best 10% of cases. A similar story repeats for regret (Figure 5-13), with the regret solution now holding the edge in the very worst cases, but otherwise looking very similar to the competitive ratio solution in both figures. The key difference is in the very worst case scenarios, where the robust solution has noticeably worse regret than the alternatives – again, possibly due to a lack of stock leading to some high-regret situations when demand is higher.

From this we conclude that if absolute worst-case profit is indeed the key criterion, then the RO solution effectively protects against this risk. However, it does so at a penalty on the majority of scenarios by the absolute profit measure, and for almost all scenarios by the other two measures. This suggests that the regret objective functions should be considered, possibly in combination with the RO objective function.

5.6 Conclusions

We have shown in this paper that problems with "relative" robust objective functions, including robust regret and competitive ratio objective functions, are computationally practical to solve and in many cases of interest are no more difficult than "absolute" robust objective functions. In particular, when the clairvoyant (offline) problem is a concave function of the uncertain parameters, we can use duality to obtain reformulations or solve a series of cutting-plane problems. When the clairvoyant problem is a convex function of the uncertain parameters, we showed that in general this problem

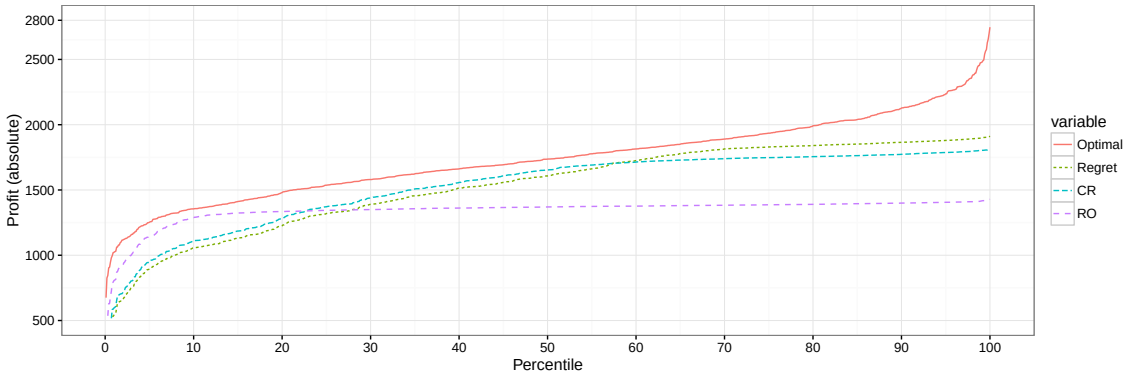

Figure 5-11: Performance in simulation of the clairvoyant optimal, RO, CR, and regret solutions for the facility location problem. Each line reflects the distribution of each solution's absolute profit in simulation – this is the objective function that the "RO" solution optimizes for. The vertical axis was truncated at 500.

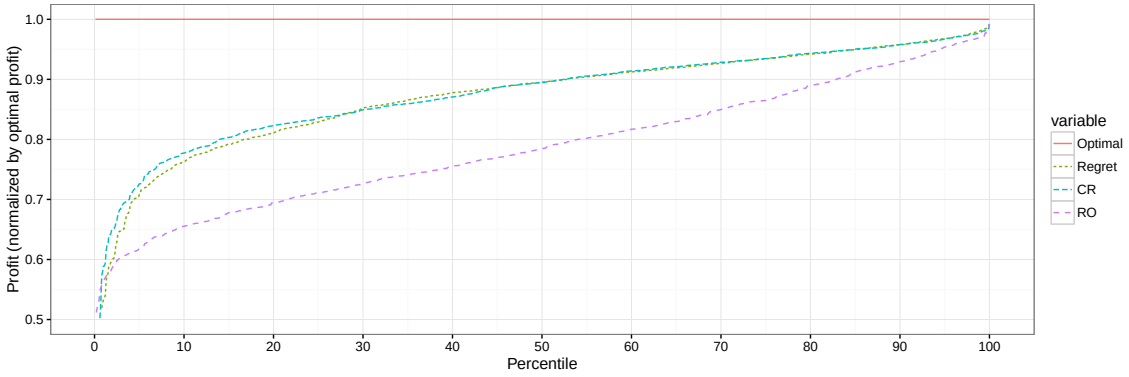

Figure 5-12: Performance in simulation of the clairvoyant optimal, RO, CR, and regret solutions for the facility location problem. Each line reflects the distribution of each solution's relative profit (normalized by the optimal solution) in simulation – this is the objective function that the "CR" solution optimizes for. The vertical axis was truncated at 0.5, and the "optimal" solution always achieves ratio 1 by construction.

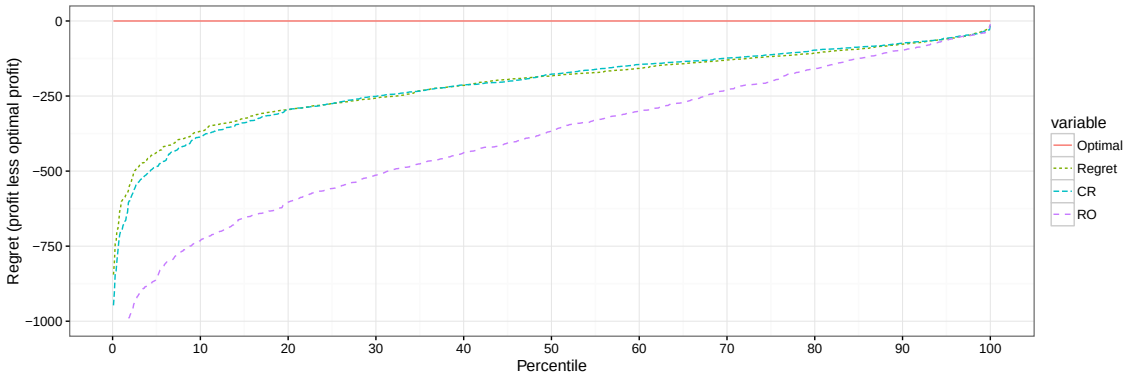

Figure 5-13: Performance in simulation of the clairvoyant optimal, RO, CR, and regret solutions for the facility location problem. Each line reflects the distribution of each solution's regret (profit less the optimal profit) in simulation – this is the objective function that the "Regret" solution optimizes for. The vertical axis was truncated at -1000, and the "optimal" solution always achieves zero regret by construction.

is intractable, but that for the popular "budget" polyhedral uncertainty set we can use a cutting-plane method that can make use of very efficient mixed-integer linear solvers. We also proposed combining both relative and absolute notions of robustness to construct a Pareto-efficient set of robust solutions, that ideally blend the properties of both.

To compare and contrast these relative objective functions, we considered three cases. The first case, minimum-cost flow, demonstrated the intuitive notion that neither absolute nor relative robust solutions dominate the other across all problem instances, but that it is possible for a given instance for one solution to be more desirable in the majority of realizations of the uncertain parameters. The second case, multistage inventory control, primarily exhibited the usefulness of considering the Pareto-efficient combined-objective solutions. In particular, if one considers near-worst-case (i.e., 10^{th} percentile) performance instead of worst-case in the simulation results, then solutions that put equal weight on both the competitive ratio and absolute robust objective functions performed the best. The third case, two-stage facility and stock location, clearly demonstrated that while absolute robust solutions do deliver worst-case protection (in this instance, providing the best solutions in the bottom 25% of scenarios), they can miss the vast majority of opportunities (with much higher

regret than the relative robust solutions).

Based on the results of this paper, we suggest that when considering a robust perspective of optimization under uncertainty, and given the practicality of solving for relative robust solutions, that the Pareto-efficient set of solutions described here should be considered. The process of selecting a particular solution can then be aided by simulation studies, "back-testing" with historical data, and by including the risk preferences of the decision maker.

Chapter 6

JuMPeR: Algebraic Modeling for Robust and Adaptive Optimization

The histories of computing and the solution of mathematical optimization problems are intricately linked, as rapid increases from the 1950s onwards in the availability and power of computers were applied widely to solving a variety of planning problems in industry and military applications [Orchard-Hays, 1984]. However, as the capability to solve ever-larger problems grew, a gap developed between the ability to solve larger problems and the ability to describe and manipulate them. Input formats for solvers were painful and slow for people to use, and were proving a major barrier to the further adoption of optimization technology [Fourer, 2012].

Algebraic modeling languages (AMLs) were developed in the late 1970s as a solution to this problem. They enabled users to express their optimization problems in a natural format that is similar to the original mathematical expressions, and automate the translation to a lower-level format suitable for solvers. Two of the first AMLs that made a significant impact on the field of optimization, and are still in use today, are the commercial packages GAMS [Brooke et al., 1999] and AMPL [Fourer et al., 2003] (created in 1978 and 1985 respectively). For the most part, AMLs have focused on two broad classes of optimization problems: mixed-integer linear optimization (MILO) and quadratic optimization (MIQO) problems, and constrained nonlinear optimization problems. For MILO problems, solvers typically expect the problem

to presented in "standard computational form" (i.e., $\min_{\mathbf{x} \geq \mathbf{0}} \mathbf{c}^T\mathbf{x},\ \mathbf{Ax} = \mathbf{b}$), so the task of an AML is primarily to generate the sparse $\mathbf{A}$ matrix and vectors $\mathbf{c}$ and $\mathbf{b}$. While there may be some problem transformations (e.g., "presolve"), in general there is normally a simple mapping between the user's input and the solver's input – but this mapping is one that would be painful to perform manually. In the majority of AMLs there is no capability for more advanced transformations or interactions with the solver – for example, in a scenario-based approach to solve a stochastic program, the user is responsible for manually iterating over the scenarios to express the recourse decision variables and constraints.

Robust optimization (RO) problems, which we address in this chapter, require the user to either manually reformulate their optimization problem using duality (necessitating the inconvenient introduction of auxiliary variables and constraints) or implementing a cutting plane method from scratch. While performing these operations even once can be time-consuming and error prone, small changes in the problem structure (e.g., changing the uncertainty set) can require substantial effort on the users part to update the model. We suggest that, much as the lack of AMLs hindered uptake of mathematical optimization on computers in the past, the lack of AMLs for more complex settings such as robust and adaptive optimization prevents uptake by practitioners and introduces inefficiencies for researchers.

In this chapter, we present JuMPeR, an extension to the JuMP modeling language [Lubin and Dunning, 2015, Dunning et al., 2015]. It is available for download with the Julia package manager. It extends the modeling capabilities of JuMP by adding primitives for uncertain parameters, adaptive decisions, and uncertainty sets, enabling a rich variety of RO problems to modeled in a high-level fashion independent of the particular solution method. It interfaces with a wide variety of solvers, inherits all the general-purpose programming capabilities of its host language Julia [Bezanson et al., 2014], and is designed to be extensible for new developments in RO and adaptive RO (ARO).

