PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Thiago de Garcia Paula Sandes Milagres

Iterative Methods for Robust Convex Optimization

Dissertação de Mestrado

Dissertation presented to the Programa de Pós-Graduação em Informatica of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informatica.

Advisor: Prof. Marco Serpa Molinaro

Rio de Janeiro
July 2019

Thiago de Garcia Paula Sandes Milagres

Iterative Methods for Robust Convex Optimization

Dissertation presented to the Programa de Pós-Graduação em Informatica of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informatica. Approved by the undersigned Examination Committee.

Prof. Marco Serpa Molinaro
Advisor
Departamento de Informatica – PUC-Rio

Prof. Marcus Vinicius Soledade Poggi de Aragao
Departamento de Informática – PUC-Rio

Prof. Thibaut Victor Gaston Vidal
Departamento de Informática – PUC-Rio

Rio de Janeiro, July 3rd, 2019

Thiago de Garcia Paula Sandes Milagres

Thiago de Garcia Paula Sandes Milagres received his B.Sc. degree in Electrical Engineering in 2017 from the Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Brazil.

Bibliographic data

Milagres, Thiago

Iterative Methods for Robust Convex Optimization / Thiago de Garcia Paula Sandes Milagres; advisor: Marco Serpa Molinaro. – Rio de Janeiro: PUC-Rio, Departamento de Informática, 2019.

v., 72 f.: il. ; 29,7 cm

Dissertação (Mestrado em Informática) – Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informatica, Rio de Janeiro, 2019.

Inclui referências bibliográfias.

1. Ciência da Computação – Teses. 2. Otimização Robusta; 3. Otimização Convexa Online; 4. Otimização Convexa; 5. Métodos Iterativos; 6. Agregação de Restrições; 7. Online Learning; 8. Multiplicative Weights Update; I. Molinaro, Marco. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informatica. III. Título.

CDD: 004

Acknowledgments

To my family, for the love, education and support. They made all of this possible.

To Julia, my girlfriend, for the example and for being on my side at all times.

To my adviser, Marco Molinaro, for the opportunity, dedication and the valuable teachings during this journey.

To my professors at PUC-Rio, for the guidance and encouragement.

To my friends, who encouraged me on this journey and helped make it possible.

To CNPq, PUC-Rio and the Informatics Department, for the support and structure.

Abstract

Milagres, Thiago; Molinaro, Marco (Advisor). **Iterative Methods for Robust Convex Optimization**. Rio de Janeiro, 2019. 72p. M.Sc. Dissertation – Departamento de Informatica, Pontifícia Universidade Católica do Rio de Janeiro.

Robust Optimization is a common paradigm to consider uncertainty in the parameters of an optimization problem. The traditional way to find robust solutions requires solving the robust counterpart of an optimization problem, which, in practice, can often be prohibitively costly. In this work, we study iterative methods to approximately solve Robust Convex Optimization problems, which do not require solving the robust counterpart. We use concepts from the Online Learning framework to propose a new algorithm that performs constraint aggregation, and we demonstrate theoretical convergence guarantees. We then develop a modification of this algorithm that, although without such guarantees, obtains better practical performance. Finally, we implement other classical iterative methods from the Robust Optimization literature and present a computational study of their performances.

Keywords

Robust Optimization; Online Convex Optimization; Convex Optimization; Iterative Methods; Constraint Aggregation; Online Learning; Multiplicative Weights Update;

Resumo

Milagres, Thiago; Molinaro, Marco. **Métodos Iterativos para Otimização Convexa Robusta**. Rio de Janeiro, 2019. 72p. Dissertação de Mestrado – Departamento de Informatica, Pontifícia Universidade Católica do Rio de Janeiro.

Otimização Robusta é uma das formas mais comuns de considerar incerteza nos parâmetros de um problema de otimização. A forma tradicional de achar soluções robustas consiste em resolver a contraparte robusta de um problema, o que em muitos casos, na prática, pode ter um custo computacional proibitivo. Neste trabalho, estudamos métodos iterativos para resolver problemas de Otimização Convexa Robusta de forma aproximada, que não exigem a formulação da contraparte robusta. Utilizamos conceitos de Online Learning para propor um novo algoritmo que utiliza agregação de restrições, demonstrando garantias teóricas de convergência. Desenvolvemos ainda uma modificação deste algoritmo que, apesar de não possuir tais garantias, obtém melhor performance prática. Por fim, implementamos outros métodos iterativos conhecidos da literatura de Otimização Robusta e fazemos uma análise computacional de seus desempenhos.

Palavras-chave

Otimização Robusta; Otimização Convexa Online; Otimização Convexa; Métodos Iterativos; Agregação de Restrições; Online Learning; Multiplicative Weights Update;

Table of contents

List of figures

List of tables

List of Abbreviations

LP	Linear Programming
QP	Quadratic Programming
QCQP	Quadratically Constrained Quadratic Programming
SOCP	Second Order Cone Programming
SDP	Semidefinite Programming
RO	Robust Optimization
OCO	Online Convex Optimization
FTL	Follow the Leader
FTRL	Follow the Regularized Leader
OGD	Online Gradient Descent
MWU	Multiplicative Weights Update
i.i.d.	Independent and identically distributed

1 Introduction

1.1 Motivation

Data used as input for optimization problems is often subject to uncertainty. Such uncertainty could have multiple sources, such as measurement errors. Even more alarmingly, in some cases the data comes from an external forecasting model, with its own sources of errors. An example is the pricing optimization setting, in which the forecasted demand passed as input could come from an entirely separate model.

While in practice such uncertainties are often ignored and deterministic optimization methods are used, a study made by Ben-Tal and Nemirovski [1] demonstrates why this may not be a good idea. They use data from the *NETLIB* collection, a well known library for Linear Programming problems, and show that by applying a small perturbation of about 0.1% on the data, the solutions x^* that were previously considered optimal not only become suboptimal, but often highly infeasible to the point of being practically useless.

In this context, there is a whole field of study, known as Sensitivity Analysis, which as defined in [2] focuses on identifying "how the uncertainty in the output can be apportioned to different sources of uncertainty in the model input". Mulvey et al. [3] argue that such methods are *reactive*, since they do not generate new recommendations, but rather serve to study the impact of perturbations in solutions that were already recommended.

More *proactive* approaches are needed: methods that are able to generate good solutions in an uncertain environment. In the field of Optimization under Uncertainty, there are many different frameworks that aim to deal with this problem. We highlight the two that are most notorious: Stochastic Optimization and Robust Optimization.

Stochastic Optimization, which is not the focus of this work, assumes that the uncertain parameters are stochastic, and have a distribution that is either known or can be estimated. Given these distributions, it seeks to find policies that are feasible for (almost) all the possible scenarios. One drawback in this approach is that in many applications, such probability distributions on the parameters are unknown and hard to estimate. Moreover, when there are many uncertain parameters the number of possible scenarios can grow too quickly. Nevertheless, many problems have been successfully modeled using this framework [4–7].

Robust Optimization, unlike Stochastic Optimization, does not rely on a stochastic model for the uncertainty. Rather, the perturbations are assumed to be restricted to given uncertainty sets. The RO approach then generates a solution that is optimal against the worst possible realization of the uncertainty sets. This makes this paradigm especially suitable when significant violations cannot be tolerated even with low probability, such as in engineering structures (one intuitive example is the case of bridge construction, as discussed in [8])

Robust Optimization has been successfully applied in several fields, such as portfolio optimization [9–11], logistics and supply chain management [12–14], healthcare [15], marketing [16], scheduling [17], Statistics and machine learning [18–21] and, more recently, Data-driven Robust Optimization [22,23]. Some of these applications are covered in the surveys [24,25].

Since the RO framework will be at the core of this project, we will now define it formally and in more details.

1.2 Robust Optimization

The goal of Robust Optimization is to find optimal solutions that are feasible for *all* possible realizations of the given uncertainty sets. One way to think about this idea is that the paradigm finds the best possible solutions while being pessimistic about nature (i.e., assuming it will behave in the worst possible way).

This is typically done by solving the *robust counterpart*, a reformulation of the original problem that takes into account all possible realizations of the uncertainty set.

More formally, consider the following convex program:

$$\begin{array}{ll} \underset{x}{\text{minimize}} & f_0(x) \\ \text{subject to} & f_i(x, \bar{u}_i) \leq 0, \; i = 1, \ldots, m \\ & x \in \mathcal{X} \end{array}$$

where $x \in \mathbb{R}^n$ is a vector of decision variables, f_i are convex functions, $\mathcal{X}$ is a convex set and the parameters $\bar{u}_i \in \mathbb{R}^K$ are fixed. From here on, we will refer to this convex problem, with fixed parameters $\bar{u}_i$, as the *nominal* problem.

Its robust counterpart is given by:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f_0(x) \\ \text{subject to} \quad & f_i(x, u_i) \le 0,\ \forall u_i \in \mathcal{U}_i,\ i = 1, \dots, m \\ & x \in \mathcal{X} \end{aligned} \tag{1-1}$$

where the parameters $u_i \in \mathbb{R}^K$ are no longer fixed, but constrained to be in the uncertainty sets $\mathcal{U}_i \subseteq \mathbb{R}^K$.

It is known that assuming the objective function to be linear and certain can be done without loss of generality. This is because, if the objective function is in fact subject to uncertain parameters $u_0 \in \mathcal{U}_0$, formulation (1-1) can be rewritten with an auxiliary variable t:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & t \\ \text{subject to} \quad & f_0(x, u_0) - t \le 0\ \forall u_0 \in \mathcal{U}_0, \\ & f_i(x, u_i) \le 0,\ \forall u_i \in \mathcal{U}_i,\ i = 1, \dots, m \\ & x \in \mathcal{X} \end{aligned} \tag{1-2}$$

We can also assume without loss of generality that the uncertainty set has the form of a cartesian product: $\mathcal{U} = \mathcal{U}_0 \times \mathcal{U}_1 \times \cdots \times \mathcal{U}_m$ (see [26] for further discussion on this point, and a proof).

Is it not clear, a priori, when problem (1-2) is tractable. In fact, as noted in [25], in its current form it seems intractable if the uncertainty sets are continuous, since it contains infinitely many constraints. The classical approach is to reformulate problem (1-2) using the idea that in order for the solution to be feasible for any realization u_i of the uncertainty $\mathcal{U}_i$, it suffices to be feasible for its worst case. The tractability of this reformulated problem largely depends on the shape of the uncertainty set chosen, as well as on the complexity of the nominal problem, as we will see in more details in Section 1.2.1.

1.2.1 Choice of the uncertainty set and related works

This section will provide a brief overview regarding some of the classical works on Robust Optimization, and explore the relationship between the tractability of the robust counterpart of an optimization problem and the chosen class of uncertainty sets. For a more detailed description of the most commonly used uncertainty sets, see e.g., the study made by Li and Floudas [27].

The first work to explicitly consider uncertain hard constraints in Mathematical Programming was made by Soyster in 1973 [28]. It explores Robust Linear Optimization, when the nominal problem is itself a linear program, with constraints expressed as $Ax \le b, x \ge 0$. Soyster considers a specific case in which the *columns* of matrix A are uncertain and known to belong to given uncertainty sets $K_j \subseteq \mathbb{R}^K$. Defining a_j as the $j-$th column of matrix A, these linear constraints can then be written as:

$$\sum_{j=1}^{n} x_j a_j \leq b \, \forall (a_j \in K_j, j \in [n]), x \geq 0$$

Which is, in fact, equivalent to the linear system:

$$A^* x \leq b, x \geq 0; \;\; \text{where } A^*_{ij} = \sup_{a_j \in K_j} (a_j)_i \tag{1-3}$$

As discussed in [29], while this reformulation has the benefit of resulting in an LP, it is way too conservative: by considering *column-wise* uncertainty, its robust counterpart has a matrix A^* in which each entry is, simultaneously, as bad as it possibly could be. As a result, this reformulation gives up too much in optimality in order to ensure robustness for cases that are extremely unlikely in real-life situations.

Decades later, a successful attempt to reduce such conservatism, by tackling the more general case of *row-wise* uncertainty, appeared in a series of papers from Ben-Tal and Nemirovski [1, 8, 26, 29, 30] and, simultaneously and independently, El-Ghaoui et al. [19, 31]. In some of these works, the nominal problem is no longer assumed to be linear, only convex. Their focus is mainly on the case of *ellipsoidal* uncertainties, i.e., the uncertainty sets are assumed to be ellipsoids. The reason to use such uncertainty sets is not only that it is a mathematically convenient way of approximating more complex sets: in [29], Ben-Tal and Nemirovski propose an interesting motivation for ellipsoidal uncertainties based on statistical considerations, using a robust portfolio optimization problem to illustrate their point. A separate study by Dokka and Goerigk [32] performed an experimental comparison of several uncertainty sets on the robust shortest path problem with real-life instances, and concluded that the ellipsoidal uncertainty set is among the best options in terms of trade-off between robustness and optimality.

One problem with these approaches is the known fact that, when using ellipsoidal uncertainties, the robust counterpart problem belongs, in general, to a different, harder class of optimization problems than its nominal version. We will present two practical cases on this matter in Chapter 7, in which we perform computational experiments. As examples, it is known that:

– The robust counterpart of an LP (linear program) with ellipsoidal uncertainty sets is an SOCP (second-order cone program)

– The robust counterpart of a convex QCQP (quadratically constrained quadratic program) with a single ellipsoidal uncertainty set is an SDP (semidefinite program) [24, 26]

– The robust counterpart of an SDP with ellipsoidal uncertainty sets is NP-Hard [26, 33]

In general, the situation is even worse: even a convex QCQP with polyhedral uncertainty is already NP-Hard.

In a celebrated work, Bertsimas and Sim [13] proposed a method that, considering polyhedral uncertainty, reformulates a linear program into a robust counterpart that is still itself linear. A parameter Γ is introduced and can then be used to calibrate the trade-off between optimality and conservatism. While this work is still very important for the RO field (since it provides an efficient way to solve robust linear programs and even robust discrete [34] programs with a reasonable uncertainty set that can be tuned by the practitioner), it still does not handle the more general case we are interested here, where the nominal problem must only be convex, not necessarily linear, and where the uncertainty sets do not have to be polyhedra.