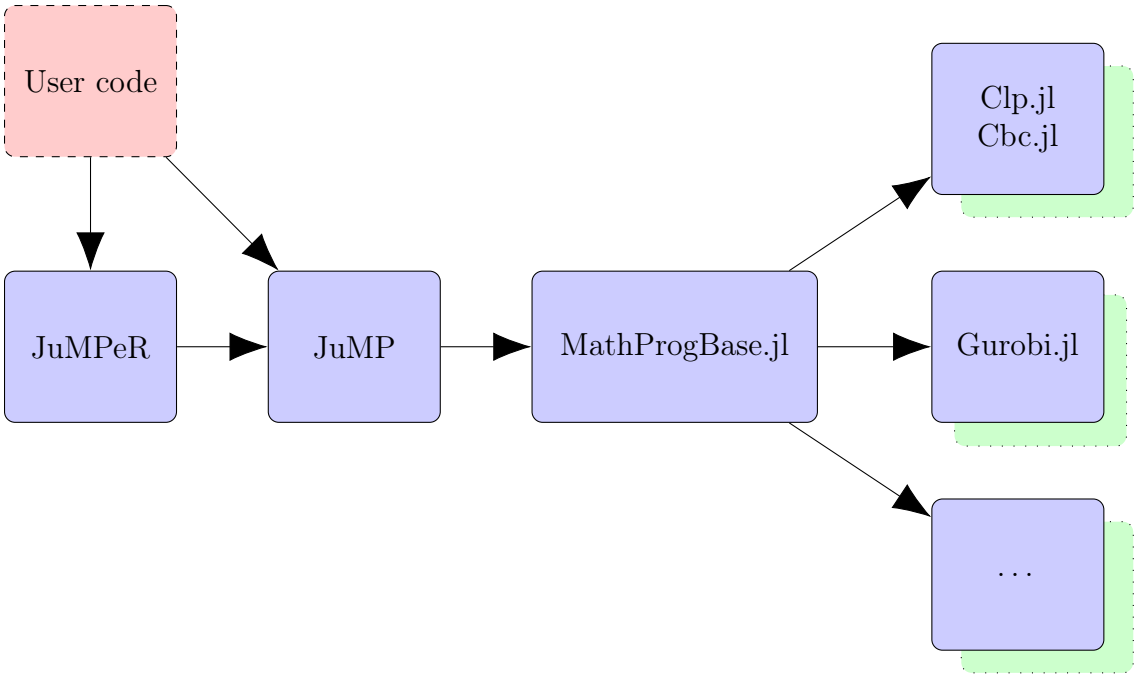

Figure 6-1: Overview of how JuMPeR interacts with related packages. User code (red, dashed) depends explicitly on both JuMPeR and JuMP. JuMP depends on the MathProgBase package which provides an abstraction over solvers. Each underlying solver library (green, loose dashes) has a thin Julia wrapper package.

Chapter structure

- In Section 6.1, we outline the capabilities of JuMPeR, its relationship to JuMP and solvers, and the key primitives available to build ARO models.

- In Section 6.2, we describe JuMPeR's uncertainty set system and how JuMPeR solves ARO problems.

- In Section 6.3, we present three case studies using JuMPeR: portfolio optimization, multistage inventory control [Ben-Tal et al., 2004], and specialized cutting plane methods.

- Finally, in Section 6.4, we compare and constrast JuMPeR with similar tools.

6.1 Overview of JuMPeR

JuMPeR is implemented as a package for the Julia programming language. Julia is a relatively new "high-level, high-performance dynamic programming language for technical computing"[1], with syntax that would be familiar for users of languages like Python or MATLAB. Some of Julia's features are particularly useful for the creation of AMLs: an expressive type system, metaprogramming (i.e., macros) to enable novel syntax, interoperability with shared libraries written in C, and garbage collection (no need for manual memory management). JuMPeR depends on the JuMP package; JuMP is itself a fully-featured AML that can model (mixed-integer) linear, quadratic, second-order cone, semidefinite, and general nonlinear optimization problems. When a user creates a RO model with JuMPeR, they use JuMP explicitly for the deterministic parts of the problem, and implicitly through JuMPeR's internals, which use JuMP to formulate auxiliary cutting plane problems and to add deterministic constraints arising from reformulations (Figure 6-1).

JuMP, and thus JuMPeR, can use the vast majority of both commercial and popular open-source solvers. This is enabled by the MathProgBase.jl package[2], which defines a shared interface that provides an abstraction over most solver idiosyncrasies. This interface is implemented by a variety of "thin" solver-specific packages that wrap solver shared libraries (normally written in C) in Julia code. MathProgBase.jl and these solver wrapper packages are all maintained together under the auspices of the JuliaOpt organization[3].

Both JuMP and JuMPeR work by composing a variety of primitives, or *types*, to construct models (refer to Figure 6-2). The highest-level type, defined by JuMP, is the `Model` type. A `Model` is a collection of decision variables (`Variable`), constraints (`LinearConstraint`, etc.), an objective function, and other metadata such as the last primal and dual solution (if the model has been solved). It also supports an extension mechanism, by which packages can build on JuMP's machinery. JuMPeR defines a

[1] As described on `http://julialang.org` on April 1st, 2016.
[2] Available at `https://github.com/JuliaOpt/MathProgBase.jl`.
[3] See `http://juliaopt.org` for more information.

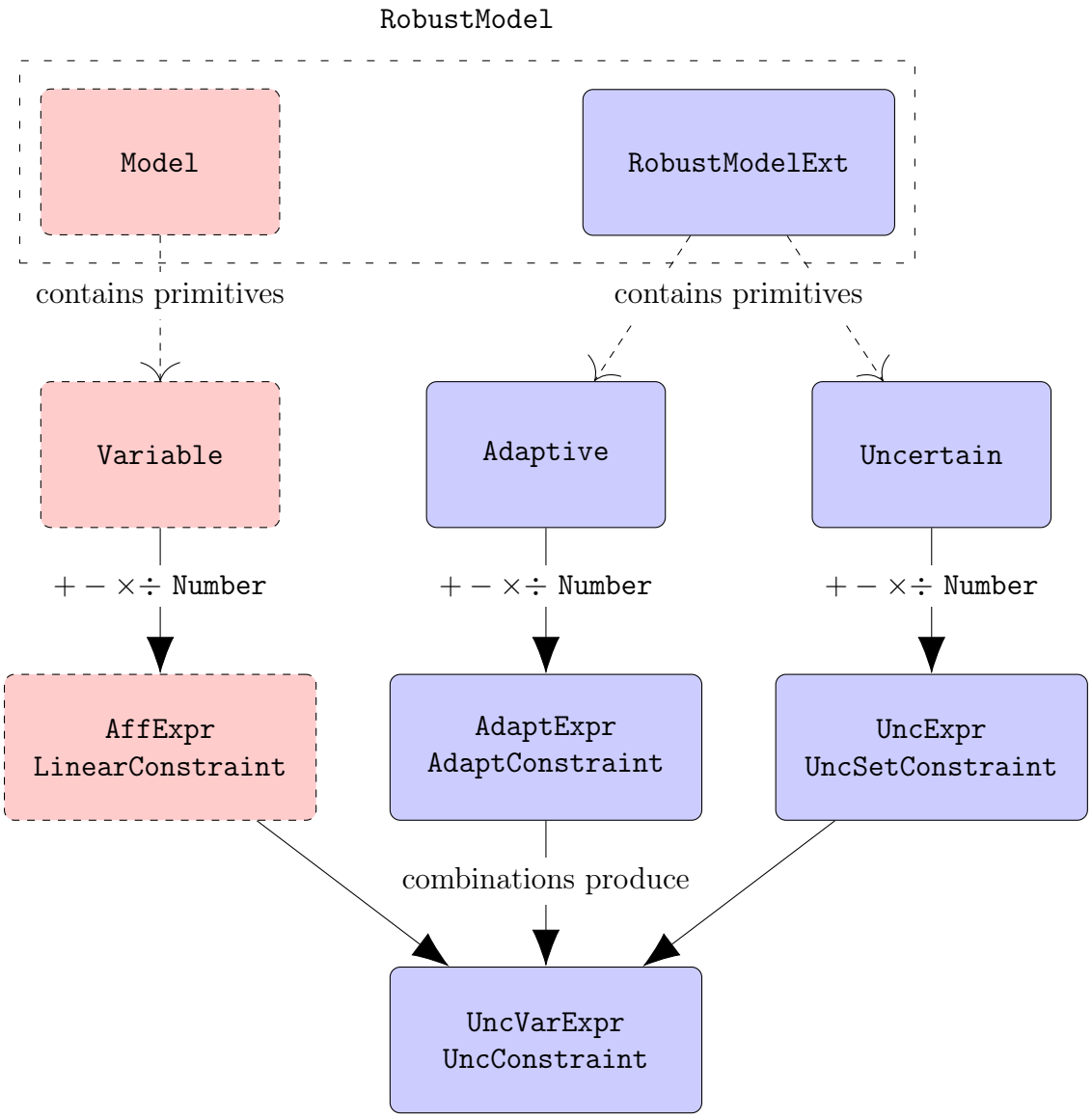

Figure 6-2: Overview of JuMPeR's type system. The red/dashed border types `Model`, `Variable`, `AffExpr` are defined by JuMP, with the remainder defined by JuMPeR. A "`RobustModel`" is a `Model` with an attached extension type `RobustModelExt` containing the RO-specific metadata.

`RobustModelExt` type that is stored in a `Model`, and defines a `RobustModel` function that produces a new model with this extension type already created and attached. We now describe in Section 6.1.1 how uncertain parameters and adaptive variables are defined, and in Section 6.1.2 we describe how these primitives combine into expressions and constraints. A discussion of how uncertainty sets are implemented is deferred to Section 6.2.

6.1.1 Uncertain Parameters and Adaptive Variables

JuMP defines the `Variable` type for constructing models, and JuMPeR introduces two more variable-like types: `Uncertain` and `Adaptive`. An `Uncertain` is an uncertain parameter – it can have lower and upper bounds, and is continuous by default but can be restricted to be binary or integer:

```
rm = RobustModel()  # Creates a new RO model
# Single uncertain parameter
@uncertain(rm, 0 <= capacity <= 100)
# Uncertain parameters indexed by numbers
@uncertain(rm, riskfactor[1:n])
# Uncertain parameters (binary) indexed by arbitrary sets
highways = [:I90, :I93, :I95]
@uncertain(rm, blocked[highways], Bin)
```

An `Adaptive` is an adaptive decision variable – the value it takes is a function of uncertain parameters. JuMPeR currently provides two simple adaptive policies by default, but can be extended to provide others: `Static` and `Affine`. Adaptive variables must be defined in relation to the uncertain parameters that they are a function of. In Section 6.2 we will discuss how `Adaptive` variables are handled when the model is solved. Here we demonstrate two examples of adaptive variables, including one where we can easily incorporate the temporal structure of a multistage problem.

```
# Creates a set of adaptive variables, where each of
# them is affinely dependent on all the risk factors
```

```
@uncertain(rm, riskfactor[1:n])
@adaptive(rm, allocation[1:n], policy=Affine,
            depends_on=riskfactor)
# Create a set of adaptive variables, where each of
# them is affinely dependent on the demand realized
# up to the time stage when the decision is made
@uncertain(rm, 10 <= demand[1:T] <= 90)
@adaptive(rm, production[t=1:T] >= 0, policy=Affine,
            depends_on=demand[1:t-1])
```

We can read the last expression as "adaptive variable `production[t]` depends on `{demand[1],...,demand[t-1]}`, for all $t \in \{1, \ldots, T\}$".