The following situation is presented thus far: the complexity of the robust counterpart is heavily dependent on the uncertainty sets chosen, and, in many practical cases, uncertainty sets that would make sense in real-life applications have to be avoided in order to ensure tractability. Meanwhile, recent connections are being made between Robust Optimization and fields with heavy usage of big data such as Machine Learning and Statistical Learning (see, e.g., [20,21,35], and, especially, [23], a recent work in Data Driven Robust Optimization in which Bertsimas et al. argue, e.g., in Chapters 1 and 2.1, that cutting-set or online optimization methods often have better performance than attempts to solve the problem directly). This motivates the need for iterative methods able to (approximately) solve robust problems without needing to work with its robust counterpart. By approximately solving we mean finding solutions that are guaranteed to not violate any of the robust constraints by more than a tolerance ϵ given as input. These methods will be the core of Chapter 4 (where some of these existing ideas will be described in depth) and the rest of the chapters from then on.

As evidence for the success of these iterative approaches, Bertsimas et al. [36] show that iterative approaches often perform better than reformulation even for polyhedral uncertainty sets. In their Practical Guide to Robust Optimization, Gorissen et al. [25] also advocate for the use of these methods whenever a reformulation into a tractable robust counterpart is not readily possible.

1.3 Our Contributions

We now briefly describe our two main contributions.

The first is mostly a theoretical one. As we will see in Chapter 4, Ben-Tal et al. show in [33] how to approximately solve robust problems by repeatedly solving the nominal problem, with m uncertain constraints, while updating the uncertainty parameters with a first-order method. In Chapter 5, we present the Single-row based RO, an algorithm that, in each iteration, needs to solve an easier version of the nominal problem, in which all the uncertain constraints are aggregated into one. In Theorem 1 we prove that this algorithm does not need much more iterations than the one presented in [33] in order to converge, and has the advantage that each iteration can be much less costly. Our algorithm uses ideas from the Multiplicative Weights Update framework [37], to be presented on Sections 3.2.4 and 3.3.

The second contribution is an expansion on the ideas explored in previous chapters to create an algorithm that, instead of focusing on theoretical guarantees, aims to achieve good practical performance. It uses ideas from the Single-row based RO (our own theoretical algorithm) and from Mutapcic and Boyd's [38], and is able to achieve interesting results by iteratively solving a modified version of the nominal problem that starts with only one constraint and slowly grows in size until convergence is achieved. We then implemented several other algorithms presented in the literature [33, 38, 39], as well as the reformulation into robust counterpart approach, and show that our procedure can be competitive in practice, especially when the nominal problem is itself not easy to solve and when its constraints have varying levels of slackness.

1.4 Organization

This work is organized as follows. Chapters 2 and 3 are two support chapters, and present the mathematical tools that the reader must be comfortable with in order to understand the models developed in later chapters. Chapter 2 focuses on convexity, defines convex functions and gives examples of convex sets, which is useful to better understand the usage of uncertainty sets in Robust Optimization. Chapter 3 presents the Online Convex Optimization framework and some of its most famous algorithms.

In Chapter 4, we present three of the most notable works in the literature of iterative methods to approximately solve Robust Optimization.

Chapters 5 and 6 present our contributions, as discussed in Section 1.3. Finally, in Chapter 7 we perform computational experiments with robust LPs and robust QCQPs.

2
Preliminaries I: Convexity

In this chapter, we will start by defining convexity itself, and will give several examples of convex sets. Finally, we will define convex functions. This chapter can be skipped by the reader already familiar with theoretical concepts in convexity, but is highly recommended otherwise.

2.1
Convex sets

A set S is convex if all the points of a line segment between any two points of S lie in S. More formally, if x, $y \in S$, then:

$$\theta x + (1-\theta)y \in S \; \forall \theta \in [0,1] \tag{2-1}$$

As natural examples, we note that any affine set is also convex. Therefore, the empty set, a single point and the whole space $\mathbb{R}^n$ are convex sets.

We will now see two examples of convex sets, which will be useful in future chapters. Proofs for each case, as well as more detailed examples, can be seen in [40].

2.1.1
Euclidean balls and ellipsoids

Let r be the radius of a ball and $x_c \in \mathbb{R}^n$ its center. An Euclidean ball in $\mathbb{R}^n$ can be represented in any of the following ways:

$$B(x_c, r) = \{x : ||x - x_c||_2 \leq r\}$$

$$B(x_c, r) = \{x : (x - x_c)^T (x - x_c) \leq r^2\}$$

$$B(x_c, r) = \{x_c + ru : ||u||_2 \leq 1\}$$

Related to the Euclidean balls is the family of ellipsoids:

$$\mathcal{E} = \{x : (x - x_c)^T P^{-1} (x - x_c) \leq 1\}$$

where P is a symmetric and positive definite scaling matrix, that serves to determine how the ellipsoid extends in each direction of x_c. The lengths of the semi-axes of $\mathcal{E}$ are given by $\sqrt{\lambda_i}$, with λ_i being the eigenvalues of P.

Another way to represent an ellipsoid is the following:

$$\mathcal{E} = \{x_c + Au : ||u||_2 \leq 1\}$$

where A is square and nonsingular.

Finally, we note that the ellipsoid is a generalization of the euclidean ball. Indeed, a ball is an ellipsoid with $P = r^2 I$

2.1.2 Norm balls and norm cones

Let $||\cdot||$ be any norm on $\mathbb{R}^n$. It can in fact be shown that any norm ball (and not just the Euclidean ball), given by the set:

$$\{x : ||x - x_c|| \leq r\}$$

is convex.

Further, it also holds that the norm cone associated with the norm $||\cdot||$, given by the set:

$$\{(x,t) : ||x|| \leq t\} \leq \mathbb{R}^{n+1}$$

is convex.

As noted in [40], the second-order cone, which is frequently referred to in the convex optimization field and which we will explicitly make use of in future chapters, is simply the norm cone for the Euclidean norm:

$$\{(x,t) \in \mathbb{R}^{n+1} : ||x||_2 \leq t\}$$

2.2 Convex functions

Let S be a convex set, as defined in (2-1). A function $f : S \to R$ is convex if for any $x, y \in S$, the following holds:

$$\forall\, \theta \in [0,1],\ f((1-\theta)x + \theta y) \leq (1-\theta)f(x) + \theta f(y) \tag{2-2}$$

It is worth noting that when f is convex, $-f$ is concave. Finally, for affine functions, the inequality in (2-2) naturally becomes an equality, and hence all affine functions are convex and concave.

Convex functions are especially important in the optimization field because a local minimum of a convex function is also a global minimum (and analogously, a local maximum of a concave function is a global maximum).

3
Preliminaries II: Online Convex Optimization

In this chapter, we introduce the Online Convex Optimization framework. Then, we will go into classic algorithms to solve this problem, namely Follow-the-Leader and Online Gradient Descent. We also present the Multiplicative Weights Update, showing how it can be useful to efficiently solve a special case of the OCO problem.

3.1
Online Convex Optimization

We now discuss the Online Convex Optimization (OCO) framework, first introduced by Zinkevich in [41]. It can be seen as a generalization of the problem of prediction with expert advice [42, 43].

In online convex optimization, an online player chooses, at each step, a point x^t in a set $\mathcal{X}$, which is assumed to be non-empty, bounded, closed and also convex. At the time of such decision, the associated outcome is unknown to the player. After x^t is chosen, a concave gain function $f^t : \mathcal{X} \to \mathbb{R}$ is revealed, and thus the gain incurred to the player for his decision is $f^t(x^t)$.

The goal of the player is to maximize the total gain, given by:

$$\sum_{t=1}^{T} f^t(x^t)$$

In this model, f^t can even be chosen by an adversarial. It is important to note, however, that in this model all information about f^t is revealed after the decision, which means that the player also sees the gains of other possible decisions he could have taken (the model where only the value $f^t(x^t)$ is revealed is the bandit model, see, e.g., the recent book [44]).

Let T be the total number of game iterations. Then, the total gain of the choices made by the player is given by $\sum_{t=1}^{T} f^t(x^t)$, and the total gain of a given static solution $x \in \mathcal{X}$ is $\sum_{t=1}^{T} f^t(x)$. In order to be able to measure the performance of an algorithm in this setting, the decisions made by the player are compared to the best possible static decision, giving rise to the concept of regret $\mathcal{R}$:

$$\mathcal{R}_T = \max_{x \in \mathcal{X}} \sum_{t=1}^{T} f^t(x) - \sum_{t=1}^{T} f^t(x^t) \tag{3-1}$$

The pseudocode in 1 describes the protocol of this game.

Algorithm 1 Online Convex Optimization

```
for $i = 1$ to $T$ do
    Player chooses $x^t \in \mathcal{X}$
    Player receives concave gain function $f^t : \mathcal{X} \to \mathbb{R}$
    Player suffers gain $f(x^t)$
end for
```

Intuitively, one would like to achieve a sublinear regret, i.e., $\mathcal{R} \in o(T)$, which implies that the algorithm is effectively learning.

We will now see classic algorithms for the OCO problem, and their guarantees on the regret.

3.2 Algorithms for Online Convex Optimization

3.2.1 Follow the Leader

The Follow the Leader (FTL) algorithm takes, at each step, what is perhaps the most natural, intuitive decision: it chooses the solution which maximizes the accumulated gain, considering all previous gain functions. Algorithm 2 summarizes this greedy approach.

Algorithm 2 Follow the Leader

```
$x^1$ chosen arbitrarily
for $i = 1$ to $T$ do
    $x^t = \mathrm{argmax}_{x \in \mathcal{X}} \sum_{s=1}^{t-1} f_s(x)$
end for
```

It is known that FTL's regret is upper bounded by the cumulative difference between the gains of x^t and x^{t+1}:

$$\mathcal{R}_T \leq \sum_{t=1}^{T} \left(f^t(x^t) - f^t(x^{t+1}) \right) \tag{3-2}$$

The proof is found in [45].

In some special cases, such as when the gain functions are strongly convex or when the gains are i.i.d. (independent and identically distributed), FTL is known to perform greatly, achieving a $\mathcal{O}(\log T)$ regret. [46] [47]

However, this is not the case in general. In fact, this algorithm is known to fail in a worst-case scenario. A common example used to demonstrate the main drawback of the Follow the Leader algorithm is the following:

Let $f^1(x) = \frac{x}{2}$, and $f^t(x) = \begin{cases} -x & \text{if } t \in \{2,4,6,8,\dots\} \\ x & \text{if } t \in \{3,5,7,9,\dots\} \end{cases}$

Suppose that we have a playing set $x \in [-1, 1]$. In this case, the FTL strategy will forever alternate between -1 and 1, and make a mistake at every step. In $t = 2$, FTL will play $x = 1$, because this solution maximizes $f^1(x) = \frac{x}{2}$ (the only gain function seen so far), and since $f^2(x) = -x$, it will receive a gain of -1. Then, in $t = 3$, it will play $x = -1$ in order to maximize the function $-\frac{x}{2}$, but will then observe gain function $f^3(x) = x$ and again receive gain of -1. The total gain, therefore, will be no bigger than $(-T + 1)$, while the total gain for the best static solution $(x = 0)$ will be zero. The regret is thus $\Omega(T)$.

As discussed in [45], the reason why FTL fails in this example is the lack of stability. Intuitively, the solution x^t shouldn't shift so drastically, from -1 to 1, when a single new gain function f^t is recognized by the player. As an example, from equation (3-2) we can see that if FTL is completely stable, i.e., $x^{t+1} = x^t$ always holds, the regret is zero. This also informally illustrates why FTL performs well when x^t is "close" to x^{t+1}, and hence this drawback is not explored. A natural attempt to fix this problem was to add a regularization term, which gave origin to an algorithm with vastly different guarantees, the Follow the Regularized Leader.

3.2.2 Follow the Regularized Leader

The Follow the Regularized Leader (FTRL) is a modification of the FTL algorithm that aims to generate more stable solutions over time. While the FTL chose solutions that maximized the cumulative gain on all previous rounds, the FTRL maximizes the same gain plus a regularization function evaluated on such solution, $R : \mathcal{X} \to \mathbb{R}$. Algorithm 3 illustrates this setting.

Algorithm 3 Follow the Regularized Leader

```
$x^1 = \text{argmin}_{x \in \mathcal{X}} R(x)$
for $i = 1$ to $T$ do
    $x^t = \text{argmax}_{x \in \mathcal{X}} \Big( \sum_{s=1}^{t-1} f_s(x) - R(x) \Big)$
end for
```

A natural question that remains is how to correctly choose the regularizer function R. As we will see, in order to achieve good theoretical guarantees on the regret, R should be strongly convex (for a formal definition and more on strong convexity, see e.g. [48, 49]). Apart from this requirement, intuitively there is a trade-off: it should be effective in preventing the solutions from varying too dramatically, but the regularization losses shouldn't grow too large as to dominate the gains of the decisions made by the player. In this context, we will now see two famous algorithms that arrive from the FTRL and their regularizers.

3.2.3 Online Gradient Ascent

The Online Gradient Ascent algorithm [41], as the name suggests, is an online version of the famous Gradient Ascent algorithm. It consists of, at each iteration, taking a step from the current point in the direction of the gradient of the current gain function. We note that its most notorious version is in fact the *Gradient Descent*, in which the goal is to minimize a cost function, and therefore the steps are taken in the *direction* against the gradient. We will define the Online Gradient Ascent because, in this work, we are mostly interested in using the OCO framework for *maximization*, but the results are analogous.

After a step in the direction of the current gradient is taken, the updated point could fall outside of the convex set, a projection operation may have to be performed in order to ensure feasibility. The projection step consists of finding the closest point in the convex set from the updated solution. In general, this is a Convex Optimization problem, and its efficiency largely depends on the shape of the convex set (see e.g. [40], Chapter 8.1).

Algorithm 4 illustrates this strategy. We denote $\Pi_{\mathcal{X}}(y)$ as the projection of y onto set $\mathcal{X}$, i.e.

$$\Pi_{\mathcal{X}} = \min_{x \in \mathcal{X}} ||y - x||$$

Algorithm 4 Online Gradient Ascent

```
$x^1 \in \mathcal{X}$, learning rate $\eta$
for $i = 1$ to $T$ do
    $y^{t+1} = x^t + \eta \nabla f_t(x^t)$
    $x^{t+1} = \Pi_{\mathcal{X}}(y^{t+1})$
end for
```

It should be noted that the algorithm as designed in [41] in fact supports an adaptive learning rate, instead of a fixed one. Works have been done focusing on how to exploit this flexibility to improve efficiency in some specific cases, such as strong convexity [50]. However, since the optimal asymptotic guarantees can already be achieved with a fixed learning rate, this will be our focus in this project.

In order to define which value of η should be used to achieve optimal guarantees, we first need to define two parameters. G, an upper bound on the ℓ_2 norm of the gradients of the gain functions:

$$G : \max_{t \in [T]} ||\nabla f_t(x^t)||_2 \leq G \tag{3-3}$$

and D, an upper bound on the ℓ_2 diameter of the convex set $\mathcal{X}$:

$$D : \max_{x,y \in \mathcal{X}} ||x - y||_2 \leq D \tag{3-4}$$

Both G and D can sometimes be calculated analytically, as we will show in an example in Section 4.3.

Given these parameters, it is known [41] that if we set $\eta = \frac{D}{G\sqrt{T}}$, the regret is guaranteed to have the following upper bound:

$$\mathcal{R}_T \leq GD\sqrt{T} \tag{3-5}$$

which is $o(T)$.

The Online Gradient Descent is directly connected to the FTRL paradigm: in fact, OGD is equivalent to the FTRL with regularization function $R(x) = \frac{1}{2\eta}||x||_2^2$. More details on this equivalence and the proof can be seen in [51].

3.2.4 Multiplicative Weights Update

The Multiplicative Weights Update (MWU) is a framework that unifies methods previously discovered in different fields, such as AdaBoost in Machine Learning [52], Plotkin et al.'s method to approximately solve packing and covering LPs fast [53] in Optimization, and Game Theory. A survey was published by Arora et al [37] discussing these applications and several others.

This framework is suitable for a setting that can be described as follows: at each time step, a decision maker is presented with m possible decisions, and must choose a decision from the set. After the decision is made, all the gains are revealed, and the player receives the gain of the decision chosen. Once again, the objective is to maximize the sum of gains over time.

As in the general OCO framework, the goal is to perform almost as well as the best possible fixed decision. To achieve this objective MWU maintains a probability distribution p^t over the set of decisions (so one can think that the action at time t is sampled from the distribution p^t). Intuitively, this probability distribution evolves over time so as to skew the distribution in favor of the decision that have been better so far.

The algorithm is described in [37] as follows:

Algorithm 5 *Multiplicative Weights Update (MWU)*

```
Step $\eta \leq \frac{1}{2}$
Initial weight of each decision, $w_i^1 = 1$
for $t = 1, 2, \ldots, T$ do
    Compute probability distribution over set of decisions:
        $p^t = \{w_1^t/\phi^t, \ldots, w_m^t/\phi^t\}$
    where $\phi^t = \sum_{i=1}^m w_i^t$
    Given probability distribution $p^t$, choose decision
    Observe gain of each decision: $m_i^t \in [-\rho, \rho] \ \ \forall i \in [m]$
    Increase gain of good decisions by updating their weight as follows:
        $w_i^{t+1} = w_i^t(1 + \eta m_i^t) \ \ \forall i \in [m]$
end for
```

The by now standard guarantee of MWU is the following.

Lemma 1 (Theorem 2.5 of [37]) *If for all $t \in [m]$ we have $m^t \in [-1,1]^m$ and $T \geq \ln m$, then the MWU algorithm (with $\eta = \sqrt{\frac{\ln m}{T}}$) satisfies the following for every $i \in [m]$:*

$$\sum_{t=1}^{T} \langle m^t, p^t \rangle \geq \sum_{t=1}^{T} m_i^t - 2\sqrt{T \ln m}. \tag{3-6}$$

Notice that if the gains are in the interval $[-\rho, \rho]$, we can simply run the MWU over the gains scaled by $\frac{1}{\rho}$ to obtain the following.

Corollary 1 *If for all $t \in [m]$ we have $m^t \in [-\rho, \rho]^m$ and $T \geq \ln m$, then running the MWU algorithm over the gain vectors $\frac{m^t}{\rho}$ produces p^t's that satisfy for every $i \in [m]$*

$$\sum_{t=1}^{T} \langle m^t, p^t \rangle \geq \sum_{t=1}^{T} m_i^t - 2\rho\sqrt{T \ln m}. \tag{3-7}$$

The parameter ρ is called the *width* of the problem. We note that if ρ, or a reasonable upper bound for it, cannot be known or estimated before running the algorithm, the so-called "doubling trick" can be used. This method consists of starting with a reasonably small guess for $\hat{\rho}$ and, whenever a gain larger than this estimate is observed, the estimate is doubled, i.e., $\hat{\rho} \leftarrow 2 \cdot \hat{\rho}$. The resulting regret is worse only by a constant multiplicative factor. More details on the doubling trick can be seen in the Section 2.3.1 of Shalev-Shwartz's book on OCO [45].

3.3 Application of the MWU framework in Convex Programming

We will now describe how the Multiplicative Weights Update framework described in Section 3.2.4 can be used to approximately solve a Convex Optimization Problem. From now on, we will refer to this algorithm as the *MWU-Based CP*. This method is inspired by Plotkin et al. [53] and is described in the survey made by Arora et al. [37]. It consists of running the MWU framework while, at each iteration, calling a feasibility oracle that aggregates the m constraints of the nominal proble into one.

Consider the following feasibility problem: Given a convex set $\mathcal{X}$ and convex functions f_i's, find a ε-feasible solution to

$$\begin{aligned} &f_i(x) \leq 0,\ i = 1, \dots, m \\ &x \in \mathcal{X}, \end{aligned} \tag{3-8}$$

or assert that no such feasible solution exists; by ε-feasible solution x we mean one where $f_i(x) \leq \varepsilon$ for all $i \in [m]$. We assume that for every $x \in \mathcal{X}$ and every $i \in [m]$, $f_i(x) \in [-\rho, \rho]$ for some parameter ρ (called the *width* of the instance).

For that we will apply the MWU framework, thinking of the constraints as experts. We assume the existence of an oracle that given a fixed distribution $\bar{p}$ over $[m]$, finds a point $\bar{x} \in \mathcal{X}$ satisfying

$$\sum_{i=1}^{m} \bar{p}_i f_i(\bar{x}) \leq 0, \tag{3-9}$$

or asserts that none exists (one can work with approximate oracles as well). The algorithm for approximate solving the convex program is the following: For each $t = 1, \dots, T$, we start with a distribution p^t and run the oracle with $\bar{p} = p^t$ to obtain a point $x^t \in \mathcal{X}$ feasible for (3-9) (if such exists); then compute the gain vector $m^t := (f_1(x^t), \dots, f_m(x^t))$ and feed it to the algorithm from Corollary 1 to update the distribution, obtaining p^{t+1}. At the end of the T iterations, the algorithm returns the average of the solutions $\frac{x^1+\dots+x^T}{T}$ (if in any iteration the oracle returns "infeasible", then the original system (3-8) is also infeasible).

The advantage of this algorithm is the following: We can think of $\mathcal{X}$ as a set of "easy" constraints. As an example, in the context of a minimization problem, if its optimal value is known to be OPT (or guessed, such as in a binary search setting), $\mathcal{X}$ could be the set of point with value at most OPT.

For such simple sets, it should be much faster to check feasibility as in (3-9) once than to solve (3-8) directly.

Using the guarantee from Corollary 1, we can bound the number of iterations T needed in order for this algorithm to return an ε-feasible solution to (3-8).

Lemma 2 *Suppose* (3-8) *is feasible. Then if* $T \geq \frac{4\rho^2 \ln m}{\varepsilon^2}$, *the solution returned by the algorithm described above is* ε*-feasible for it.*

Prova. By the definition of the oracle, and recalling the definition of the gain vectors, we have that for all $t \in [T]$ $\langle m^t, p^t \rangle \leq 0$. Then employing Corollary 1 we get that for every $i \in [m]$

$$0 \geq \sum_{t=1}^{T} \langle m^t, p^t \rangle \geq \sum_{t=1}^{T} m_i^t - 2\rho\sqrt{T \ln m} = \sum_{t=1}^{T} f_i(x^t) - 2\rho\sqrt{T \ln m} \geq \sum_{t=1}^{T} f_i(x^t) - T\varepsilon,$$

where the last equation uses the definition of T. Rearranging, dividing through by T, and using Jensen's inequality, we get that for every $i \in [m]$

$$f_i(\tfrac{x^1 + \ldots + x^T}{T}) \leq \frac{1}{T} \sum_{t=1}^{T} f_i(x^t) \leq \varepsilon.$$

This concludes the proof. ■

4
Iterative Methods for Robust Optimization

In Chapter 1 we mentioned the idea of using iterative methods to (approximately) solve the robust version of an optimization problem using only its nominal version, since solving the nominal problem is usually much easier than solving its robust problem directly.

We will now cover important previous works done on this topic. We focus in the case where the constraints are convex functions of the decision variables and concave functions of the uncertainty parameters.

4.1
Overview

Before describing each algorithm, we will provide a brief overview of their differences.

The first method we will study is a cutting-set method developed by Mutapcic and Boyd [38]. It starts by solving the nominal optimization problem, and then performs a maximization on the uncertainties of each constraint in order to find scenarios that violate the current solution. All these scenarios are *added* to the nominal problem as new constraints, and the process is repeated until a solution is found such that no new scenario violates the constraints by more than a tolerance ϵ. The problem solved to find x^t in each iteration is therefore an extension of the nominal problem, with more constraints than the original version.

Secondly, we study the Dual-Subgradient method by Ben-Tal et al. [33], which instead of running a full maximization (as defined in Equation (4-2)) on the noises at each iteration, updates the uncertainty parameters with a first-order procedure, such as the OGD we have seen in Section 3.2.3. Solutions x^t are found by solving an approximated version of the nominal problem, with m constraints and current values of the noises u_i^t. Ben-Tal et al. are able to provide guarantees of convergence of the averaged solution, $\bar{x} = \frac{1}{T}\sum_t x^t$.

The last method we will study from the literature is the Online First-Order method (OFO), developed recently by Ho-Nguyen and Kilinc-Karzan [39]. Their goal is to achieve the following improvement on the Dual-Subgradient method: in order to find a new solution x^t in each iteration, instead of solving an optimization problem with m uncertain constraints in each iteration, they propose to use first-order procedures, such as OGD, as is done to update the noises. They also provide guarantees for the convergence of the average solution. We note two possible drawbacks in this framework:

1. After a first-order procedure, as explained in Section 3.2.3, a projection is needed in order to ensure feasibility of the updated solution. If the set $\mathcal{X}$ does not have a very favourable structure, this projection could itself be costly. An example is if the set $\mathcal{X}$ has many linear constraints (that are not uncertain): in this case, the projection would consist of solving an optimization problem of minimizing a distance subject to several linear constraints.

2. Since a full optimization on x^t is never performed, this method requires knowledge of the optimal value of the robust problem (which we will call OPT) in advance. Since this is usually not the case, a binary search to find OPT is needed, as will be explained in Section 4.4.

Finally, we will present our algorithm, Single-row based RO, in Chapter 5. We show that we can adapt ideas from the MWU framework seen in Sections 3.2.4 and 3.3 and from the Dual-Subgradient method [33] to run a procedure similar to the Dual-Subgradient, but that in each iteration finds a solution x^t by solving a way smaller version of the nominal problem, with only one aggregated constraint that represents the m uncertain ones (as illustrated in Equation (1-row oracle)). We are still able to provide guarantees of convergence that are not much worse than these of [33], with the benefit that each iteration can be much less costly in our method.

Table 4.1 summarizes these ideas.

	Mutapcic's Cutting-Set [38]	Dual-Subgradient [33]	OFO [39]	Single-row based RO
Solution Update	Extended nominal problem (4-1)	Approximated nominal problem (4-3)	First-order method	Single-row nominal problem (1-row oracle)
Uncertainty Update	Full pessimization (4-2)	First-order method	First-order method	First-order method
Binary Search on OPT needed?	No	No	Yes	No

Table 4.1: Overview of the algorithms to be studied in Chapters 4 and 5

4.2 Cutting-set methods with pessimizing oracles

This method was introduced by Mutapcic and Boyd [38] in 2009. To introduce their idea, we first need to define the so-called sampled problem. First, we consider the nominal problem, and as discussed in Section 1.2, we can assume the objective to be certain and linear without loss of generality. Therefore, to simplify notation, we will describe the nominal problem considering this to be the case. The nominal problem is then written as follows:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & c^T x \\ \text{subject to} \quad & f_i(x, \bar{u}_i) \leq 0, \; i = 1, \ldots, m \end{aligned}$$

where $\bar{u}_i$ is the nominal value of the parameter u_i.

We are interested in the case in which each u_i is uncertain and belongs to the set $\mathcal{U}$, with $\bar{u}_i \in \mathcal{U}_i$. Thus, the robust version of the problem is the following:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & c^T x \\ \text{subject to} \quad & \max_{u_i \in \mathcal{U}_i} f_i(x, u_i) \leq 0, \; i = 1, \ldots, m \end{aligned}$$

Now, consider a case in which there are subsets $\hat{\mathcal{U}}_i \subseteq \mathcal{U}_i$, defined as:

$$\hat{\mathcal{U}}_i = \{u_i^1, u_i^2, \ldots, u_i^{K_i}\}$$

which means that, instead of the whole set $\mathcal{U}$, only K_i scenarios are being considered for constraint i. Assume that $u_i^1 = \bar{u}_i$, so that each subset is guaranteed to contain the nominal value $\bar{u}_i$.