6.1.2 Expressions and Constraints

In both JuMP and JuMPeR affine expressions are stored as lists of tuples, i.e., $\{(c_1, v_1), \ldots, (c_m, v_m)\}$. For example, an expression of numbers and decision variables (e.g., the left-hand-side of a linear constraint), is an `AffExpr`, which is an alias for `GenericAffExpr{Float64,Variable}` (Figure 6-2). A linear constraint (`LinearConstraint`) is then a combination of an `AffExpr`, a sense, and a bound. JuMPeR introduces new aliases and expression-construction machinery for handling the following possibilities:

- `UncExpr`/`UncSetConstraint`: Affine expression/constraint of numbers and uncertain parameters. This is used to define uncertainty sets, and the coefficients of variables in uncertain constraints.

- `AdaptExpr`/`AdaptConstraint`: Affine expression/constraint of numbers and adaptive variables, but no explicitly appearing uncertain parameters. These will mostly like become uncertain constraints when the actual adaptive policy is inserted (see Section 6.2). Examples of these can be seen in the multistage inventory case in Section 6.3.2.

- `UncVarExpr/UncConstraint`: Affine expression/constraint where the "coefficients" are `UncExpr` (which includes just a number as a degenerate case), and the "variables" are either `Variable` or `Adapt` (or a mix). This expression unifies everything, and is the constraint type that is replaced with deterministic equivalents when the problem is solved.

JuMPeR doesn't support all possible constraint types. Notably, it does not have support for quadratic or semidefinite constraints on uncertain parameters, nor does it allow for constraints with uncertain parameters that are quadratic in the decision variables (i.e., uncertain second-order cone constraints). The lack of these is not inherent to the design of JuMPeR, and could be added in a future version. JuMPeR does have support for expressing simple "norm" constraints on uncertain parameters, including the $1-$, $2-$, and $\infty-$norms, e.g.,

```
@uncertain(rm, riskfactor[1:n])
@constraint(rm, norm(riskfactor, 1) <= 1)
@constraint(rm, norm(riskfactor, 2) <= 1)
@constraint(rm, norm(riskfactor, Inf) <= 1)
```

These are all used in the portfolio optimization case in Section 6.3.1.

6.2 Uncertainty Sets

Perhaps the defining feature of JuMPeR is its extensible uncertainty set system. At the heart of this system is the interaction between JuMPeR's core `solve` function, and (possibly user-defined) `UncertaintySet` types that implement the simple `AbstractUncertaintySet` interface. In this section we describe the flow of the `solve` function and its interactions with uncertainty sets.

When a user calls `solve`, the transformation of an RO (or ARO) problem into an deterministic problem begins. We can break this process down into five main stages:

1. `Adaptive` variables are "expanded".

2. Uncertainty sets are notified of which constraints they must reformulate/provide cutting planes for. They are then given the chance to complete any constraint-independent setup.

3. Uncertainty sets are asked to reformulate their associated constraints (and can chose to not do so).

4. Cutting planes:

   - If problem has only continuous variables, solve problem with current constraints. Ask uncertainty sets for any new constraints. If none, terminate. Otherwise, resolve problem and repeat.
   - If problem has any discrete variables, construct lazy constraint callback that queries uncertainty sets for any new constraints. Add lazy constraint callback and solve.

5. If requested, obtain the worst-case uncertain parameters for each constraint from the uncertainty sets.

Step 1 of 5: "Expanding" Adaptive Variables

The first step in the solution process is to replace all `Adaptive` variables with a combination of normal `Variable`s and uncertain parameters. This is broken down into two phases. In the first phase, an `UncVarExpr` is created for each `Adaptive` in the model, which will be spliced into the model wherever the adaptive variable appears. For `Static` variables, this simply involves creating a new `Variable` with the same bounds as the `Adaptive`, and setting the expression equal to this new variable. For `Affine` variables, the process is slightly more involved. A new `Variable` p_i is created for each uncertain parameter u_i that the variable depends on, as well as an independent variable p_0 – all these variables have no bounds. The expression is then equal to $p_0 + \sum_i u_i p_i$. To handle the bounds on the adaptive variable, up to two constraints on this expression may be added. This is most easily understood through the following snippet, where both versions are equivalent.

```
@defUnc(rm, u[1:n])
# This...
@defAdapt(rm, lb <= x <= ub, policy=Affine, depends_on=u)
# ... is equivalent to the block of code
@defVar(rm, p[1:n])
@defVar(rm, p_indep)
x = p0 + dot(p, u)
@addConstraint(rm, x >= lb)
@addConstraint(rm, x <= ub)
```

The second phase is then, for every constraint with an adaptive variable in it (i.e., all `AdaptConstraint` and some or all of the *UncConstraint*), a new *UncConstraint* is made with these per-`Adaptive` expressions inserted. If the user has a constraint with an affine adaptive variable multiplied with an uncertain parameter, then an error will be thrown at this point as quadratic functions of uncertain parameters and variables are currently unsupported.

Step 2 of 5: Uncertainty Set Setup

After the `Adaptive` variables have been processed out, we are left with a model containing only uncertain parameters and "normal" decision variables. At this point we set up any uncertainty sets associated with the model. In this context, an uncertainty set is a type that implements the `AbstractUncertaintySet` interface. Each constraint can be explicitly associated with an uncertainty set, but if one is not provided then a default model-wide uncertainty set is used. During this phase of the solution process, the uncertainty set for each `UncConstraint` is determined (either the explicitly provided one, or the default), and the `setup_set` method is called for each uncertainty set once. This is typically used by the set to do any work that can be reused for multiple constraints. For example, the uncertainty set may want to formulate the dual of the cutting plane problem for the purposes of reformulation, or it may want to create an internal JuMP model that, when solved, will produce a

cutting plane.

Step 3 of 5: Reformulation

We now give each uncertainty set the chance to reformulate any of their associated constraints. Some uncertainty sets may not support reformulation, and so will do nothing at this step. Others might support both reformulation and cutting planes, and will only take action at this stage if the user has passed an option to the uncertainty set requesting that it do so – this is the case for the in-built `BasicUncertaintySet`.

While "reformulation" may evoke the duality-based approach commonly described in the literature, the uncertainty set has full freedom in how it approaches this step. The key feature is that it is done before any solving of the RO (or a relaxation of it) takes place. For example, an uncertainty set may be defined by a finite set of scenarios, and at this stage the uncertainty set may choose to add a deterministic constraint for each of them.

Step 4 of 5: Cutting Planes

We now solve the current version of the RO problem (using, of course, only the deterministic constraints provided initially and by reformulation). If all the uncertainty sets were doing solely reformulation, we would be done with this step. If some sets are using cutting planes, then each uncertainty set is given the chance to return any number of cutting planes it chooses, using the current solution. If any cutting planes are produced, then these cuts are added and the problem is solved again. If none are produced, then we move to the next step.

The exact details of the implementation vary depending on whether there exist any discrete decision variables. If there are none, then the cutting planes are added to the JuMP model and it is resolved, in a loop. This is highly efficient for any solver that supports hot-starting using the dual simplex method, which is the case for the majority of solvers for LO problems. If there are discrete variables, then the above approach would be very inefficient as integer optimization solvers do not have this "hot-start" capability. Instead, we use the "lazy constraint callback" feature that many

of these solvers support to integrate cutting planes into the solution process. In this approach, the solver queries JuMPeR at each integer solution whether there any "lazy" constraints that it is not aware of that would make the solution infeasible. JuMPeR then asks this question of each of the uncertainty sets and manages communicating these constraints back to the solver. For more discussion of cutting-plane methods for MIO problems, see Chapter 2.

Step 5 of 5: "Active" Scenarios

Some algorithms, including those described in Chapter 3, require access to the worst-case uncertain parameters at optimality. We refer to particular realizations of the uncertain parameters as scenarios, and thus "active" scenarios are the scenarios for which each constraint has minimal slack (or possibly zero slack, which is an active constraint). This final step is an optional one, and is only used if the `active_scenarios` flag is passed to the solve function. If that flag is passed, then each uncertainty set will be asked to produce at most a single scenario for each of its constraints. These can then be accessed on a per-constraint basis by the user after the problem is solved, similar to how dual values are accessed for normal LO problems.

6.3 Case Studies

Here we present three case studies that demonstrate different capabilities of JuMPeR:

- In Section 6.3.1, we model a simple single-stage portfolio optimization problem. This demonstrates the basics of JuMPeR, and demonstrates how code can be structured to easily switch between different uncertainty sets with minimal user effort.
- In Section 6.3.2, we model a multi-stage inventory control problem. This demonstrates how we can use affine adaptive variables (linear decision rules).
- In Section 6.3.3, we demonstrate the implementation of a specialized cutting plane generator for "budget" polyhedral uncertainty sets.

6.3.1 Portfolio Construction

Our first case is a simple single-stage portfolio construction problem. We have n assets in which we can invest, and our decision is what fraction $x_i \geq 0$ of our wealth to invest in each asset i. The return on asset i is an uncertain parameter r_i, which we model as being drawn from an uncertainty set $\mathcal{U}$. This leads to the RO problem

$$\begin{aligned} \max_{\mathbf{x},z} \; & z \\ \text{subject to } & z \leq \mathbf{r}^T\mathbf{x} \qquad \forall \mathbf{r} \in \mathcal{U} \\ & \mathbf{1}^T\mathbf{x} = 1 \\ & \mathbf{x} \geq \mathbf{0}, \end{aligned} \tag{6.1}$$

where we have introduced an auxiliary variable z that moves the uncertain objective function $\min_{\mathbf{r}\in\mathcal{U}} \left\{\mathbf{r}^T\mathbf{x}\right\}$ into an epigraph constraint.

We will consider two different "data-driven" uncertainty sets that differ only in their choice of norm. We assume that we have historical data for the returns of each asset, allowing us to estimate their mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$ (a matrix). The two uncertainty sets we consider are the polyhedral set

$$\mathcal{U}_P = \left\{(\mathbf{r}, \boldsymbol{\xi}) \mid \mathbf{r} = \boldsymbol{\Sigma}^{\frac{1}{2}}\boldsymbol{\xi} + \boldsymbol{\mu},\ \|\boldsymbol{\xi}\|_1 \leq \Gamma_P,\ \|\boldsymbol{\xi}\|_\infty \leq 1\right\}, \tag{6.2}$$

and the ellipsoidal set

$$\mathcal{U}_E = \left\{(\mathbf{r}, \boldsymbol{\xi}) \mid \mathbf{r} = \boldsymbol{\Sigma}^{\frac{1}{2}}\boldsymbol{\xi} + \boldsymbol{\mu},\ \|\boldsymbol{\xi}\|_2 \leq \Gamma_E\right\}, \tag{6.3}$$

where $\boldsymbol{\Sigma}^{\frac{1}{2}}$ can be obtained by the Cholesky decomposition of $\boldsymbol{\Sigma}$, and where Γ_P and Γ_E control the conservatism of each of the sets. We can consider these $\boldsymbol{\xi}$ to be underlying market factors that induce correlations amongst the returns of the assets. If we take a reformulation approach to solving the RO problem, then the deterministic problem will be a linear optimization (LO) problem in the case of $\mathcal{U}_P$, and a second-order cone optimization (SOCO) problem in the case of $\mathcal{U}_E$.

To demonstrate solving this problem with JuMPeR, we will create a function that takes the past returns (as a matrix with one column per asset), the uncertainty set type, and the value of Γ. We first load JuMP and JuMPeR, but will not explicitly load any solver – instead, one will be selected automatically from the solvers that have been installed depending on the problem class.

```
using JuMP, JuMPeR
```

We now start our function and initialize the deterministic portion of the problem, which is a direct translation of the mathematical description in Equation (6.1). Note the use of non-Latin characters, such as Γ: this is fully supported by Julia and is commonly used for mathematical code such as this. Apart from `RobustModel`, which is defined by JuMPeR, this following lines are using the functionality of JuMP.

```
function solve_portfolio(n, past_returns, set_type, Γ)
m = RobustModel()
@variable(m, 0 <= x[1:n] <= 1)
@constraint(m, sum(x) == 1)
@variable(m, z)
@objective(m, Max, z)
```

Before constructing the uncertainty set we need to extract $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ from the return data, and calculate $\boldsymbol{\Sigma}^{\frac{1}{2}}$ (which is called `L` in the code).

```
μ = vec(mean(past_returns, 1)) # Column mean, as vector
Σ = cov(past_returns)
L = full(chol(Σ))' # Lower Cholesky factor, as matrix
```

We now use JuMPeR to create the uncertainty set, which has four components. First, we define the uncertain parameters $\mathbf{r}$ that appear directly in the model. Second, we define the underlying factor uncertain parameters $\boldsymbol{\xi}$. Third, we connect $\mathbf{r}$ and $\boldsymbol{\xi}$ through $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}^{\frac{1}{2}}$. Finally, we apply norm constraints on the factors.

```
@uncertain(m, r[1:n])
@uncertain(m, ξ[1:n])
```

```
@constraint(m, r .== L*z + μ)
if set_type == :Polyhedral
    @constraint(m, norm(z,   1) <= Γ) # ‖z‖₁ ≤ Γ
    @constraint(m, norm(z, Inf) <= 1) # ‖z‖∞ ≤ Γ
else
    @constraint(m, norm(z,   2) <= Γ) # ‖z‖₂ ≤ Γ
end
```

We then link the uncertain parameters $\mathbf{r}$ with decision variables $\mathbf{x}$ and z, and solve the model. Internally, JuMPeR will reformulate the problem to a deterministic problem and hand it off to a solver. We return the asset allocation and end the function.

```
@constraint(m, z <= dot(r, x))
solve(m)
return getvalue(x)
end  # function
```

By arranging the code in a function we are able to easily evaluate the model for a variety of parameters, and embed the robust optimization model into complex simulations and software. This also demonstrates the smooth transition between a deterministic model and a robust model: we can simply replace the uncertainty set code with $\mathbf{r} = \boldsymbol{\mu}$ and vice versa, staying with the same modeling and solver infrastructure. This is in contrast to RO-only modeling tools like ROME.

6.3.2 Multistage Inventory Control

In our second case, we show the implementation of the multistage inventory control problem described by Ben-Tal et al. [2004]. In this problem, we must determine production levels at each factory i and time period t for a single product across a T period planning horizon. All demand must be satisfied, and there are constraints on the total amount of production at each time period and the amount of inventory we can store. The deterministic optimization model, in the notation of Ben-Tal et al.

[2004], is

$$\begin{aligned}
&\min_{p_i(t),F} \; F \\
&\text{subject to } \sum_{t=1}^{T}\sum_{i=1}^{I} c_i(t)p_i(t) \le F \\
&\qquad 0 \le p_i(t) \le P_i(t), \qquad \forall i \in \{1,\dots,I\}, t \in \{1,\dots,T\} \\
&\qquad \sum_{t=1}^{T} p_i(t) \le Q_i \qquad \forall i \in \{1,\dots,I\} \\
&\qquad V_{min} \le v(1) + \sum_{s=1}^{t}\sum_{i=1}^{I} p_i(s) - \sum_{s=1}^{t} d_s \le V_{max} \qquad \forall t \in \{1,\dots,T\},
\end{aligned} \tag{6.4}$$

where $p_i(t)$ is the amount produced of the product at a factory i at time t, $c_i(t)$ is the unit cost of production at the same, $P_i(t)$ is the production capacity for factory i, Q_i is the cumulative production capacity, and V_{min} and V_{max} are the minimum and maximum total inventory limits. In the robust setting the demand d_t is an uncertain parameter, and the production decisions at time t can be made with full knowledge of the demand realized at time periods $1,\dots,t-1$. In Ben-Tal et al. [2004] the demand belongs to a box uncertainty set, and the adaptive production decisions are approximated with an affine adaptability policy, also known as a linear decision rule:

$$p_i(t) = \beta^0_{i,t} + \sum_{r=1}^{t-1} \beta^r_{i,t} d_r,$$

where $\beta^r_{i,t}$ are auxiliary variables that define the policy.

We will solve an instance of this model using the same parameters as in Ben-Tal et al. [2004]. As before we must first load JuMP and JuMPeR. We then define the parameters, which is (JuMPeR-independent) standard Julia code.

```
using JuMP, JuMPeR
I = 3            # Number of factories
T = 24           # Number of time periods
# Nominal demand
```

```
d_nom = 1000*[1 + 0.5*sin(π*(t-1)/12) for t∈1:T]
θ = 0.20        # Uncertainty level
α = [1.0, 1.5, 2.0] # Production costs
c = [α[i]*(1 + 0.5*sin(π*(t-1)/12)) for i∈1:I, t∈1:T]
P = 567          # Maximimum production per period
Q = 13600        # Maximumum production overall
Vmin = 500       # Minimum inventory at warehouse
Vmax = 2000      # Maximum inventory at warehouse
v1 = Vmin        # Initial inventory
```

We can now initialize our model and the uncertain parameters, which belong to a simple box uncertainty set where each uncertain parameter falls in a interval.

```
rm = RobustModel()
@uncertain(rm, d_nom[t]*(1-θ) <= d[t=1:T] <= d_nom[t]*(1+θ))
```

To define the adaptive production decisions $p_i(t)$ we can use `Adaptive` variables, which only require that we define what uncertain parameters the variable should be a function of, and the structure of the policy (in this case, affine).

```
@adaptive(rm, 0 <= p[i=1:I,t=1:T] <= P,
              policy=Affine, depends_on=d[1:t-1])
```

We can read the above line of code as "adaptive variable `p[i,t]`, which belongs to the model `rm`, is an affine function of uncertain parameters `d[1]` through `d[t-1]`". We next define an auxiliary variable F to represent the objective function value, and constrain it as in the Equation (6.4) above.

```
@variable(rm, F)  # Overall cost
@objective(rm, Min, F)
@constraint(rm, F >= sum{c[i,t]*p[i,t], i=1:I, t=1:T})
```

Note that while the objective function constraint doesn't explicitly include any uncertain parameters, as `p[i,t]` is a function of `d` this constraint will be handled as an uncertain constraint when transforming the problem later. The remaining tasks are

to constrain the total production to respect the cumulative limit, and to ensure we do not exceed the inventory limits. All these constraints are uncertain constraints, as they include `d` explicitly or implicitly (as part of the production decision).

```
for i in 1:I
    @constraint(rm, sum{p[i,t], t=1:T} <= Q)
end
for t in 1:T
    @constraint(rm, v1 + sum{p[i,s], i=1:I, s=1:t}
                          - sum{d[s], s=1:t} >= Vmin)
    @constraint(rm, v1 + sum{p[i,s], i=1:I, s=1:t}
                          - sum{d[s], s=1:t} <= Vmax)
end
```

Finally, we solve the problem. By default the `BasicUncertaintySet` will be used to reformulate the problem, but if we pass the `prefer_cuts=true` option then cutting planes will be used instead.

```
solve(rm, prefer_cuts=true)
println(getobjectivevalue(rm))
```

6.3.3 Specialized Uncertainty Set (Budget)

To this point we have demonstrated the use of common polyhedral or ellipsoidal uncertainty sets. However, more exotic uncertainty sets have been proposed in the literature, including the "data-driven" sets described by Bertsimas et al. [2013a]. Many of those sets have complex descriptions that arise from applying different hypothesis tests to the provided data. The choice of test has a large impact on the computational practicality of the set: some tests correspond to simple box sets, while others require exponential cones (which are supported by few solvers). Intriguingly, some of the sets admit very simple cutting plane generation algorithms – either closed-form formulae, or the solution of a line search problem.

As detailed in Section 6.2, JuMPeR defines a simple interface that a researcher can implement for a new set, and can choose to generate cutting planes however is most efficient. Unfortunately, the implementation of the more interesting uncertainty sets in Bertsimas et al. [2013a] is too long for inclusion here. We instead present the implementation of a specialized `BudgetUncertaintySet`, that provides an efficient cutting plane method for the family of uncertainty sets

$$\mathcal{U}(\boldsymbol{\mu}, \boldsymbol{\sigma}, \Gamma) = \{(\boldsymbol{\xi}, \mathbf{z}) \mid \xi_i = \mu_i + \sigma_i z_i,\ \|\mathbf{z}\|_1 \leq \Gamma,\ \|\mathbf{z}\|_\infty \leq 1\}\,. \tag{6.5}$$

This set was initially proposed by Bertsimas and Sim [2004] (albeit not in this exact form), and the cutting plane method for this set is described in Chapter 2.

To begin defining a new uncertainty set, we must first construct a Julia type that extends (`<:`) JuMPeR's `AbstractUncertaintySet`. This type stores the parameters $\boldsymbol{\mu}$, $\boldsymbol{\sigma}$, and Γ that define the set, as well as the violation ϵ required to add a new constraint.

```
type BudgetUncertaintySet <: JuMPeR.AbstractUncertaintySet
    Γ::Int
    μ::Vector{Float64}
    σ::Vector{Float64}
    ϵ::Float64
end
```

There is a method that we take no action in, but must be defined to complete the interface for our new type: `setup_set`. This is called by the main `solve` function once per uncertainty set to let it know what constraints it has been associated with. As we do not need to do any per-constraint preparation or setup, we simply define a method that does and returns nothing.

```
setup_set(us::BudgetUncertaintySet, ...) = nothing
```

We can now implement the core of the cutting plane method. We will define a function that, given the uncertainty set and an uncertain constraint, will return the values for

the uncertain parameters in that constraint that reduce the slack as much as possible. That is, for a constraint

$$\sum_j \left(\boldsymbol{\xi}^T \mathbf{a}_j + \bar{a}_j\right) x_j + \boldsymbol{\xi}^T \mathbf{a}_j^0 \leq b, \tag{6.6}$$

find the values of $\boldsymbol{\xi}$ such that the left-hand-side is maximized. To do so, we first rearrange the terms and accumulate the coefficients for each ξ_i at the current value of $\mathbf{x}$ in the master problem, i.e.,

$$\sum_i \left(a_{0,i} + \sum_j a_{j,i} x_j +\right) \xi_i + \sum_j \bar{a}_j x_j \leq b. \tag{6.7}$$