Then, the so-called sampled robust problem, which is an extension of the nominal problem, is defined as:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & c^T x \\ \text{subject to} \quad & f_i(x, u_{i,j}) \leq 0, \; i = 1, \ldots, m, \; j = 1, \ldots, K_i \end{aligned} \tag{4-1}$$

which is a convex optimization problem with $K = K_1 + \cdots + K_m$ constraints.

Mutapcic and Boyd [38] show that, under mild technical conditions (e.g., Slater's condition), by solving the sampled problem we automatically have both a lower and an upper bound on the optimal value for the robust problem. Intuitively, the gap decreases with the quality of the sampled solution, which means that adding more scenarios (constraints) to the sampled problem leads to increasingly better bounds on the robust optimal value.

With these concepts, their algorithm can now be introduced. It consists of solving a sequence of sampled problems while increasing the number of scenarios. New scenarios are found by a *pessimizing oracle*, an oracle that finds the worst possible scenario of a constraint for a given point x. More formally, for a given constraint i and for a fixed x, a *pessimizing oracle* solves:

$$\begin{aligned} \underset{u}{\text{maximize}} \quad & f_i(x,u) \\ \text{subject to} \quad & u \in \mathcal{U}_i \end{aligned} \tag{4-2}$$

Mutapcic and Boyd [38] note that this pessimization can be exact or approximate. In the approximate version, the resulting u would still be in the set $\mathcal{U}_i$, but not guaranteed to maximize $f_i(x,u)$. We highlight that, unless it is prohibitively costly, solving (4-2) exactly in each iteration can be convenient, because this creates a natural stopping criteria for the algorithm: whenever a solution x^t is found such that even a maximization on the uncertain parameters for each constraint is not able to find violations larger than the tolerance ϵ by the solution x^t, then x^t is robust $\epsilon-$feasible by definition.

From now on, we will call u_i^* the solution of problem (4-2).

Algorithm 6 Basic cutting-set method

```
Tolerance $\epsilon$
Initial sampled sets $\hat{\mathcal{U}}_i = \{\bar{u}_i\},\ i = 1,\dots,m$
while $\max_i \max_{u_i \in \mathcal{U}_i} f_i(x^t, u_i) > \epsilon$ do
    Solve sampled problem (4-1) with current sampled sets
    Let $x^t$ be the returned solution
    for $i = 1,\dots,m$ do
        Solve pessimizing oracle (4-2) with fixed $x^t$ to evaluate $u_i^*$ - exactly
or approximately
        if $f_i(x^t, u_i^*) > 0$ then Add $u_i^*$ to the sampled set $\hat{\mathcal{U}}_i$
        end if
    end for
end while
Return $x^t$
```

This method has been shown to perform very well in practice in many cases.

However, one drawback of the algorithm is the lack of good guarantees for such performances. The authors show, using ideas from Kelley et al. [54], an upper bound on the total number of constraints to be added that is *exponential* in n. Therefore, the sampled problem being solved in each iteration (4-1) can theoretically grow to be too large. This also illustrates why, as discussed by Mutapcic and Boyd in the Introduction of [38], this method is especially interesting when the nominal problem and the sampled problem can be efficiently solved. When this is not the case, each iteration could become too costly, especially if too many constraints have been added.

The need to run the pessimizing oracle (4-2) for every constraint in each iteration can also theoretically be too costly if such procedure cannot be done efficiently.

These facts motivated the search for other algorithms that could ideally have better guarantees without losing good performance on “easy” data.

4.3 Dual-Subgradient method

In the aforementioned context of trying to create iterative methods to approximately solve robust optimization problems with better guarantees, Ben-Tal et al. [33] published in 2014 an important work that improved some aspects of previous algorithms. In the paper, they deal with two cases: the first is when the constraints are concave functions of the noise u, and the uncertainty sets $\mathcal{U}_i$ are required to be convex, and the second case drops the requirement for the uncertainty set to be convex, but instead needs the constraints to be linear functions in u. We will focus on the former, with convex uncertainty sets, since it is more in line with the other algorithms presented here.

The idea of the method stems from the following question: “is it possible to approximately solve a robust optimization problem using only an algorithm for the original, nominal formulation?”. Again, the goal is to avoid the need of converting the problem into its robust counterpart, which in general can be much harder.

As usual, we will consider the objective to be certain and linear without loss of generality, to simplify notation. Ben-Tal et al.’s approach relies primarily on a nominal feasibility oracle, which approximately solves the nominal feasibility problem for a fixed noises $u_i \in \mathcal{U}_i$, returning an $\epsilon-$feasible solution x or declares the robust problem to be infeasible.

Formally, it receives fixed noises $u_1 \in \mathcal{U}_1, u_2 \in \mathcal{U}_2, \ldots, u_m \in \mathcal{U}_m$ for each constraint, and a tolerance ϵ, and solves the following feasibility problem:

$$\begin{aligned} &\exists ? x \in \mathcal{X} \\ &\text{subject to} \quad f_i(x, u_i) \leq \epsilon, \; i = 1, \ldots, m \end{aligned} \tag{4-3}$$

or, if it doesn’t exist, correctly concludes that the robust problem is infeasible.

We note that this oracle is, in general, no easier than solving the nominal version of the optimization problem. Therefore, the oracle itself could be costly. For this reason, the authors claim that the method is specially good if the nominal problem has a structure that could be exploited (and that would be lost in its robust counterpart version), such as the case of network flow problems.

The algorithm proposed uses this oracle to find, at each time step t, a solution x^t. The noises u_i^t, on the other hand, are updated via the Online Convex Optimization framework, more specifically using Online Gradient Descent (in its maximization version), seen in more details in Section 3.2.3. As we have seen, the OGD provides guarantees that depend on the diameter D, a parameter of the uncertainty sets such that $D \geq \max_{u,v \in \mathcal{U}_i} ||u - v||_2$ for all $i \in [m]$, and on G, a constant such that $||\nabla_u f_i(x, u)||_2 \leq G$ for all $x \in \mathcal{X}$, $u_i \in \mathcal{U}_i$ and $i \in [m]$. These parameters are not always easy to calculate, especially G. If it is not possible to calculate them efficiently, a doubling trick could be used, as mentioned for ρ in Section 3.2.4. However, we will now provide an **example**, as in [33], of a case in which it is possible to calculate them, for illustration purposes.

Suppose that a linear nominal problem has only one constraint of the form:

$$a^T x \leq b$$

Considering ellipsoidal uncertainty, its robust version can be written as:

$$(a + Pu)^T x \leq b$$

where $P \in \mathbb{R}^{n \times K}$ is a scaling matrix and $||u||_2 \leq 1$. Further assume a bound on the norm of x, $||x||_2 \leq 1$.

If we denote $f_i(x, u) = (a + Pu)x - b$, it is clear that $\nabla_u f_i(x, u) = P^T x$. Therefore:

$$||\nabla_u f_i(x, u)||_2^2 = x^T P P^T x \leq ||P||_F^2 \cdot ||x||_2 \leq ||P||_F^2$$

where $||P||_F^2$ represents the maximal magnitude of the noise.

Then, by definition, we can set $G = ||P||_F$. Moreover, due to the fact that $||u||_2 \leq 1$, the diameter is simply $D = 2$.

After this simple example of how the input parameters could be calculated, we can present their algorithm. For simplicity of notation, we assume, as do the authors, that D and G are the same for every uncertain constraint, i.e., $D_i = D$ and $G_i = G$ $\forall i = 1, \ldots, m$, which could be easily relaxed. The setting, called Dual-Subgradient approach, is then described as follows:

Algorithm 7 Dual-Subgradient Robust Optimization

```
Tolerance $\epsilon$
Parameters $D$, $G$
Set $T = \left\lceil \frac{G^2 D^2}{\epsilon^2} \right\rceil$ and $\eta = \frac{D}{G\sqrt{T}}$
Initialize $\{u_1^0, \dots, u_m^0\}$
for $t = 1, \dots, T$ do
    for $i = 1, \dots, m$ do
        Update each $u_i^t$ via OGD: $u_i^t = \Pi_{\mathcal{U}}\left(u_i^{t-1} + \eta \nabla_u f_i(x^{t-1}, u_i^{t-1})\right)$
    end for
    Set $x^t$ as the solution of the nominal feasibility oracle with the current noises $(u_1^t, \dots, u_m^t)$
    if Oracle declared infeasibility then
        return Robust problem is infeasible
    end if
end for
return $\bar{x} = \frac{1}{T}\sum_{t=1}^{T} x^t$
```

Lemma 3 (Theorem 3 of [33]) *After running $T = \left\lceil \frac{G^2 D^2}{\epsilon^2} \right\rceil$ of Algorithm 7, it is guaranteed to either return a $2\epsilon-$approximate solution to the robust problem or correctly conclude that it is infeasible.*

Prova. First, suppose that Algorithm 7 returns that the robust problem is infeasible. By the definition of the problem solved in each iteration, given in Equation (4-3), this means that for some $t \in [T]$, $u_i^t \in \mathcal{U}_i$, there does not exist $x \in \mathcal{X}$ such that:

$$f_i(x, u_i^t) \leq 0$$

which implies that the robust problem is not feasible.

Now, suppose that a solution $\bar{x}$ is returned. By the definition of the oracle, we know that for all $t \in [T]$ and $i \in [m]$, the following holds:

$$f_i(x^t, u_i^t) \leq \epsilon$$

Therefore,

$$\forall i \in [m], \frac{1}{T}\sum_{t=1}^{T} f_i(x^t, u_i^t) \leq \epsilon \tag{4-4}$$

From the regret guarantees of the OGD seen in Equatoin (3-5), we have:

$$\forall i \in [m], \max_{u_i \in \mathcal{U}_i} \frac{1}{T}\sum_{t=1}^{T} f_i(x^t, u_i) - \frac{1}{T}\sum_{t=1}^{T} f_i(x^t, u_i^t) \leq \frac{GD}{\sqrt{T}} \leq \epsilon \tag{4-5}$$

Combining (4-4) and (4-5), it follows:

$$\epsilon \geq \frac{1}{T}\sum_{t=1}^{T} f_i(x^t, u_i^t) \geq \max_{u_i \in \mathcal{U}_i} \frac{1}{T}\sum_{t=1}^{T} f_i(x^t, u_i) - \epsilon$$

From Jensen's Inequality:

$$\max_{u_i \in \mathcal{U}_i} \frac{1}{T}\sum_{t=1}^{T} f_i(x^t, u_i) - \epsilon \geq \max_{u_i \in \mathcal{U}_i} f_i(\bar{x}, u_i) - \epsilon$$

Therefore,

$$f_i(\bar{x}, u_i) \leq 2\epsilon \ \forall u_i \in \mathcal{U}_i \ \forall i \in [m]$$

which concludes the proof. ■

One main advantage of this algorithm, in comparison to Mutapcic and Boyd's cutting-set approach, is its worst-case guarantee on the number of iterations: due to the known guarantees of the Online Gradient Descent setting, Algorithm 7 will terminate after at most $\mathcal{O}(\frac{D^2G^2}{\epsilon^2})$ rounds. Even though this quantity could potentially be large, it is typically much better than the exponential guarantee available for Algorithm 6.

Another theoretical advantage of Algorithm 7 is the fact that it does not rely on a *pessimizing oracle*, as defined in equation (4-2), for the noises. While x^t is still found by solving the nominal problem (although not its sampled, expanded version), the noises u_i^t are computed using the faster OGD scheme.

On a more practical observation, however, we note that we would ideally like to evaluate the solution at every time step, or every few time steps, to check for robust feasibility (and hence return a valid solution before completing all T rounds). Since each of these checks consists precisely of running pessimizing oracles on the noise of each constraint, the advantage of not relying on such pessimization for the noise could in practice be smaller than it seems, depending on how often the practitioner wants to check for robust feasibility.

Another important remark is the following: the algorithm as designed in 7, relying on nominal *feasibility* oracles, would in fact require a binary search on the optimal value as an outside process, multiplying the performance guarantees by an extra $\mathcal{O}(\log(1/\epsilon))$. However, Ho-Nguyen and Kilinc-Karzan [39] observe in Remark 4.3 that by changing the feasibility oracle to an optimization oracle, i.e. ensuring that the oracle returns an ϵ-optimal instead of simply an ϵ-feasible solution, the average solution $\bar{x}$ is guaranteed to be robust ϵ-optimal and therefore no binary search is needed.

4.4

Online First-Order Framework

Ho-Nguyen and Kilinc-Karzan [39] explored the previous works discussed in Sections 4.2 and 4.3 with the following observations: even though Ben-Tal et al. [33] theoretically improved the noise update by substituting the need for a worst-case pessimization oracle on each constraint by an online learning procedure, both the Dual-Subgradient [38] and Cutting-set [33] methods still rely on solution oracles that could be too costly. Ho-Nguyen and Kilinc-Karzan note that while the nominal problem is typically much easier than its robust counterpart, depending on repeated calls can be prohibitive in the high-dimensional setting.

Ho-Nguyen and Kilinc-Karzan propose, as do Abernethy et al. [55], to view approaches such as the Dual-Subgradient method in a unified framework, which can be described in terms of Game Theory. The two iterative processes running simultaneously to generate and update solutions and noises can be seen as a dynamic game in which, in each round, Player 1 chooses a solution x^t to the current scenario u^t and Player 2 chooses a realization of the scenario u^{t+1}. They discuss that in the Dual-Subgradient [33], Player 1 considers the current noise, while Player 2 update the noise by minimizing the regret associated with past solutions.

The first goal in [39] is, therefore, to describe a general framework that includes [33] as a special case. We will now present this meta-template, but simplifying its notation.

First, we define:

$$\epsilon_u(\{x_t, u_t\}_{t=1}^T) = \max_{i\in[m]} \sup_{u^i\in\mathcal{U}^i} \frac{1}{T}\sum_{t=1}^T f^i(x_t, u^i) - \max_{i\in[m]} \frac{1}{T}\sum_{t=1}^T f^i(x_t, u_t^i) \leq \max_{i\in[m]} \mathcal{R}_i(T) \tag{4-6}$$

$$\epsilon_x(\{x_t, u_t\}_{t=1}^T) = \frac{1}{T}\sum_{t=1}^T \max_{i\in[m]} f^i(x_t, u_t^i) - \inf_{x\in\mathcal{X}} \frac{1}{T}\sum_{t=1}^T f^i(x, u_t^i) \leq \mathcal{R}_x(T) \tag{4-7}$$

Ho-Nguyen and Kilinc-Karzan [39] prove that we only need to bound the sum of the two terms, $\epsilon_u(\{x_t, u_t\}_{t=1}^T) + \epsilon_x(\{x_t, u_t\}_{t=1}^T) \leq \epsilon$ in order to obtain an $\epsilon-$feasible solution for the robust problem.

With this result, their framework is now introduced in Algorithm 8.

Algorithm 8 Ho-Nguyen and Kilinc-Karzan's framework for Iterative Robust Optimization

```
Tolerance $\epsilon$
Update algorithms $\mathcal{A}_i \; \forall \; i \in [m]$ and $\mathcal{A}_x$
Set $T$ large enough so that $max_{i\in[m]}\mathcal{R}_i(T) + \mathcal{R}_x(T) \leq \epsilon$
Initialize $u_1^i = \mathcal{A}_i(\{\}) \; \forall \; i \in [m]$ and $x_1 = \mathcal{A}_x(\{\})$
for $t = 2, \dots, T$ do
    for $i = 1, \dots, m$ do
        Compute $u_i^= \mathcal{A}_i(\{x_s, u_s\}_{s=1}^{t-1} \; \in \mathcal{U}^i$, where $\mathcal{A}_i$ can be an OCO procedure.
    end for
    Compute $x^t = \mathcal{A}_x(\{x_s, u_s\}_{s=1}^{t-1}) \; \in \mathcal{X}$, where $\mathcal{A}_x$ can be an OCO procedure.
    Obtain upper bounds $\hat{\epsilon}_u \geq \epsilon_u(\{x_s, u_s\}_{s=1}^{t})$ and $\hat{\epsilon}_x \geq \epsilon_x(\{x_s, u_s\}_{s=1}^{t})$, as explained in more details in [39], Section 4.
    Compute $\kappa_u^t = \hat{\epsilon}_u/\epsilon$ and $\hat{\epsilon}_x/\epsilon$
    if $\kappa_u^t + \kappa_x^t \leq 1$ then
        Set $v^t = \max_{i\in[m]} \frac{1}{t}\sum_{s=1}^{t} f^i(x^s, u_i^s)$
        Set $\tau^t = 1 - \kappa_x^t$
        if $v^t > (1-\tau^t)\epsilon$ then
            return Robust problem is infeasible
        end if
        if $v^t \leq (1-\tau^t)\epsilon$ then
            return $\bar{x}^t = \frac{1}{T}\sum_{s=1}^{t} x_s$, robust $\epsilon$-feasible solution.
        end if
    end if
end for
```

Ho-Nguyen and Kilinc-Karzan show that with the above meta-template, Ben-Tal et al.'s algorithm [33] can be directly seen as its special case, by defining the algorithms $\mathcal{A}_i$ as some variation of the Online Gradient Descent and $\mathcal{A}_x$ as an algorithm that finds x^t with a nominal feasibility oracle.

On the other hand, Mutapcic et al.'s method [38] can't be expressed in terms of the framework described in Algorithm 8, due to its cutting-set nature of adding constraints only when violations occur.

Finally, they propose their final algorithm, called *Online First-Order* method (OFO), in which both x^t and the noises $u_i^t \; \forall i \in [m]$ are chosen using first-order procedures such as OGD. If the problem has more than one constraint, the loss function for the first-order procedure to update x^t is given by the constraint with worst error in time t, as shown in (4-7).

Since x^t is updated by an OGD, a projection onto the set $\mathcal{X}$ must be performed after the update, which can be a non-trivial task unless $\mathcal{X}$ has a very favourable structure. This formulation also requires knowledge of the optimal value of the robust problem, OPT, in advance. If it is not known, a binary search must be performed. The The way to use binary search would be as follows: we start with a guess $O\hat{P}T$ and run the OFO algorithm. If it successfully converges, our guess of OPT decreases, since we are working with a minimization problem. If it returns infeasible, the guess $O\hat{P}T$ decreases. This is repeated until the binary search algorithm stops.

If the noises and the solutions are updated via an OGD procedure and if the projection steps can be performed easily, this algorithm takes $\mathcal{O}(\log(1/\epsilon) \cdot 1/\epsilon^2)$, where the $\mathcal{O}(\log(1/\epsilon))$ refers to the binary search procedure.

On a final note on this algorithm, we note that it might be possible to avoid the costly projection step if we treat even the deterministic constraints as “uncertain” constraints with a fixed uncertain parameters under the OCO framework. However, in practice this could lead to a larger number of iterations needed for convergence. Furthermore, the solution returned by this hypothetical algorithm would not fully satisfy the deterministic constraints of the feasibility set $\mathcal{X}$ (as do the OFO and the other algorithms we have studied), but only approximately up to a tolerance ϵ.

5 Robust Optimization via Single-row Nominal Oracle

In this chapter we take the idea of Ben-Tal et al. further: we present an algorithm with **provable guarantees** for solving a robust problem by solving only a 1-row version of the nominal problem in each iteration (instead of the whole nominal problem, whose feasibility version is seen in equation (4-3)). This is interesting when solving the full nominal problem in each iteration is prohibitively costly, as when it has many non linear constraints.

To make this chapter more self-contained, we recall some important notions. First, we want to approximately solve the following robust problem:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & c^T x \\ \text{subject to} \quad & f_i(x, u_i) \leq 0,\ \forall u_i \in \mathcal{U}_i,\ i = 1, \ldots, m \\ & x \in \mathcal{X}. \end{aligned} \tag{5-1}$$

Let OPT be the optimal value of this problem. We assume that each $f_i(x, u)$ is convex in x and concave in u, and that each $\mathcal{U}_i$ is convex. The nominal oracle used by the Dual-Subgradient approach [33] seen in Section 4.3 to approximately solve this problem is (in its optimization version) what is obtained by fixing a single scenario $\bar{u}_i \in \mathcal{U}_i$ for each robust constraint:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & c^T x \\ \text{subject to} \quad & f_i(x, \bar{u}_i) \leq 0,\ i = 1, \ldots, m \\ & x \in \mathcal{X}. \end{aligned} \tag{nominal oracle}$$

The *1-row nominal oracle* that we want to use is what is obtained by further aggregating the constraints of the nominal oracle using non-negative multipliers $\bar{p}_1, \ldots, \bar{p}_m$

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & c^T x \\ \text{subject to} \quad & \sum_{i=1}^{m} \bar{p}_i f_i(x, \bar{u}_i) \leq 0 \\ & x \in \mathcal{X}. \end{aligned} \tag{1-row oracle}$$

Observation 1 *We note that the optimization problem with constraint aggregation, presented in* (1-row oracle)*, can theoretically be unbounded even if* (nominal oracle) *is not. However, this drawback is easily avoidable, as it suffices to ensure that $\mathcal{X}$ includes a box constraint on the solutions x. Since we are simply trying to avoid unboundedness, even a very loose box will be sufficient.*

Theorem 1 *Consider a feasible robust problem* (5-1)*. Let G be a constant such that $||\nabla_u f_i(x,u)||_2 \leq G$ for all $x \in \mathcal{X}$, $u_i \in \mathcal{U}_i$ and $i \in [m]$. Let D represent the diameter of the sets $\mathcal{U}$, i.e., $D \geq \max_{u,v \in \mathcal{U}_i} ||u-v||_2$ for all $i \in [m]$. Further assume that $f_i(x,u) \in [-\rho, \rho]$ for all $x \in \mathcal{X}$, $u \in \mathcal{U}_i$ and $i \in [m]$. Finally, assume that* (1-row oracle) *is bounded. Then the algorithm Single-row based Robust Optimization computes a solution $\bar{x}$ with the following guarantees:*

1. *$\bar{x}$ is ϵ-feasible, namely $\bar{x} \in \mathcal{X}$ and $f_i(\bar{x}, u_i) \leq \epsilon$ for all $u_i \in \mathcal{U}_i$ and all $i = 1, \ldots, m$*
2. *The value of $\bar{x}$ is at most that of the optimal solution: $c^T\bar{x} \leq \mathrm{OPT}$.*

Moreover, the algorithm makes at most $\max\left\{\frac{(2GD)^2}{\epsilon^2}, \frac{8\rho^2 \ln m}{\epsilon^2}\right\}$ calls to 1-row oracles.

In order to get an intuition, first notice that the nominal oracle can be solved using 1-row oracles using the algorithm MWU-Based CP, presented in Section 3.3. Thus, the direct idea for solving the robust problem using 1-row oracles is to run the Dual-Subgradient algorithm of Ben-Tal et al. [33], and in each iteration, when a nominal oracle is needed, it is solved using 1-row oracles via the MWU-Based CP in Section 3.3. The main drawback of this method is the following: if the Dual-Subgradient algorithm [33] requires T_1 iterations and the MWU-Based algorithm for solving each nominal oracle requires T_2 iterations, the total calls for 1-row oracles ends up being $T_1 \cdot T_2$. Concretely, replacing the value of T_1 by $\Theta((GD)^2/\epsilon^2)$ (as shown in Section 4.3) and of T_2 by $\Theta(\rho^2 \ln(m)/\epsilon^2)$ (as shown in Section 3.3), this direct approach would require $\Theta((GD\rho)^2 \ln(m)/\epsilon^4)$ calls to 1-row oracles. Thus, that Theorem 1 above is saying is that we do not need to pay the **product** of T_1 and T_2 but essentially just the **largest** of these quantities, up to a small constant factor.

The main idea is to run the algorithms Dual-Subgradient and MWU-Based CP "in parallel", rather than "in series". More precisely, we show that running only a single iteration of MWU-Based CP per iteration of the Dual-Subgradient suffices for convergence. The algorithm Single-row based RO is more formally described in Algorithm 9.

Algorithm 9 Single-row based Robust Optimization

```
Tolerance $\epsilon$
Parameters $D$, $G$ of the OGD framework
Parameter $\rho$, the width of the problem
Set $T = \max\left\{\frac{(2GD)^2}{\epsilon^2}, \frac{8\rho^2 \ln m}{\epsilon^2}\right\}$
Initialize $\{u_1^0, \dots, u_m^0\}$
Initialize $p^1 = 1/m \ \forall i \in [m]$
for $t = 1, \dots, T$ do
    Compute $x^t$ as the optimal solution to the problem:
        $$\begin{aligned} \underset{x}{\text{minimize}} \quad & c^T x \\ \text{subject to} \quad & \sum_{i=1}^{m} p_i^t f_i(x, u_i^t) \leq 0 \\ & x \in \mathcal{X}. \end{aligned}$$
    if Oracle declared infeasibility then
        return Robust problem is infeasible
    end if
    Define gain vector $g^t \in \mathbb{R}^m$ for the MWU, with coordinates $g_i^t := f_i(x^t, u_i^t)$
    Feed the gain function $g^t$ to the generalized MWU seen in Corollary 1, obtaining the updated distribution $p^{t+1}$
    for $i = 1, \dots, m$ do
        Define the gain function $h_i^t : \mathcal{U}_i \to \mathbb{R}$ as $h_i^t(u) := f_i(x^t, u)$
        Feed the gain function $h_i^t$ to the $i$th OGD, obtaining the updated point $u_i^{t+1} \in \mathcal{U}_i$
    end for
end for
return $\bar{x} = \frac{1}{T}\sum_{t=1}^{T} x^t$
```

The idea of running Dual-Subgradient and MWU-Based CP "in parallel" can be better visualized in Figure 5.1, a diagram that illustrates how the algorithm works.

Figure 5.1 also serves to further clarify the difference between our approach and the Dual-Subgradient algorithm. A diagram of the latter would not have the *MWU* box, and therefore would have to solve the nominal problem with m constraints, instead of our easier (1-row oracle).