As $\boldsymbol{\xi}_i = \mu_i + \sigma_i z_i$, we can further rearrange the constraint to isolate only the components that involve $\mathbf{z}$ – we say that the rest is the nominal part of the constraint:

$$\sum_i \left(a_{0,i} + \sum_j a_{j,i} x_j +\right) \sigma_i z_i + \sum_i \left(a_{0,i} + \sum_j a_{j,i} x_j +\right) \mu_i + \sum_j \bar{a}_j x_j \leq b. \tag{6.8}$$

```
function get_worst_case(us::BudgetUncertaintySet, con)
# Collect the coefficients for each uncertain parameter,
# as well as the nominal part
unc_x_vals = zeros(length(us.μ))
nominal_value = 0.0
# For every variable term in the constraint...
for (unc_expr, var) in linearterms(con.terms)
    # Get the value of x_j in the current solution
    x_val = getvalue(var)
    # unc_expr is ξ^T a_j for x_j. We need
    # to iterate over this expression as well.
    for (coeff, unc) in linearterms(unc_expr)
        nominal_value += coeff * us.μ[unc.id] * x_val
        unc_x_vals[unc.id] += coeff * x_val
```

```
    end
    # The deterministic part ā_j
    nominal_value += unc_expr.constant * x_val
end
# The ξ^T a_j^0 term
for (coeff, unc) in linearterms(con.terms.constant)
    nominal_value += coeff * us.μ[unc.id]
    unc_x_vals[unc.id] += coeff
end
nominal_value += con.terms.constant.constant
```

We now scale the $\mathbf{x}$ values by the "deviations" $\boldsymbol{\sigma}$, and take the absolute values of the result as we need to rank the uncertain parameters by the magnitude of this quantity. The uncertain parameters with the largest magnitudes should be set to their upper or lower bounds, as doing so has the largest effect on the left-hand-side of the original uncertain constraint.

```
scaled_vals = abs(unc_x_vals) .* us.σ
# Obtain the permutation vector of the indices as if
# we had sorted by the magnitudes. Take the top Γ.
max_inds = sortperm(scaled_vals)[(end - us.Γ + 1):end]
```

Given the uncertain parameters we should set to their bounds, we can easily calculate the effect of doing so. We will use this later to determine if we should add a new constraint or not.

```
cut_value = nominal_value + sum(scaled_vals[max_inds])
```

Finally, we determine the actual values of the uncertain parameters that achieve this value. We determine which bound to by observing the sign of the coefficients on each ξ_i: if they are positive, then setting the uncertain parameter to its upper bound should maximize the left-hand-side, and if they are negative then setting the uncertain parameter to its lower bound should do the same.

```
unc_values = copy(us.μ)
```

```
for i in max_inds
    if unc_x_vals[i] > 0
        unc_values[i] += us.σ[i] # Push up, LHS goes up
    else
        unc_values[i] -= us.σ[i] # Push down, LHS goes up
    end
end
return cut_value, unc_values
end # function
```

Given this function, we can complete the JuMPeR uncertainty set interface. The `generate_cut` function receives a list of constraints and the model, and iterates through these constraints trying to generate new constraints using the function we just defined. Finally, we do not provide reformulation support for this uncertainty set, as it would be no different from the generic duality-based reformulation supported by `BasicUncertaintySet` (which we used by default for the other constraints).

```
function generate_cut(us::BudgetUncertaintySet,
                      rm::Model, idxs::Vector{Int})
# Extract the RobustModelExt from the JuMP model
# This contains all the RO-specific information
rmext = get_robust(rm)::RobustModelExt
# The vector of new constraints we will add
new_cons = Any[]
# For each constraint we need to generate a cut for
for idx in idxs
    # Get the uncertain constraint object
    con = rmext.unc_constraints[idx]
    # Determine worst-case uncertain parameters
    cut_value, unc_values = get_worst_case(us, con)
    # Use a utility function from uncsets_util.jl
```

```
    # to check violation status
    if check_cut_status(con, cut_value, us.ϵ) != :Violate
        # No violation, no new cut
        continue  # try next constraint
    end
    # Build a deterministic constraint from the
    # uncertain constraint by filing in values
    new_con = build_certain_constraint(con, unc_values)
    push!(new_cons, new_con)
end
return new_cons
end  # function

generate_reform(us::BudgetUncertaintySet, ...) = nothing
```

6.4 Comparisons with Other Tools

JuMPeR is not the first AML that has support for RO. In this section, we describe five alternatives, and contrast their capabilities versus JuMPeR (summarized in Table 6.1). The two most commonly used RO AMLs are YALMIP [Löfberg, 2012] and ROME [Goh and Sim, 2011]. Both are implemented in MATLAB, and support many key features, including polyhedral and ellipsoidal uncertainty set reformulations. There is one commercial package with support for RO, AIMMS, and a further two more experimental frameworks that are implemented in C++: ROPI [Goerigk, 2014] and ROC [Bertsimas et al., 2016].

One of the key features for any AML is usability. AIMMS, as a dedicated standalone modeling language, does well in this regard as it has full flexibility in its syntax. YALMIP and ROME are embedded in MATLAB, but have intuitive syntax that is readable by a novice. Additionally, as MATLAB is a dynamic language, there is no need for complex memory management or a compilation stage. ROPI and ROC are

Name	JuMPeR	YALMIP	ROME	AIMMS	ROPI	ROC
Availability	Free (MPL2.1)	Free (custom)	Free (GPLv3)	Commercial	Free (MIT)	Free (unknown)
Language C++	Julia	MATLAB	MATLAB	Standalone	C++	
General?	Yes (JuMP)	Yes	No	Yes	No	No
Solvers	Many	Many	SDPT3, MOSEK, CPLEX	Many	CPLEX, Gurobi, Xpress	CPLEX
Uncertainty sets	Polyhedral, ellipsoidal, custom user-extensible	Polyhedral, ellipsoidal, conic, ...	Polyhedral, ellipsoidal, DRO	Box, ellipsoidal, convex hull, chance constraints	Scenario sets, "light" robustness	Polyhedral, ellipsoidal, DRO
Cutting planes for (MI)LO	Yes	No	No	No	No	No
Adaptive optimization	LDR	None	Deflected LDR	LDR	NA	LDR

Table 6.1: RO modeling language comparison. "Language" refers to the host language for the modeling language. "General" refers to whether the language can be/is intended to be used for a variety of problems, especially deterministic problems. "Cutting planes" refers to whether the language has support for cutting plane approaches to RO, including for robust MIO problems.

both implemented as C++ libraries, which has less elegant syntax, manual memory management, and requires a separate compilation step – all factors which slow prototyping and development. JuMPeR is most similar to YALMIP and ROME, in that it is embedded in a high-level dynamic language and must make some minor syntax concessions as a result. The following snippet demonstrates a simple RO problem in ROME, taken from the ROME documentation:

```
h = rome_begin ('simple');
newvar x y;  % Set up modeling variables
newvar z uncertain;  % scalar uncertainty
rome_box(z , 1.5 , 2.5);  % and its support
rome_maximize(12 * x + 15 * y);  % objective function
rome_constraint(x + z*y <= 40);
rome_constraint(4*x + 3*y <= 120);
rome_constraint(x >= 0);
rome_constraint(y >= 0);
h.solve;
rome_end;
```

The JuMPeR equivalent of this code is as follows:

```
h = RobustModel()
@variable(h, x >= 0)
@variable(h, y >= 0)
@uncertain(h, 1.5 <= z <= 2.5)
@objective(h, Max, 12x + 15y)
@constraint(h, x + z*y <= 40)
@constraint(h, 4x + 3y <= 120)
solve(h)
```

Related to usability is the notion of generality. By this, we mean that a user can use the AML for more than just RO. All the tools here can be used for deterministic problems, but some were made solely for the purpose. We would suggest that a

language that is solely made for the purposes of RO will generally be less suitable for deterministic problems than a language that is more general by design. In particular, we claim that YALMIP, JuMPeR and AIMMS are all have this property of generality, with JuMPeR inheriting all the features of JuMP. In all three of these tools, a user can begin with a deterministic model before smoothly transitioning to a RO model. This correlates with solver support: these three languages support a very wide variety of solvers, as they all have a base infrastructure for communicating with them that is painful to build for a dedicated RO modeling tool.

Finally, the six languages all vary significantly in what aspects of RO and adaptive RO they support. JuMPeR is the only language with first-class support for cutting planes so far. All but ROPI support polyhedral and ellipsoidal sets, while only ROME, AIMMS, and ROC have support for "distributionally robust" sets built in. In terms of more complicated uncertainty sets, JuMPeR's general framework for uncertainty sets and YALMIPs very general reformulation capabilities stand apart from the rest. Finally, JuMPeR, ROME, AIMMS, and ROC all support linear decision rules, with ROME going a step further and offering deflected linear decision rules as well.

We conclude from this comparison that JuMPeR is a valuable contribution to the landscape of RO modeling tools. It is particularly suited for problems that are linear in the decision variables, and for adaptive optimization problems. The cutting plane support makes it the best choice for many kinds of RO problems, and a good platform for conducting research into new uncertainty sets.

Appendix A

Notes for "Multistage Robust MIO with Adaptive Partitions"

A.1 Comparison of partitioning schemes for a large gap example

Consider the following problem, which was presented in Bertsimas and Goyal [2010] to demonstrate large gaps between static and fully adaptive policies:

$$\begin{aligned} z(n) = \min_{\mathbf{x}^2 \geq 0, z} \quad & z \\ \text{subject to} \quad & \mathbf{e} \cdot \mathbf{x}^2(\boldsymbol{\xi}) \leq z \quad \forall \boldsymbol{\xi} \in \Xi \\ & \mathbf{x}^2(\boldsymbol{\xi}) \geq \boldsymbol{\xi} \quad \forall \boldsymbol{\xi} \in \Xi, \end{aligned}$$

where $\Xi = \left\{ \boldsymbol{\xi} \left| \sum_{j=1}^{n} \xi_j \leq 1, \boldsymbol{\xi} \geq 0 \right. \right\}$. The initial static policy solution to this problem has objective value n, compared to the fully adaptive solution 1, and the active uncertain parameter for each component i of $\mathbf{x}$ is the case where $\hat{\xi}_i = 1$ and all other components are zero.

Our method

Our method creates n partitions, one for each component i. The partition for $i = 1$ would be

$$\Xi\left(\hat{\boldsymbol{\xi}}_1\right) = \left\{ \boldsymbol{\xi} \left| \sum_{i=1}^{n} \xi_i \leq 1, \boldsymbol{\xi} \geq 0, \xi_1 \geq \xi_2, \ldots, \xi_1 \geq \xi_n \right. \right\},$$

and similarly for the other partitions. If we consider the corresponding solution for this partition we obtain $\mathbf{x}_1^2 = \left(1, \frac{1}{2}, \ldots, \frac{1}{2}\right)$, as while for the first component the worst-case is unchanged ($\hat{\xi}_1 = 1$), the value of ξ_1 must be at least as large as ξ_i which limits the largest value any other component can take to $\frac{1}{2}$. Thus the problem at iteration 2 has n partitions and an objective of $1 + \frac{n-1}{2} = \frac{n+1}{2}$, approximately half the initial objective for $n \gg 1$.