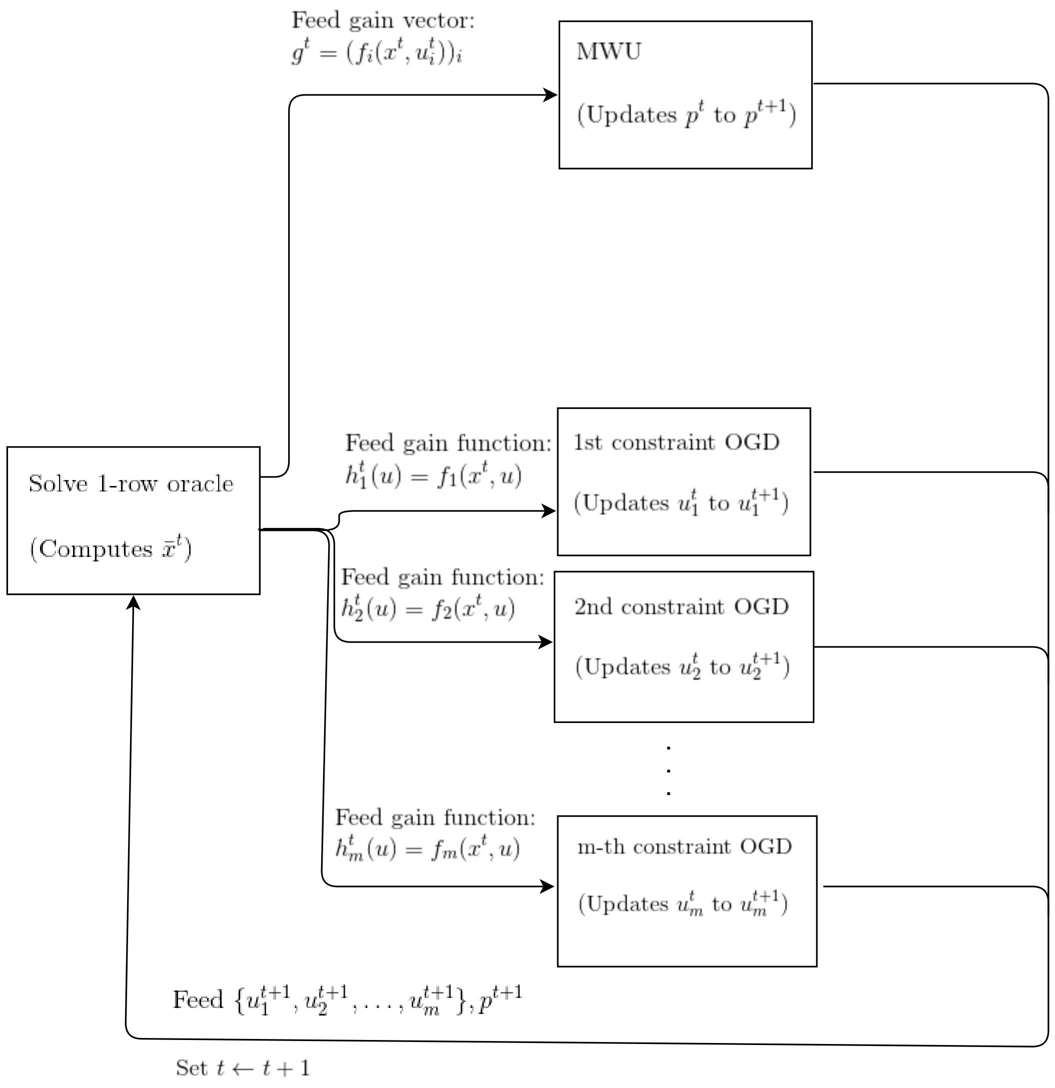

Figure 5.1: Illustration of the algorithm Single-row based RO

Proof of Theorem 1. First we argue that the returned solution $\bar{x}$ has value $c^T\bar{x} \leq \mathrm{OPT}$: this follows from the fact that since x^t is the optimal solution to a relaxation of the robust minimization problem (with same objective function), we have $c^Tx^t \leq \mathrm{OPT}$, and since $\bar{x} = \frac{1}{T}\sum_{t=1}^{T} x^t$ the result follows.

Now we argue that this solution is ϵ feasible. Since each function $f_i(x,u)$ is convex in x, by Jensen's inequality we have for all i and u

$$f_i(\bar{x}, u) \leq \frac{1}{T}\sum_{t=1}^{T} f_i(x^t, u).$$

Thus, to show ϵ feasibility of $\bar{x}$ it suffices to show that the right-hand side of the above inequality is at most ϵ for all i, $u \in \mathcal{U}_i$, namely

$$\max_{i=1,\dots,m} \max_{u_i \in \mathcal{U}_i} \frac{1}{T}\sum_t f_i(x^t, u_i) \leq \epsilon. \tag{5-2}$$

Let $\Delta \subseteq \mathbb{R}^m$ denote the set of all probability distributions over m coordinates. Since for every vector $v \in \mathbb{R}^m$ we have $\max_i v_i = \max_{p\in\Delta} \sum_i p_i v_i$, the left-hand side of (5-2) is equal to

$$\max_{p\in\Delta} \sum_i p_i \cdot \max_{u_i\in\mathcal{U}_i} \frac{1}{T}\sum_t f_i(x^t, u_i) = \max_{p\in\Delta} \max_{u_1,\dots,u_m} \frac{1}{T}\sum_t \sum_i p_i\, f_i(x^t, u_i),$$

where in the right-hand side u_i ranges over all of $\mathcal{U}_i$. Thus, to show that $\bar{x}$ is ϵ feasible, it suffices to show that this right-hand side is at most ϵ:

$$\max_{p\in\Delta} \max_{u_1,\dots,u_m} \frac{1}{T}\sum_t \sum_i p_i\, f_i(x^t, u_i) \leq \epsilon \quad \Rightarrow \quad \bar{x} \text{ is } \epsilon\text{-feasible.} \tag{5-3}$$

We prove the left-hand side in two steps. The first, a "dual" type bound, shows that the iterates $\{p^t\}_t, \{u_i^t\}_{i,t}$ computed via online learning approximate the maximum in the left-hand side.

Lemma 4 (Dual guarantee) *We have that*

$$\max_{p\in\Delta} \max_{u_1,\dots,u_m} \frac{1}{T}\sum_t \sum_i p_i\, f_i(x^t, u_i) \leq \frac{1}{T}\sum_t \sum_i p_i^t\, f_i(x^t, u_i^t) + \frac{GD}{\sqrt{T}} + \frac{2\rho\sqrt{\ln m}}{\sqrt{T}}.$$

Prova. Fix $i \in \{1,\dots,m\}$. Since the sequence $\{u_i^t\}_t$ was computed using the OGD over gain functions $h_i^t(u) = f_i(x^t, u)$, the OGD guarantee from [41] (given in equation (3-5)) gives

$$\sum_t f_i(x^t, u^t) \geq \max_{u_i\in\mathcal{U}_i} \sum_i f_i(x^t, u_i) - GD\sqrt{T}.$$

Similarly, since the sequence $\{p^t\}_t$ was computed using MWU over the gain vectors $g^t = (f_i(x^t, u_i^t))_i$, the MWU guarantee from Corollary 1 gives

$$\sum_t \sum_i p_i^t \, f_i(x^t, u_i^t) \geq \max_{p \in \Delta} \sum_t p_i \, f_i(x^t, u_i^t) - 2\rho\sqrt{T \ln m}.$$

Chaining these two displayed inequalities gives

$$\sum_t \sum_i p_i^t \, f_i(x^t, u_i^t) \geq \max_{p \in \Delta} \max_{u_1, \ldots, u_m} \sum_i p_i \, f_i(x^t, u_i) - GD\sqrt{T} - 2\rho\sqrt{T \ln m}.$$

Rearranging this inequality concludes the proof. ■

The second step is the "primal" guarantee: it follows directly from the fact that x^t is computed as a "feasible" solution for p^t and the u_i^t's, namely $\sum_i p_i^t \, f_i(x^t, u_i^t) \leq 0$.

Lemma 5 (Primal guarantee) *We have that*

$$\frac{1}{T} \sum_t \sum_i p_i^t \, f_i(x^t, u_i^t) \leq 0$$

Putting these two lemmas together we obtain that

$$\max_{p \in \Delta} \max_{u_1, \ldots, u_m} \frac{1}{T} \sum_t \sum_i p_i \, f_i(x^t, u_i) \leq \frac{GD}{\sqrt{T}} + \frac{2\rho\sqrt{\ln m}}{\sqrt{T}}.$$

But the choice of T makes the right-hand side at most ε. This concludes the proof of Theorem 1.

6
A cutting-set method with constraint aggregation

In this chapter, we aim to modify our Single-row based RO algorithm presented in Chapter 5 in order to seek better practical performance. Unfortunately, this loses the theoretical guarantees proved in Chapter 5.

To do this, we focus on three main ideas:

1. Accumulate the aggregated constraints of the Single-row based RO algorithm, inspired in part by the Cutting-set method of Mutapcic and Boyd [38]
2. Change the updating scheme of the uncertain parameters, by using either full pessimization or backtracking line search
3. Change the updating scheme of the probability distributions p^t

We will now briefly discuss each of these points in more detail.

6.1
Accumulating constraints

Out of the three algorithms described in Chapter 4, Mutapcic and Boyd's Cutting-set method [38] seems to be the one most widely used in practice, despite its lack of tight guarantees. Studies such as [36] and [56] show that, even without good guarantees, the cutting-set method often outperforms reformulation and regret minimization methods.

We argue that one of the main reasons for the Cutting-set method's better performance is the fact that the calculation of each solution x^t essentially takes into account all (important) previous realizations of the uncertain parameters. On the other hand, due to the way the regret minimization methods work, only the current realization of the noise is considered when generating a new solution. In terms of game theory, Player 1 (i.e. the one choosing x^t's) has much more information at his disposal in the latter case. As a result, the Cutting-set method typically need much less iterations to converge, as also shown in Kroer's results [56]. For this reason, we will incorporate constraint accumulation in this practical version of our algorithm. In each iteration, we add two constraints: one aggregated constraint and the single worst constraint of the iteration. This process will become more clear in Section 6.3.

6.2 Updating the uncertainty parameters

When using OGD (in its maximization version) to update the noises u_i^t, as proposed in the regret minimization methods and particularly in our Single-row based RO method seen in Chapter 5, a similar situation to the one described several times in this work is presented: in order to ensure that the guarantees hold, often too much conservatism is necessary. In this case, the step size η taken to update each u_i^t within OGD can be too small. In our setting, for example, we have $\eta = \mathcal{O}(1/\sqrt{T})$ and $T = \mathcal{O}(1/\epsilon^2)$, which means that for small values of tolerance ϵ, T could already become too big and η, too small.

One way to speed up the convergence in practice is to select the gradient step size η in an optimized way using a technique commonly used in offline methods: backtracking line search (see, e.g., [57, 58]). In fact, Ho-Nguyen et al also used a similar line search technique when comparing iterative approaches to solve robust optimization [39], even though the theoretical guarantees are lost. The idea of the backtracking line search method is to first try to use a full step of $\eta = 1$ on the direction of the gradient (while ensuring feasibility). If this does not lead to a point that increases the current value of the function we are trying to maximize, the step is reduced by half, and this is done repeatedly until a point that increases the current value of the function is found. In practice, this allows the algorithm to use larger steps whenever possible, instead of always using a small value of η.

However, another reason why Mutapcic and Boyd's cutting-set procedure seems to outperform most others in practice is the fact that, for many common problems in robust optimization, running a full pessimization oracle to find the worst possible u_i, instead of simply updating it using a gradient step, is not that costly. In special cases, such as the robust version of LP with ellipsoidal uncertainty, there is even an analytic solution that can be easily computed at the cost of computing a norm. This fact is also discussed by Ho-Nguyen et al in [39], chapter 6. Naturally, when the worst possible u_i is not too hard to compute, it is often worth finding it at every iteration instead of running first order procedures that could have a potentially slow convergence.

For this reason, in our optimized algorithm, instead of using a first-order method, we use a pessimizing oracle, as defined in (4-2), to perform an optimization on the noises of each constraint, similarly to what is done in [38]. As also discussed in [38], in cases in which this pessimization is prohibitively costly, it can still be done approximately or via backtracking line search. A big advantage of using the *exact* pessimizing oracle is that since it computes worst-case noises for the current solution x^t, we are essentially checking for the robust feasibility of x^t at each iteration. Whenever (4-2) is solved for every constraint and no violations bigger than ϵ are observed, the algorithm can stop, since x^t is already an $\epsilon-$feasible robust solution.

6.3 Updating the probability distributions of the constraints

We would like to make the update of p^t more direct, hopefully resulting in a faster convergence of the algorithm. We *started* with the following idea: once a solution x^t is calculated by the oracle and a noise u_i^t is computed for each constraint, the new probability distribution can be calculated as follows:

$$w_i^t = \begin{cases} f_i(x^t, u_i^t), \text{ if} f_i(x^t, u_i^t) > 0 \\ 0, \text{ otherwise} \end{cases}$$

and

$$p_i^t = \frac{w_i^t}{\sum_{i=1}^m w_i^t} \;\; \forall i \in [m] \tag{6-1}$$

This means that in the new probability distribution, only constraints violated by solution x^t have weight, and its weight is proportional to the violation. After this new distribution is calculated, the following new constraint can be *added* to the minimization problem solved, in each iteration, by the oracle:

$$\sum_{i=1}^m p_i^t f_i(x, u_i^t) \leq 0$$

However, we note that the setting described leads to a big problem: it could potentially *cycle* and never converge. Suppose, for example, a case in which only 3 constraints are violated by the current solution x^t, and with equal violations. The subsequent p^t to be added would give them equal weights of 1/3. Then, the new solution x^t could still violate the same 3 constraints with the same weight, forcing the algorithm to add the same constraint to the problem indefinitely.

A less obvious but still relevant problem with this setting can be discussed as follows: if a given constraint is itself too hard, it should probably be dealt with individually, in order to increase the speed of convergence for the algorithm as a whole.

We then propose the following modification: once the noises are updated and the violations of the current solution x^t are known for each constraint, we add two constraints to our problem:

1. The constraint $f_{v_t}(x, u_{v_t}^t) \leq 0$, where $v_t \in [m]$ represents the index of the

constraint with the worst violation in the current time step

2. An aggregation of the *other* constraints, $\sum_{i=1}^{m} p_i^t f_i(x, u_i^t) \leq 0$, where p^t comes from weights computed as in (6-2)

$$w_i^t = \begin{cases} f_i(x^t, u_i^t), \text{ if } f_i(x^t, u_i^t) > 0 \text{ and } i \text{ is not } v_t \\ 0, \text{ otherwise} \end{cases} \tag{6-2}$$

In summary, our algorithm starts with one constraints and, in each round, adds two new ones. To illustrate this setting, after s rounds the problem being solved by the oracle will be the following:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & c^T x \\ \text{subject to} \quad & \sum_{i=1}^{m} p_i^1 f_i(x, u_i^1) \leq 0 \\ & f_{v_2}(x, u_{v_2}^2) \leq 0 \\ & \sum_{i=1}^{m} p_i^2 f_i(x, u_i^2) \leq 0 \\ & f_{v_3}(x, u_{v_3}^3) \leq 0 \\ & \sum_{i=1}^{m} p_i^3 f_i(x, u_i^3) \leq 0 \\ & \dots \\ & f_{v_s}(x, u_{v_s}^s) \leq 0 \\ & \sum_{i=1}^{m} p_i^s f_i(x, u_i^s) \leq 0 \\ & x \in \mathcal{X} \end{aligned} \tag{modified nominal problem}$$

6.4 Overview of the algorithm

We now summarize the procedure described so far in Algorithm 10

Algorithm 10 Cutting-set method with constraint aggregation

```
Tolerance $\epsilon$
Initial sampled sets $\hat{\mathcal{U}}_i = \{\bar{u}_i\},\ i = 1, \ldots, m$
while $\max_{i \in [m]} f_i(x^t, u_i^t) > \epsilon$ do
    Solve modified version of the nominal problem with aggregated constraints, as exemplified in equation (modified nominal problem)
    if This version is infeasible then
        Return "Robust problem is infeasible"
    end if
    Let $x^t$ be the returned solution
    for $i = 1, \ldots, m$ do
        Solve pessimizing oracle (4-2) with fixed $x^t$ to evaluate $u_i^*$ and $\max_{u_i \in \mathcal{U}_i} f(x^t, u_i)$ - exactly or approximately
    end for
    Let $v_t$ be the index of the constraint with worst violation
    Compute weights vector as in (6-2)
    Compute probability vector as in (6-1)
    Add the constraints $f_{v_t}(x, u_{v_t}^t) \leq 0$ and $\sum_{i=1}^{m} p_i^t f_i(x, u_i^t) \leq 0$ to the modified nominal problem being solved
end while
Return x$^t$
```

7
Computational Experiments

In this chapter we will present computational experiments to illustrate the performances of three algorithms: the cutting-set method [38] seen in Section 4.2, a modification of the Dual-Subgradient method seen in 4.3 using backtracking line search to update the uncertainties (instead of using the theoretical value of the step η, which can be way too small for the algorithm to converge in reasonable time, as discussed in Section 6.2) and our cutting-set method with constraint aggregation, the algorithm described in Chapter 6.

One important comment regarding the implementation of our modified Dual-Subgradient method is the following: since the uncertainty parameters are not found with a pessimizing oracle, but instead with a first-order method, we are not able to know in every iteration if the current solution or the averaged solution so far are robust feasible. Running this check in every iteration is not a good strategy, since part of the motivation for this algorithm is to avoid the need to run pessimizing oracles for each constraint in every iteration. Therefore, we run pessimizing oracles on the following situations:

1. If, in a given iteration t, the solution x^t does not violate any constraint by more than the tolerance ϵ considering the *current* noises u_i^t found by first-order methods, we then run pessimizing oracles to check if x^t is in fact approximately robust feasible even for worst case noises

2. Every 5 iterations, we check the average of the solutions so far for robust $\epsilon-$feasibility.

We also implemented the Online First-Order procedure [39] seen in Section 4.4. However, we do not include its results in our comparisons, since even using backtracking line search to update the first-order procedures, we were not able to make the algorithm converge in reasonable time. We speculate that this happened because:

1. Unlike the computational experiments performed in [39] and [56], which have one uncertain constraint, our tests have several constraints subject to uncertainty. This seems to be able to cause a slow convergence when x^t is updated via an Online Learning procedure.

2. Our experiments also have deterministic equality constraints, apart from the uncertain inequalities. For this reason, the projection of a solution x^t into the feasible set is not trivial, since an optimization with several equality constraints is needed in each iteration.

The tests were run on Julia, using the JuMP package and with either Gurobi or Mosek as solvers, depending on which performed better in a given situation. As an example, Gurobi seemed to be faster to solve LPs (linear programs) in our case, but Mosek had better performance on QCQPs (quadratically constrained quadratic programs).

7.1 Robust version of an LP with ellipsoidal uncertainty

Consider the following LP.

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & c^T x \\ \text{subject to} \quad & \tilde{a}_i^T x \leq b_i \\ & x \geq 0 \end{aligned} \tag{7-1}$$

For simplicity of notation, we let only the parameters $\tilde{a}_i \in \mathbb{R}^n$ be subject to uncertainty. They are known to lie in given ellipsoids:

$$\tilde{a}_i \in \{a_i + P_i u_i : ||u_i||_2 \leq 1\}$$

where $P_i \in \mathbb{R}^{n \times K}$ is a given scaling matrix and $u_i \in \mathbb{R}^K$. Note that if P_i is a matrix of zeros, then $\tilde{a}_i$ is exactly known and the problem reduces to a deterministic LP.

Problem (7-1) can be rewritten as:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & c^T x \\ \text{subject to} \quad & (a_i + P_i u_i)^T x \leq b_i \ \forall u_i : ||u_i||_2 \leq 1 \ \forall i \in [m] \\ & x \geq 0 \end{aligned}$$

which is equivalent to:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & c^T x \\ \text{subject to} \quad & a_i^T x + u_i^T P_i^T x \leq b_i \ \forall u_i : ||u_i||_2 \leq 1 \ \forall i \in [m] \\ & x \geq 0 \end{aligned} \tag{7-2}$$

If x is to be feasible for every possible realization of u with $||u||_2 \leq 1$, then it must be feasible for the worst case. Then, using the known fact that:

$$\sup_{u^T P^T x : ||u||_2} = ||P^T x||_2$$

we can rewrite problem (7-2), which theoretically has infinitely many constraints, as:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & c^T x \\ \text{subject to} \quad & a_i^T x + ||P_i^T x||_2 \leq b_i \ \forall i \in [m] \\ & x \geq 0 \end{aligned} \tag{7-3}$$

which an SOCP.

7.1.1 Iterative methods

We now aim to approximately solve the robust problem without having to reformulate it as an SOCP as presented in (7-3). For the Dual-Subgradient method we need to compute the gradient of the constraint functions with respect to the noises. Defining $f_i(x, u_i) = (a_i + P_i u_i)^T x - b_i$, we have:

$$\nabla_u f(x, u) = P_i^T x$$

Therefore, the update of the variables u_i^t with a first-order method takes the following form:

$$u_i^t = \frac{u_i^{t-1} + \eta P^T x}{\max\{||u_i^t + \eta P^T x||_2, 1\}}$$

7.1.2 Problem instances

For our tests, we use instances from the well known NETLIB library for Linear Programming problems [59]. Most of the instances in NETLIB have both equality and inequality constraints. Since we only want to handle uncertainty on the inequality constraints, the others remain deterministic. We also re-scaled the inequalities so that each $b_i = 1$, which does not change the optimal point nor the optimal value of the solution, but ensures that a tolerance of ϵ for each constraint is more meaningful. The problems we want to solve can therefore be expressed as:

$$\begin{aligned}
\underset{x}{\text{minimize}} \quad & c^T x \\
\text{subject to} \quad & (a_i + P_i u_i)^T x \leq b_i \ \forall u_i : ||u_i||_2 \leq 1 \ \forall i \in [m] \\
& d_l^T x = e_l \ \forall l \in [q] \\
& x \geq 0
\end{aligned}$$

We consider the dimension K of each vector $u_i \in \mathbb{R}^K$ to be equal to n. Each scaling matrix P_i is created as a diagonal $\mathbb{R}^{n\times n}$ matrix, and the value of such diagonal in the $j-$th row corresponds to the $j-$th entry multiplied by 5%. This means that we let each entry of the matrix A, individually, to vary up to 5%. As a brief **example**, consider the case in which one row of an LP is given by:

$$\begin{pmatrix} 3 & 5 & 7 \end{pmatrix}^T x \leq b \tag{7-4}$$

Then, we set the matrix P corresponding to this row as

$$\begin{pmatrix} 0.3 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.7 \end{pmatrix}$$

Then, the robust version of the inequality in (7-4) becomes:

$$(3 + 0.3u_1)x_1 + (5 + 0.5u_2)x_2 + (7 + 0.7u_3)x_3 \leq b; \quad ||u||_2 \leq 1$$

7.1.3 Results

We ran the three algorithms for the robust version of several problems from *NETLIB*, with tolerance $\epsilon = 0.005$. We highlight that many of the instances became infeasible even with such small perturbation (we do not include them here).

We also report the time required for the reformulation method, i.e., solving the SOCP described in Equation (7-3). The results can be seen in Table 7.1. The Dual-Subgradient method failed to converge on instance *brandy*.

				SOCP	Cutting-set			Cutting-set with constraint aggregation			Dual-Subgradient with backtracking LS	
Instance	n	m	q	Time (s)	Time (s)	Iter.	Inequality constraints in biggest problem solved	Time (s)	Iter.	Inequality constraints in biggest problem solved	Time (s)	Iter.
afiro	32	19	8	0.021	**0.012**	3	26	0.016	11	21	0.16	30
blend	83	31	43	0.11	**0.04**	6	62	0.2	23	44	0.57	55
beaconfd	262	33	140	0.89	0.18	2	34	**0.17**	2	2	3.65	30
brandy	249	54	166	1.21	**0.91**	9	103	3.26	46	88	-	-
lotfi	308	58	95	1.85	**0.59**	2	74	2.24	20	38	21.18	105
scagr7	140	45	84	0.51	**0.08**	4	64	0.63	19	36	3.19	40
scarg25	500	171	300	13.13	**5.93**	7	275	43.27	101	201	23.43	35
agg2	302	456	60	13.46	**6.70**	8	672	66.18	184	366	83.78	145

Table 7.1: Results for the robust versions of NETLIB LP problems

The results indicate that, in this setting, the Cutting-set method by Mutapcic and Boyd [38] shines. Since solving bigger versions of the extended nominal problem is typically not an issue when the problem is an LP (which can be solved very efficiently), the empirical fact that the algorithm often converges in very few iterations becomes a big advantage. The exact reformulation technique also works reasonably well (although slower than the cutting-set). The Dual-Subgradient method suffers from the fact that since the uncertainty parameters are not found by pessimization but by OGD, it often takes several iterations for it to converge. In the particular case of the Robust LP, updating the noises via OGD does not represent an advantage, especially because the pessimizing oracle also has the computational cost of computing a norm.

This setting is not too favorable for our method, especially because aggregating constraints does not represent a big computational difference when solving LPs of these sizes. However, part of the motivation can already be understood: the biggest LP that we have to solve is often much smaller than that of the pure cutting-set method. In many cases, it is even smaller than m, the number of constraints of the nominal problem. In some extreme cases this becomes more evident: in the *beaconfd* instance, we start with one aggregated constraint and only have to add a single new one in order to obtain an $\epsilon-$feasible solution for the robust problem.

For illustration purposes, Figure 7.1 shows the maximum violation in each iteration of the *scagr7* instance for the Cutting-set method and the Cutting-set with constraint aggregation, until each of them converge (on the 4th and 19th iterations, respectively). We note that Cutting-set method already starts with a relatively small error, since the full nominal problem is solved. On the other hand, our method starts with a greater violation and, over the iterations, the error is reduced due to the constraint accumulation process. Again, we highlight that even though it is clear why our algorithm typically takes more

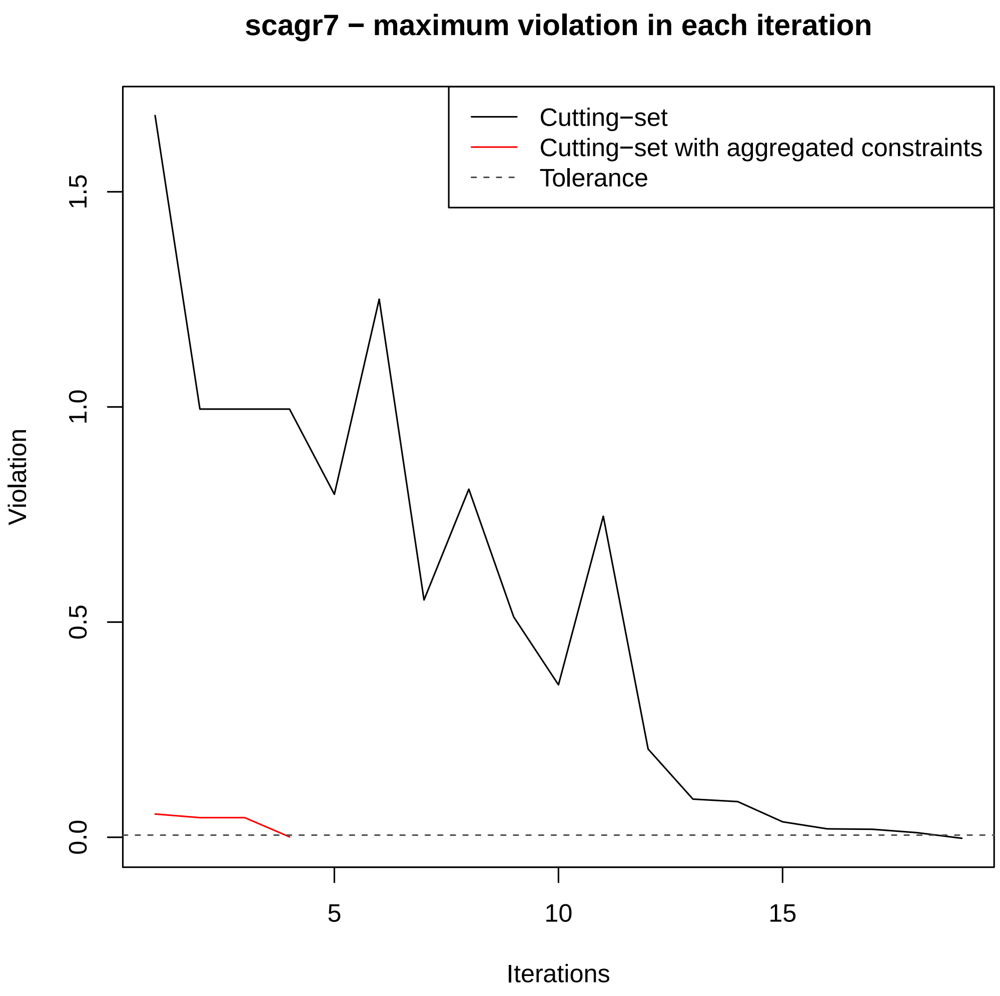

Figure 7.1: Evolution of the maximum violation over the iterations for the *scagr7* problem

iterations to converge, each iteration becomes less computationally costly.