Method of Postek and Den Hertog [2014]

The method of Postek and Den Hertog [2014] splits each partition into two sub-partitions at each iteration, by selecting the two active uncertain parameters furthest apart. At the end of the first iteration we can select the active uncertain parameters for components $j = 1$ and $j = 2$ without loss of generality. This results in two partitions

$$\Xi\left(\hat{\boldsymbol{\xi}}_1\right) = \left\{ \boldsymbol{\xi} \left| \sum_{i=1}^{n} \xi_i \leq 1, \boldsymbol{\xi} \geq 0, \xi_1 \geq \xi_2 \right. \right\},$$

and

$$\Xi\left(\hat{\boldsymbol{\xi}}_2\right) = \left\{ \boldsymbol{\xi} \left| \sum_{i=1}^{n} \xi_i \leq 1, \boldsymbol{\xi} \geq 0, \xi_1 \leq \xi_2 \right. \right\}.$$

The corresponding solutions are $\mathbf{x}_1^2 = \left(1, \frac{1}{2}, 1, \ldots, 1\right)$ and $\mathbf{x}_2^2 = \left(\frac{1}{2}, 1, \ldots, 1\right)$ respectively, giving an objective of $n - \frac{1}{2}$. At each subsequent iteration another of the components will be restricted to be no more than $\frac{1}{2}$. Thus the objective at iteration k is $n - \frac{k-1}{2}$ and the the number of partitions is 2^{k-1}. At iteration n the objective will match that of our method $n - \frac{n-1}{2} = \frac{n+1}{2}$ and will have 2^{n-1} partitions.

A.2 Admissability of rule for enforcing nonanticipativity

We now demonstrate the rule described in Section 3.4.1 is sufficient to enforce nonanticipativity, but then show by example that it is overconservative: it will enforce nonanticipativity constraints needlessly in some cases.

Proposition 5. *If $\mathbf{x}_i^t$ and $\mathbf{x}_j^t$ are the decisions at time stage $t \geq 2$ corresponding to the partitions $\Xi\left(\hat{\boldsymbol{\xi}}_i\right)$ and $\Xi\left(\hat{\boldsymbol{\xi}}_j\right)$ respectively, then enforcing $\mathbf{x}_i^t = \mathbf{x}_j^t$ (or not) according to the above decision rule ensures nonanticipativity.*

Proof. Proof

Consider the case where $\hat{\boldsymbol{\xi}}_i$ and $\hat{\boldsymbol{\xi}}_j$ are siblings. $\mathbf{x}^t$ is a function of the uncertain parameters $\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^{t-1}$. If $t \leq t^{A,B}$ then $\hat{\boldsymbol{\xi}}_i^{1,\ldots,t-1} = \hat{\boldsymbol{\xi}}_j^{1,\ldots,t-1}$, and thus we cannot distinguish from the revealed uncertain parameters between $\Xi\left(\hat{\boldsymbol{\xi}}_i\right)$ and $\Xi\left(\hat{\boldsymbol{\xi}}_j\right)$, and so must enforce the nonanticipativity constraints. If $t > t^{A,B}$ then a realization of the uncertain parameters $\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^{t-1}$ can be "assigned" to either one of the partitions (or neither) as a hyperplane based on the $t^{A,B}$ component of $\boldsymbol{\xi}$ separates the two, and so we do not need to enforce nonanticipativity constraints.

If $\hat{\boldsymbol{\xi}}_i$ and $\hat{\boldsymbol{\xi}}_j$ are not siblings, we let $\hat{\boldsymbol{\xi}}_A \leftarrow Parent\left(\hat{\boldsymbol{\xi}}_i\right)$ and $\hat{\boldsymbol{\xi}}_B \leftarrow Parent\left(\hat{\boldsymbol{\xi}}_j\right)$, and update $t_{A,B}$. If $\hat{\boldsymbol{\xi}}_A$ and $\hat{\boldsymbol{\xi}}_B$ are now siblings, then we know that all child partitions are separated by a constraint involving the terms $\boldsymbol{\xi}^{t^{A,B}}$. If $t \leq t^{A,B}$, then, using same logic as above, we must conservatively add enforce the nonanticipativity constraints (even though they may be distinguished at a lower level). If $t > t^{A,B}$ then can be assigned to one of the child partitions of either for the same reasons as above. If they are not siblings, we repeat and apply the same logic. □

As suggested in this proof, we may add constraints we do not need due to missing that two leafs might be separated in a way not easily determined just by the tree's structure.

Example 4. Consider the two-stage uncertainty set $\Xi = \{(\xi^1, \xi^2) \,|0 \leq \xi^1, \xi^2 \leq 10\}$,

with tree $\mathcal{T}$

$$\mathcal{T} = root \begin{cases} (10,0)\,, & \begin{cases} (0,0) & : A \\ (10,0) & : B \end{cases} \\ (10,10)\,, & \begin{cases} (0,10) & : C \\ (10,10) & : D \end{cases} \end{cases}$$

where we have labelled each leaf A through D. The uncertainty sets corresponding to each leaf are

$$\begin{aligned} \Xi\left(\hat{\boldsymbol{\xi}}_A\right) &= \left\{\left(\xi^1,\xi^2\right) \middle| 0 \le \xi^1 \le 5,\ 0 \le \xi^2 \le 5 \right\} \\ \Xi\left(\hat{\boldsymbol{\xi}}_B\right) &= \left\{\left(\xi^1,\xi^2\right) \middle| 5 \le \xi^1 \le 10,\ 0 \le \xi^2 \le 5 \right\} \\ \Xi\left(\hat{\boldsymbol{\xi}}_C\right) &= \left\{\left(\xi^1,\xi^2\right) \middle| 0 \le \xi^1 \le 5,\ 5 \le \xi^2 \le 10 \right\} \\ \Xi\left(\hat{\boldsymbol{\xi}}_D\right) &= \left\{\left(\xi^1,\xi^2\right) \middle| 5 \le \xi^1 \le 10,\ 5 \le \xi^2 \le 10 \right\} \end{aligned}$$

which we can manually inspect to determine that we must enforce the nonanticipativity constraints $\mathbf{x}_A^2 = \mathbf{x}_C^2$ and $\mathbf{x}_B^2 = \mathbf{x}_D^2$. However, consider the application of the decision rule above to this problem for $t = 2$, that is $\mathbf{x}^2\left(\xi^1\right)$. $\hat{\boldsymbol{\xi}}_A$ and $\hat{\boldsymbol{\xi}}_D$ are not siblings, so we must compare their parents (which are). The parents first differ in the second time stage, so we add a nonanticipativity constraint (as they appear indistinguishable knowing only ξ^1). However, this constraint is not required, because only $\xi^1 = 5$ could be in either $\Xi\left(\hat{\boldsymbol{\xi}}_A\right)$ or $\Xi\left(\hat{\boldsymbol{\xi}}_D\right)$, but that is a boundary point. Thus the rule is overconservative in this case.

A.3 Comparison of partitioning schemes for multistage lot sizing

The multistage lot sizing problem of Bertsimas and Georghiou [2015] provides a good case study to understand the differences between the method in this chapter and

the proposal of Postek and Den Hertog [2014]. We will walk through two iterations of both methods for $T = 4$, using the randomly generated uncertainty set $\Xi = \{\xi^1 = 1, 21 \leq \xi^2 \leq 89, 13 \leq \xi^3 \leq 87, 4 \leq \xi^4 \leq 100\}$. The other parameters are not directly relevant to the partitioning schemes, so we will not list them here.

We begin by solving the unpartitioned problem, which has an objective value of $1830\frac{2}{3}$. The unique active uncertain parameters we extract are

$$\hat{\boldsymbol{\xi}}_1 = (1, 21, 13, 4), \qquad \hat{\boldsymbol{\xi}}_3 = (1, 89, 13, 4), \qquad \text{and } \hat{\boldsymbol{\xi}}_5 = (1, 89, 87, 100).$$
$$\hat{\boldsymbol{\xi}}_2 = (1, 21, 87, 4), \qquad \hat{\boldsymbol{\xi}}_4 = (1, 89, 87, 4),$$

At this point the two methods diverge, as our method creates five partitions while the other creates two.

Our method

Iteration 1

Our method considers each pair of active uncertain parameters, pairwise, when constructing partitions. We now detail the construction for $\hat{\boldsymbol{\xi}}_3$. Consider first $\hat{\boldsymbol{\xi}}_3$ and $\hat{\boldsymbol{\xi}}_1$: $t_{3,1}$, the first time stage they differ, is 2. Thus the first constraint is $\xi^2 \geq 55$, with the same constraint produced for $\hat{\boldsymbol{\xi}}_2$ as well. For $\hat{\boldsymbol{\xi}}_4$ and $\hat{\boldsymbol{\xi}}_5$ we find $t_{3,4} = t_{3,5} = 3$, giving the constraint $\xi^3 \leq 50$. We can now express all five partitions:

$$\Xi\left(\hat{\boldsymbol{\xi}}_1\right) = \left\{\xi^1 = 1, 21 \leq \xi^2 \leq 55, 13 \leq \xi^3 \leq 50, 4 \leq \xi^4 \leq 100\right\},$$
$$\Xi\left(\hat{\boldsymbol{\xi}}_2\right) = \left\{\xi^1 = 1, 21 \leq \xi^2 \leq 55, 50 \leq \xi^3 \leq 87, 4 \leq \xi^4 \leq 100\right\},$$
$$\Xi\left(\hat{\boldsymbol{\xi}}_3\right) = \left\{\xi^1 = 1, 55 \leq \xi^2 \leq 89, 13 \leq \xi^3 \leq 50, 4 \leq \xi^4 \leq 100\right\},$$
$$\Xi\left(\hat{\boldsymbol{\xi}}_4\right) = \left\{\xi^1 = 1, 55 \leq \xi^2 \leq 89, 50 \leq \xi^3 \leq 87, 4 \leq \xi^4 \leq 52\right\},$$
$$\Xi\left(\hat{\boldsymbol{\xi}}_5\right) = \left\{\xi^1 = 1, 55 \leq \xi^2 \leq 89, 50 \leq \xi^3 \leq 87, 52 \leq \xi^4 \leq 100\right\}.$$

The final consideration is non-anticipativity constraints. We must enforce the following constraints, all of which are relatively intuitive as all our constraints include

only a single time stage: $x_1^2 = x_2^2$, $x_3^2 = x_4^2 = x_5^2$, $x_4^3 = x_5^3$, with similar constraints on $\mathbf{y}$. Naturally the decision x^1 is the same for all partitions as well.

Iteration 2

We improve the objective from $1830\frac{2}{3}$ to 1672 thanks to our partitioning scheme. The "contributions" to the objective from each partition are $1426\frac{1}{3}$, 1672, 1627, $1530\frac{2}{3}$, and 1672 respectively. Only two of these partitions are "active partitions", so we will further divide only these two. Here we consider partition 2 only for the sake of exposition. This partition has the active uncertain parameters $\hat{\boldsymbol{\xi}}_{2,1} = (1, 21, 87, 4)$, $\hat{\boldsymbol{\xi}}_{2,2} = (1, 21, 50, 4)$, $\hat{\boldsymbol{\xi}}_{2,3} = (1, 55, 87, 4)$, and $\hat{\boldsymbol{\xi}}_{2,4} = (1, 21, 50, 100)$. These produce the following partitions, all of which are sub-partitions of the initial partition:

$$\Xi\left(\hat{\boldsymbol{\xi}}_{2,1}\right) = \left\{\xi^1 = 1, 21 \leq \xi^2 \leq 38, 68.5 \leq \xi^3 \leq 87, 4 \leq \xi^4 \leq 100\right\},$$
$$\Xi\left(\hat{\boldsymbol{\xi}}_{2,2}\right) = \left\{\xi^1 = 1, 21 \leq \xi^2 \leq 38, 50 \leq \xi^3 \leq 68.5, 4 \leq \xi^4 \leq 52\right\},$$
$$\Xi\left(\hat{\boldsymbol{\xi}}_{2,3}\right) = \left\{\xi^1 = 1, 38 \leq \xi^2 \leq 55, 50 \leq \xi^3 \leq 87, 4 \leq \xi^4 \leq 100\right\},$$
$$\Xi\left(\hat{\boldsymbol{\xi}}_{2,4}\right) = \left\{\xi^1 = 1, 21 \leq \xi^2 \leq 38, 50 \leq \xi^3 \leq 68.5, 52 \leq \xi^4 \leq 100\right\},$$

For nonanticipativity, we must first enforce the constraints of the first iteration. As we are not further partitioning partition 1 at this time, this translates to $x_1^2 = x_{2,1}^2 = x_{2,2}^2 = x_{2,3}^2 = x_{2,4}^2$. We must also add new non-anticipativity constraints, namely $x_{2,1}^2 = x_{2,2}^2 = x_{2,4}^2$ and $x_{2,2}^3 = x_{2,4}^3$

Method of Postek and Den Hertog [2014]

Iteration 1

The first step is to identify the t we will be using in the split. As described in §5.1 of Postek and Den Hertog [2014], we aim to find the component of the uncertain parameters that has maximal "dispersion" for a t such that $t_{max} \leq t \leq t_{max} + q$. The initial t_{max} for the set is 0, and the authors used $q = 2$ in their computational experiments. We are left with just $t = 2$ as as the uncertainty set is fixed for $t = 1$.

We now proceed to creating a "2-SH". Using "Heuristic 1" of §5.2 we select $\hat{\boldsymbol{\xi}}_2$ and $\hat{\boldsymbol{\xi}}_5$ as having maximal pairwise distance over the the selected time components, giving the hyperplane $\xi^2 = 55$. Each partition has $t_{max} = 2$, and no non-anticipativity constraints are required apart from the trivial one on x^1.

Iteration 2

The objective value remains at $1830\frac{2}{3}$, with both partitions individually also "contributing" this objective. We will focus on the partition that was defined by the constraint $\xi^2 \geq 55$.

This partition has the active uncertain parameters $\hat{\boldsymbol{\xi}}_{2,1} = (1, 55, 13, 4)$, $\hat{\boldsymbol{\xi}}_{2,2} = (1, 89, 13, 4)$, $\hat{\boldsymbol{\xi}}_{2,3} = (1, 89, 87, 4)$, and $\hat{\boldsymbol{\xi}}_{2,4} = (1, 89, 13, 100)$. We now search for the t to partition on, between $t_{max} = 2$ and $t_{max}+q = 4$. The maximum "dispersion" occurs for $t = 4$, so we will construct a "4-SH". The two two $\hat{\boldsymbol{\xi}}$ with maximum pairwise distance are $\hat{\boldsymbol{\xi}}_{2,3}$ and $\hat{\boldsymbol{\xi}}_{2,4}$, so the hyperplane we construct is $-74\xi^3 + 96\xi^4 = 1292$. We would thus need ξ^4 to distinguish which partition we are in, so we need nonanticipativity constraints $x^2_{2,3} = x^2_{2,4}$ and $x^3_{2,3} = x^3_{2,4}$.

Appendix B

Reformulation of Stock Location Problem from "Relative Robust"

In this appendix we present the reformulation of robust regret version of the stock location problem in Section 5.5.3. The RO problem's objective function, with the affine policies substituted, is

$$p\sum_{i=1}^{n}\left(s_{i,0}+\sum_{k=1}^{n}d_k s_{i,k}\right)-\sum_{i}x_i-\sum_{i=1}^{n}\sum_{j=1}^{n}t_{i,j}\left(y_{i,j,0}+\sum_{k=1}^{n}d_k y_{i,j,k}\right) \tag{B.1}$$

For the purposes of reformulation we first need to collect the coefficients for each d_k, i.e.,

$$\sum_{k=1}^{n}\left(\sum_{i=1}^{n}ps_{i,k}+\sum_{i=1}^{n}\sum_{j=1}^{n}t_{i,j}y_{i,j,k}\right)d_k. \tag{B.2}$$

For the sake of further exposition, let this coefficient for each d_k be C_k

Recall that the uncertainty set we are using is a budget uncertainty set, with nominal values μ_i, deviations σ, and a "budget" of $\sqrt{n}$. Our inner problem for the

regret problem is thus

$$\begin{aligned}
\Pi^I_{REG}(\mathbf{s},\mathbf{y}) = \min_{\mathbf{d},\boldsymbol{\gamma},\bar{\boldsymbol{\gamma}}} \quad & \sum_{k=1}^{n} (C_k - p + 1)\, d_k \\
\text{subject to} \quad & d_k = \mu_k + \sigma\gamma_i \\
& \bar{\gamma}_i \geq \gamma_i, \quad \bar{\gamma}_i \geq -\gamma_i \qquad \forall i \\
& \sum_i \bar{\gamma}_i \leq \sqrt{n} \\
& 0 \leq \bar{\gamma}_i \leq 1 \qquad \forall i,
\end{aligned} \tag{B.3}$$

where the $-p+1$ terms correspond to $\Pi(\mathbf{d})$, and $\bar{\gamma}_i = |\gamma_i|$.

As d_k is completely determined by γ_k, we can further simplify the problem to

$$\begin{aligned}
\Pi^I_{REG}(\mathbf{s},\mathbf{y}) = \sum_{k=1}^{n} (C_k - p + 1)\, \mu_k + \min_{\boldsymbol{\gamma},\bar{\boldsymbol{\gamma}}} \quad & \sum_{k=1}^{n} (C_k - p + 1)\, \sigma_k \gamma_k \\
\text{subject to} \quad & \bar{\gamma}_i - \gamma_i \geq 0 \qquad \forall i \\
& \bar{\gamma}_i + \gamma_i \geq 0 \qquad \forall i \\
& \sum_i \bar{\gamma}_i \leq \sqrt{n} \\
& \bar{\gamma}_i \leq 1 \qquad \forall i \\
& \bar{\gamma}_i \geq 0 \qquad \forall i,
\end{aligned} \tag{B.4}$$

with the dual of the inner problem being

$$\begin{aligned}
\Pi^{I,D}_{REG}(\mathbf{s},\mathbf{y}) = \sum_{k=1}^{n} (C_k - p + 1)\, \mu_k + \max_{\boldsymbol{\alpha}^+,\boldsymbol{\alpha}^-,\beta,\boldsymbol{\pi}} \quad & \sqrt{n}\beta + \sum_{k=1}^{n} \pi_k \\
\text{subject to} \quad & \alpha_k^- - \alpha_k^+ = (C_k - p + 1)\, \sigma_k \qquad \forall k \\
& \alpha_k^- + \alpha_k^+ + \beta + \pi_k \leq 0 \qquad \forall k \\
& \boldsymbol{\alpha}^+ \geq \mathbf{0}, \boldsymbol{\alpha}^- \geq \mathbf{0}, \beta \leq 0, \boldsymbol{\pi} \leq 0
\end{aligned} \tag{B.5}$$

The complete reformulated regret problem, with the linear decision rule substi-

tuted, is

$$
\begin{aligned}
\Pi_{REG} = \max_{\mathbf{x},\mathbf{s},\mathbf{y},\boldsymbol{\alpha}^+,\boldsymbol{\alpha}^-,\beta,\boldsymbol{\pi}} \quad & p\sum_{i=1}^{n}\left(s_{i,0}+\sum_{k=1}^{n}\mu_k s_{i,k}\right)-(p+1)\sum_{i=1}^{n}\mu_k-\sum_{i=1}^{n}x_i- \\
& \sum_{i=1}^{n}\sum_{j=1}^{n}t_{i,j}\left(y_{i,j,0}+\sum_{k=1}^{n}\mu_k y_{i,j,k}\right)+\sqrt{n}\beta+\sum_{i=1}^{n}\pi_i \\
\text{subject to} \quad & \alpha_k^- - \alpha_k^+ = \sigma_k\left(\sum_{i=1}^{n}p s_{i,k}+\sum_{i=1}^{n}\sum_{j=1}^{n}t_{i,j}y_{i,j,k}-p+1\right) \qquad \forall k \\
& \alpha_k^- + \alpha_k^+ + \beta + \pi_k \leq 0 \qquad \forall k \\
& s_{i,0}+\sum_{k=1}^{n}d_k s_{i,k} \leq x_i + \sum_{j=1}^{n}\left(y_{j,i,0}+\sum_{k=1}^{n}d_k y_{j,i,k}\right)- \\
& \sum_{j=1}^{n}\left(y_{i,j,0}+\sum_{k=1}^{n}d_k y_{i,j,k}\right) \quad \forall i, \forall \mathbf{d}\in\mathcal{U} \\
& 0 \leq s_{i,0}+\sum_{k=1}^{n}d_k s_{i,k} \leq d_i \qquad \forall i, \forall \mathbf{d}\in\mathcal{U} \\
& 0 \leq y_{i,j,0}+\sum_{k=1}^{n}d_k y_{i,j,k} \qquad \forall i,j, \forall \mathbf{d}\in\mathcal{U} \\
& \mathbf{x}\geq\mathbf{0}, \boldsymbol{\alpha}^+\geq\mathbf{0}, \boldsymbol{\alpha}^-\geq\mathbf{0}, \beta\leq 0, \boldsymbol{\pi}\leq 0.
\end{aligned}
\tag{B.6}
$$

Bibliography

T. Assavapokee, M. J. Realff, and J. C. Ammons. Min-max regret robust optimization approach on interval data uncertainty. *Journal of Optimization Theory and Applications*, 137(2):297–316, 2008.

F. Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405, 1991.

I. Averbakh. Minmax regret solutions for minimax optimization problems with uncertainty. *Operations Research Letters*, 27(2):57–65, 2000.

I. Averbakh and O. Berman. Minimax regret p-center location on a network with demand uncertainty. *Location Science*, 5(4):247–254, 1997.

M. O. Ball and M. Queyranne. Toward robust revenue management: Competitive analysis of online booking. *Operations Research*, 57(4):950–963, 2009.

A. Ben-Tal and A. Nemirovski. Robust solutions of uncertain linear programs. *Operations Research Letters*, 25(1):1 – 13, 1999.

A. Ben-Tal and A. Nemirovski. Robust solutions of linear programming problems contaminated with uncertain data. *Mathematical Programming*, 88:411–424, 2000.

A. Ben-Tal, A. Goryashko, E. Guslitzer, and A. Nemirovski. Adjustable robust solutions of uncertain linear programs. *Mathematical Programming*, 99(2):351–376, 2004.

A. Ben-Tal, B. Golany, A. Nemirovski, and J.-P. Vial. Retailer-supplier flexible commitments contracts: a robust optimization approach. *Manufacturing & Service Operations Management*, 7(3):248–271, 2005.

A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *Robust optimization*. Princeton University Press, 2009.

D. Bertsimas and H. Bidkhori. On the performance of affine policies for two-stage adaptive optimization: a geometric perspective. *Mathematical Programming*, 153 (2):577–594, 2015.

D. Bertsimas and D. Brown. Constructing uncertainty sets for robust linear optimization. *Operations Research*, 57(6):1483–1495, 2009.

D. Bertsimas and C. Caramanis. Adaptability via sampling. In *Decision and Control, 2007 46th IEEE Conference on*, pages 4717–4722, Dec 2007.

D. Bertsimas and C. Caramanis. Finite adaptability in multistage linear optimization. *Automatic Control, IEEE Transactions on*, 55(12):2751–2766, 2010.

D. Bertsimas and I. Dunning. Multistage robust mixed integer optimization with adaptive partitions. *Operations Research (to appear)*, 2016a.

D. Bertsimas and I. Dunning. Relative robust and adaptive optimization. *submitted to INFORMS Journal on Computing*, 2016b.

D. Bertsimas and A. Georghiou. Design of near optimal decision rules in multistage adaptive mixed-integer optimization. *Operations Research*, 63(3):610–627, 2015.

D. Bertsimas and A. Georghiou. Binary decision rules for multistage adaptive mixed-integer optimization. *Available on Optimization Online*, 2016.

D. Bertsimas and V. Goyal. On the power of robust solutions in two-stage stochastic and adaptive optimization problems. *Mathematics of Operations Research*, 35(2): 284–305, 2010.

D. Bertsimas and V. Goyal. On the power and limitations of affine policies in two-stage adaptive optimization. *Mathematical Programming*, 134(2):491–531, 2012.

D. Bertsimas and M. Sim. The price of robustness. *Operations Research*, 52(1):35–53, Jan. 2004.

D. Bertsimas and J. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.

D. Bertsimas, D. Iancu, and P. Parrilo. Optimality of affine policies in multistage robust optimization. *Mathematics of Operations Research*, 35(2):363–394, 2010.

D. Bertsimas, D. B. Brown, and C. Caramanis. Theory and applications of robust optimization. *SIAM review*, 53(3):464–501, 2011a.

D. Bertsimas, D. Iancu, and P. Parrilo. A hierarchy of near-optimal policies for multistage adaptive optimization. *Automatic Control, IEEE Transactions on*, 56 (12):2809–2824, 2011b.

D. Bertsimas, V. Gupta, and N. Kallus. Data-driven robust optimization. *arXiv preprint arXiv:1401.0212*, 2013a.

D. Bertsimas, E. Litvinov, X. A. Sun, J. Zhao, and T. Zheng. Adaptive robust optimization for the security constrained unit commitment problem. *Power Systems, IEEE Transactions on*, 28(1):52–63, 2013b.

D. Bertsimas, H. Bidkhori, and I. Dunning. The price of flexibility. *submitted to Operations Research*, 2015a.

D. Bertsimas, I. Dunning, and M. Lubin. Reformulation versus cutting-planes for robust optimization. *Computational Management Science*, 13(2):195–217, 2015b.

D. Bertsimas, M. Sim, and M. Zhang. Distributionally adaptive optimization. *available as preprint*, 2016.

J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *CoRR*, abs/1411.1607, 2014. URL `http://arxiv.org/abs/1411.1607`.

S. Biller, A. Muriel, and Y. Zhang. Impact of price postponement on capacity and flexibility investment decisions. *Production and Operations Management*, 15(2): 198–214, 2006. ISSN 1937-5956.

R. Bixby, S. Ceria, C. McZeal, and M. Savelsberg. An updated mixed integer programming library: MIPLIB 3.0. *Optima*, 58:12–15, 1998.

O. Briant, C. Lemaréchal, P. Meurdesoif, S. Michel, N. Perrot, and F. Vanderbeck. Comparison of bundle and classical column generation. *Mathematical Programming*, 113(2):299–344, 2008.

A. Brooke, D. Kendrick, A. Meeraus, and R. Raman. *GAMS: A User's Guide*. Scientific Press, 1999.

R. Carvajal, S. Ahmed, G. Nemhauser, K. Furman, V. Goel, and Y. Shao. Using diversification, communication and parallelism to solve mixed-integer linear programs. *Operations Research Letters*, 42(2):186 – 189, 2014.

X. Chen, M. Sim, and P. Sun. A robust optimization perspective on stochastic programming. *Operations Research*, 55(6):1058–1071, 2007.

X. Chen, M. Sim, P. Sun, and J. Zhang. A linear decision-based approximation approach to stochastic programming. *Operations Research*, 56(2):344–357, 2008.

X. Chen, J. Zhang, and Y. Zhou. Optimal sparse designs for process flexibility via probabilistic expanders. *Available at SSRN 2400768*, 2014.

M. C. Chou, G. A. Chua, C.-P. Teo, and H. Zheng. Design for process flexibility: Efficiency of the long chain and sparse structure. *Operations Research*, 58(1):43–58, 2010.

M. C. Chou, G. A. Chua, C.-P. Teo, and H. Zheng. Process flexibility revisited: the graph expander and its applications. *Operations Research*, 59(5):1090–1105, 2011.

G. Dantzig. Linear programming. In A. Lenstra, A. Rinnooy Kan, and A. Schrijver, editors, *History of Mathematical Programming – A Collection of Personal Reminiscences*. Elsevier Science Publishers, Amsterdam, 1991.

T. Deng and Z.-J. M. Shen. Process flexibility design in unbalanced networks. *Manufacturing & Service Operations Management*, 15(1):24–32, 2013.

E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213, 2002.

I. Dunning, J. Huchette, and M. Lubin. JuMP: A modeling language for mathematical optimization. *arXiv preprint arXiv:1508.01982*, 2015.

B. Efron and R. J. Tibshirani. *An introduction to the bootstrap*, volume 57. CRC press, 1994.

M. Fischetti and M. Monaci. Cutting plane versus compact formulations for uncertain (integer) linear programs. *Mathematical Programming Computation*, 4:239–273, 2012.

P. J. Fleming and J. J. Wallace. How not to lie with statistics: the correct way to summarize benchmark results. *Communications of the ACM*, 29(3):218–221, 1986.

R. Fourer. On the evolution of optimization modeling systems. In M. Grotschel, editor, *Optimization Stories*, page 377–388. Documenta Mathematica, 2012.

R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A modeling language for mathematical programming*. Brooks/Cole, Pacific Grove, CA, 2nd edition, 2003.

D. M. Gay. Electronic mail distribution of linear programming test problems. *Mathematical Programming Society COAL Newsletter*, 13:10–12, 1985.

Z. Geng and L. Huang. Robust stability of the systems with mixed uncertainties under the IQC descriptions. *International Journal of Control*, 73(9):776–786, 2000.

M. Goerigk. Ropi - a robust optimization programming interface for C++. *Optimization Methods and Software*, 29(6):1261–1280, 2014.

J. Goh and M. Sim. Robust optimization made easy with ROME. *Operations Research*, 59(4):973–985, 2011.

J. Gorski, F. Pfeuffer, and K. Klamroth. Biconvex sets and optimization with biconvex functions: a survey and extensions. *Mathematical Methods of Operations Research*, 66(3):373–407, 2007.

P. J. Goulart, E. C. Kerrigan, and J. M. Maciejowski. Optimization over state feedback policies for robust control with constraints. *Automatica*, 42(4):523–533, 2006.

Gurobi Optimization Inc. Gurobi Optimizer Reference Manual, 2016. URL `http://www.gurobi.com`.

M. Hadjiyiannis, P. Goulart, and D. Kuhn. A scenario approach for estimating the suboptimality of linear decision rules in two-stage robust optimization. In *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*, pages 7386–7391, Dec 2011.

G. A. Hanasusanto, D. Kuhn, and W. Wiesemann. K-adaptability in two-stage robust binary programming. *Operations Research*, 63(4):877–891, 2015.

D. Iancu and N. Trichakis. Pareto efficiency in robust optimization. *Management Science*, 60(1):130–147, 2013.

W. C. Jordan and S. C. Graves. Principles on the benefits of manufacturing process flexibility. *Management Science*, 41(4):577–594, 1995.

T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R. E. Bixby, E. Danna, G. Gamrath, A. M. Gleixner, S. Heinz, A. Lodi, H. Mittelmann, T. Ralphs, D. Salvagnin, D. E. Steffy, and K. Wolter. MIPLIB 2010. *Mathematical Programming Computation*, 3(2):103–163, 2011.

T. Koch, T. Ralphs, and Y. Shinano. Could we use a million cores to solve an integer program? *Mathematical Methods of Operations Research*, 76(1):67–93, 2012.

P. Kouvelis and G. Yu. *Robust discrete optimization and its applications*. Kluwer Academic Publishers, Boston, 1997.

D.-T. Lee and R. L. Drysdale, III. Generalization of Voronoi diagrams in the plane. *SIAM Journal on Computing*, 10(1):73–87, 1981.

J. Löfberg. Automatic robust convex programming. *Optimization methods and software*, 27(1):115–129, 2012.

M. Lubin and I. Dunning. Computing in operations research using Julia. *INFORMS Journal on Computing*, 27(2):238–248, 2015.

H.-Y. Mak and Z.-J. M. Shen. Stochastic programming approach to process flexibility design. *Flexible Services and Manufacturing Journal*, 21(3-4):75–91, 2009.

H. E. Mausser and M. Laguna. Minimising the maximum relative regret for linear programmes with interval objective function coefficients. *Journal of the Operational Research Society*, pages 1063–1070, 1999a.

H. E. Mausser and M. Laguna. A heuristic to minimax absolute regret for linear programs with interval objective function coefficients. *European Journal of Operational Research*, 117(1):157–174, 1999b.

H. Mittelmann. Benchmark of parallel LP solvers. URL `http://plato.asu.edu/ftp/lpcom.html`.

A. Mutapcic and S. Boyd. Cutting-set methods for robust convex optimization with pessimizing oracles. *Optimization Methods & Software*, 24(3):381–406, 2009.

W. Orchard-Hays. History of mathematical programming systems. *IEEE Annals of the History of Computing*, 6(3):296–312, 1984. ISSN 1058-6180.

G. Perakis and G. Roels. Robust controls for network revenue management. *Manufacturing & Service Operations Management*, 12(1):56–76, 2010.

K. Postek and D. Den Hertog. Multi-stage adjustable robust mixed-integer optimization via iterative splitting of the uncertainty set. *CentER Discussion Paper Series*, 2014.

D. Simchi-Levi and Y. Wei. Understanding the performance of the long chain and sparse designs in process flexibility. *Operations Research*, 60(5):1125–1141, 2012.

D. Simchi-Levi and Y. Wei. Worst-case analysis of process flexibility designs. *Operations Research*, 63(1):166–185, 2015.

Y. Song and J. Luedtke. An adaptive partition-based approach for solving two-stage stochastic programs with fixed recourse. *SIAM Journal on Optimization*, 25(3): 1344–1367, 2015.

G. L. Vairaktarakis. Robust multi-item newsboy models with a budget constraint. *International Journal of Production Economics*, 66(3):213–226, 2000.

P. Vayanos, D. Kuhn, and B. Rustem. Decision rules for information discovery in multi-stage stochastic programming. In *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*, pages 7368–7373. IEEE, 2011.

X. Wang and J. Zhang. Process flexibility: A distribution-free bound on the performance of k-chain. *Operations Research*, 63(3):555–571, 2015.

Y. Wei and D. Simchi-Levi. Worst-case analysis of process flexibility designs. *Operations Research*, 63(1):166–185, 2015.

B. Zeng and L. Zhao. Solving two-stage robust optimization problems using a column-and-constraint generation method. *Operations Research Letters*, 41(5):457–461, 2013.

V. Zverovich, C. Fábián, E. Ellison, and G. Mitra. A computational study of a solver system for processing two-stage stochastic LPs with enhanced benders decomposition. *Mathematical Programming Computation*, 4(3):211–238, 2012.