7.2 Robust version of a QCQP with ellipsoidal uncertainty

We are now interested in a case of Quadratically Constraints Quadratic Programming with ellipsoidal uncertainty. Several applications of large scale QCQPs can be seen in [60]. The nominal version of the optimization problem can be written as follows:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f_0^T x \\ \text{subject to} \quad & ||A_i x||_2^2 \leq b_i^T x + c_i, \ \forall i \in [m] \\ & x \in \mathcal{X} \end{aligned} \tag{7-5}$$

where $A_i \in \mathbb{R}^{n \times n}$, $b_i \in \mathbb{R}^n$ and $c_i \in \mathbb{R}$. We are interested in its robust version in which we consider that the matrices A_i are uncertain, given by:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f_0^T x \\ \text{subject to} \quad & \sup_{u_i \in \mathcal{U}_i} ||\Big(A_i + \sum_{k=1}^{K} P_k^i u_i^{(k)}\Big)x||_2^2 \leq b_i^T x + c_i, \; \forall i \in [m] \\ & x \in \mathcal{X} \end{aligned} \tag{7-6}$$

in which $P_1^i, P_2^i, \ldots, P_K^i \; \forall i \in [m]$ are uncertainty matrices. Since we are working with ellipsoidal uncertainty, $\mathcal{U}_i = \{u \in \mathbb{R}^K : ||u||_2 \leq 1\}$.

7.2.1 Reformulation as an SDP

We will now explore what would be done in order to solve the robust problem in the most common way, with a reformulation to its robust counterpart. This is discussed in Ben-Tal and Nemirovski's 2002 work [26].

In [26], and later in [24], it is proven that the reformulation of the minimization version of (7-6) (in which we only consider uncertainty in the matrices A_i) is given by:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f_0^T x \\ \text{subject to} \quad & \begin{bmatrix} x^T b_i + c_i - \lambda_i & 0 & 0 & \ldots & 0 & [A_i^0 x]^T \\ 0 & \lambda_i & 0 & 0 & 0 & [P_i^1 x]^T \\ 0 & 0 & \lambda_i & 0 & 0 & [P_i^2 x]^T \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & 0 & \lambda_i & [P_i^K x]^T \\ A_i x & P_i^1 x & P_i^2 x & \ldots & P_i^K x & I \end{bmatrix} \succeq 0, \; \forall i \in [m] \end{aligned} \tag{7-7}$$

with variables $x \in \mathcal{X}$ and $\lambda \in \mathbb{R}^m$.

One important aspect of problem (7-7) is the size of its constraints: we need to create m matrices, each in $\mathbb{R}^{(n+K+1)\times(n+K+1)}$. As we will see, this leads to memory problems even with instances of only moderate size. Moreover, it is known, as discussed in [33], that solvers can handle QPs two to three orders of magnitude larger than SDPs.

As an example, we were successfully able to use the above formulation for small values of m and n, in order to compare the optimal solution with the ones being found by the iterative methods. However, we were not able to solve this formulation for most of the instances we generate in the next section, because the solver runs out of memory. This motivates the use of iterative methods that do not require a conversion to the robust counterpart.

7.2.2 Iterative methods

It should be noted that it is not immediately clear that this problem can be solved with the iterative frameworks presented in this work, since they require the constraint functions to be both convex in x and concave in u_i. In the case of robust QCQP, the constraint functions are convex in x but not concave in u_i.

However, a reformulation can be performed in order to circumvent this problem [39,61]. We will first introduce some definitions, following the notation in [39].

First, define, for each $i \in [m]$, the matrix $\mathcal{P}_x^i \in \mathbb{R}^{n \times K}$, whose column $k \in [K]$ is given by $P_k^i x$.

Then, define:

$$\mathcal{Q}_x^i = (P_x^i)^T (P_x^i) \in \mathbb{S}_+^K$$

$$r_x^i = (P_x^i)^T A_i x \in \mathbb{R}^K$$

$$s_x^i = ||A_i x||_2^2 - b_i^T x - c_i \in \mathbb{R}$$

It follows directly that:

$$\left\| \left(A_i + \sum_{k=1}^{K} P_k^i u_i^{(k)} \right) x \right\|_2^2 - b_i^T x - c_i = u_i^T \mathcal{Q}_x^i u_i + 2(r_x^i)^T u_i + s_x^i$$

We then define for each $i \in [m]$ the function $\phi^i(x, u_i)$ as:

$$\phi^i(x, u_i) := \left\| \left(A_i + \sum_{k=1}^{K} P_k^i u_i^{(k)} \right) x \right\|_2^2 - b_i^T x - c_i + \lambda_{\max}(\mathcal{Q}_x^i)(1 - \|u_i\|_2^2)$$

and therefore it also holds that:

$$\phi^i(x, u_i) = u_i^T \mathcal{Q}_x^i u_i + 2(r_x^i)^T u_i + s_x^i + \lambda_{\max}(\mathcal{Q}_x^i)(1 - \|u_i\|_2^2) \qquad (7\text{-}8)$$

There are two important observations to be made about the functions as defined in (7-8) (both are proven in, e.g., [39], Lemma 5.1). One is that it is concave in u_i, as initially desired. Even more importantly, the following relation is shown:

$$\sup_{u_i \in \mathcal{U}_i} \left\| \left(A_i + \sum_{k=1}^{K} P_k^i u_i^{(k)} \right) x \right\|_2^2 - b_i^T x - c_i = \sup_{u_i \in \mathcal{U}_i} \phi^i(x, u_i)$$

Since this reformulation enables us to maximize the constraints in terms of the scenarios u_i, we can then use the iterative methods to approximately solve the robust problem.

Finally, note that, with this reformulation, the gradient of ϕ_i with respect to u_i (which we need for the Dual-Subgradient method) is given by:

$$\nabla_{u_i}\phi^i(x, u_i) = 2(Q_x^i - \lambda_{max}(Q_x^i I_K)u_i + 2r_x^i$$

7.2.3 Problem instances

The NETLIB library that we used for the Robust LP problems, described in Section 7.1, has many instances of reasonable size that were well suited to test the methods studied in this work. However, this is not the case for QPLIB [62] library of QCQP problems: most of the instances in it are either too large (with tens of thousands of variables and/or constraints) or not even convex. Therefore, the QPLIB library is not ideal to study robust problems. We note than in the QCQP literature, the problems are very often randomly generated in such a way as to create problems that are reasonably hard to solve (such as in [39,56,60,63,64]). The recent works that performed computational experiments with robust QCQPs only studied problems with a *single* uncertain constraint, such as the Robust SVM [39,56]. We will therefore randomly generate instances in a similar way as described in [63] (which does not treat the robust case). The *nominal* problem for which we want to generate instance is as follows:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f_0^T x \\ \text{subject to} \quad & ||A_i x||_2^2 \le b_i^T x + c_i, \ \forall i \in [m] \\ & d_l^T x \ge e_l \ \forall l \in [q] \\ & x \in [0,1]^n \end{aligned} \tag{7-9}$$

where $q = \lceil m/10 \rceil$.

This means that we aim to create m quadratic constraints and q linear constraints. The matrices A_i are randomly sampled with every entry in the range $[-1, 1]$, in such a way that ensures A_i to be symmetric and the overall problem to be is convex. As in [63], we uniformly sample each entry of each b_i, d_l, f_0 and also each e_l to be in the range $[-1, 1]$. Finally, while [63] samples each c_i from a uniform distribution in the range $[0, 100]$, we reduce this range to $[0, 10]$ in order to avoid possibly creating constraints that are too easy. Nonetheless, generating the instances in this way creates a favorable setting for our algorithm, since they have inherently different levels of slackness. Some of them will naturally be easier than others, and are therefore more likely to be already satisfied by the aggregated constraints.

For the robust case, we then consider that the matrices A_i are uncertain, as illustrated in Equation (7-6). We generate each matrix P_k^i as follows: for each i and each k, P_k^i is initialized a matrix of zeros. Then, we sample 20% of its entries, and set its values to the absolute values of the corresponding entries in A_i multiplied by 0.1. The intuition is that each P_k^i represents a possible "scenario" that affects some of the coefficients in A_i, but not all at the same time (as that would be too pessimistic).

One practical advantage of setting the matrices P_k^i in this manner is that they can be stored as sparse matrices. If they had to be stored as dense matrices, we could run into memory problems when attempting to solve big instances.

Now, we can apply the iterative methods to approximately solve this robust problem. We set the tolerance ϵ to 0.001, and run the problem for values of $m \in \{50, 100, 200, 400, 600\}$ and, similarly, also vary $n \in \{50, 100, 200, 400, 600\}$. We also set K, the dimension of each u_i, as 15, similarly to [39]. Setting K too big could result in memory problems, since we need K $n \times n$ matrices for each of the m non-linear constraints.

For each pair of m and n we run the algorithms five times and provide averaged solutions. In the next section we show and discuss the results.

7.2.4 Results

The results, for varying values of m and n, are shown in Tables 7.2, 7.3, 7.4, 7.5 and 7.6. We first note that the reformulation approach, that solves the SDP presented in Equation (7-7), was only able to solve the two smallest classes of instances: the ones with $(m = 50, n = 50)$, in which it took on average 139.4 seconds, and the ones with $(m = 100, n = 50)$, in which it took 283.2 seconds on average (both much higher than the times needed by the cutting-set methods). For all the other instances, the solver ran **out of memory**.

The solver also ran out of memory when running the pure cutting-set method, on three classes of instances: $(m = 400, n = 600)$, $(m = 600, n = 400)$ and $(m = 600, n = 600)$. For these, our cutting-set with constraint aggregation method successfully found a solution without running into memory problems precisely because it was able to converge before the problem being solved became too big.

For each test, we set a time limit of 20 minutes, except for the ones with $m = 600$ or $n = 600$, for which we set the limit to one hour. It is important to note that the Dual-Subgradient method could not converge (i.e., achieve robust violation smaller than the tolerance of 0.001) in reasonable time in most of the cases. The column *Converged?* indicates the percentage of instances in which the Dual-Subgradient converged. This happened mainly for two reasons:

– We reduced the tolerance from $\epsilon = 0.005$ in the robust LP case to $\epsilon = 0.001$ in the robust QCQP experiment, to compensate for the fact that the instances generated are not as adversarial as some of the *NETLIB* ones. Since the Dual-Subgradient method typically has to rely on the convergence of the averaged solution $\bar{x}$, intuitively the convergence can become too slow if the tolerance is too small.

– Since the nominal problem, unlike the LP case, is itself costly, the algorithm is not able to run as many iterations as it would need for convergence.

As for the comparison between our cutting-set method with constraint aggregation and the original version of [38], we note that for smaller problems, such as in Table 7.2, our algorithm does not have an advantage, since each iteration is not too costly for the original cutting-set method [38].

However, taking, for example, the instance ($m = n = 400$), the pure cutting-set method must solve a problem with at least 400 non-linear constraints up to three times, while the biggest problem our method had to solve (on average) had 45 non-linear constraints, since it converged in 22 iterations. This is possible because some of the constraints are inherently easier than others, and we are able to satisfy their robust version with aggregated constraints, instead of including them individually in the problem.

	Cutting-set		Cutting-set with constraint aggregation		Dual-Subgradient with backtracking LS		
n	Iter.	Time(s)	Iter.	Time(s)	Converged?	Iter. (when converged)	Time(s) (when converged)
50	2.6	1.02	5	1,02	0.4	12	3.05
100	2.4	**2.79**	7	3.07	0	-	-
200	2.4	**6.31**	9.4	8.73	0.2	11	35
400	2.2	**27.1**	10.2	30.2	0	-	-
600	2	**108.4**	15.2	173.2	0	-	-

Table 7.2: Results for the case with $m = 50$

	Cutting-set		Cutting-set with constraint aggregation		Dual-Subgradient with backtracking LS		
n	Iter.	Time(s)	Iter.	Time(s)	Converged?	Iter. (when converged)	Time(s) (when converged)
50	3	**2.35**	6.8	2.75	0.4	8	3.27
100	3	5.95	8.4	**5.85**	0	-	-
200	2.8	18.4	11.4	**14.35**	0	-	-
400	2	**58.5**	16.5	76.3	0	-	-
600	2	277.4	20.2	**263.42**	0	-	-

Table 7.3: Results for the case with $m = 100$

	Cutting-set		Cutting-set with constraint aggregation		Dual-Subgradient with backtracking LS		
n	Iter.	Time(s)	Iter.	Time(s)	Converged?	Iter. (when converged)	Time(s) (when converged)
50	3	**3.95**	8	4.89	0	-	-
100	2.8	12.03	9.5	**10.7**	0	-	-
200	3	38.4	13	**27.6**	0	-	-
400	2.8	165.0	16.5	**123.4**	0	-	-
600	2.2	1266.82	21.2	**475.8**	0	-	-

Table 7.4: Results for the case with $m = 200$

	Cutting-set		Cutting-set with constraint aggregation		Dual-Subgradient with backtracking LS		
n	Iter.	Time(s)	Iter.	Time(s)	Converged?	Iter. (when converged)	Time(s) (when converged)
50	2.2	**5.84**	7.6	8.4	0	-	-
100	3	25.2	10.2	**22.98**	0	-	-
200	2.8	77.4	14.8	**67.2**	0	-	-
400	2.6	665.4	22.4	**272.4**	0	-	-
600	-	-	26.6	**1031.2**	0	-	-

Table 7.5: Results for the case with $m = 400$

	Cutting-set		Cutting-set with constraint aggregation		Dual-Subgradient with backtracking LS		
n	Iter.	Time(s)	Iter.	Time(s)	Converged?	Iter. (when converged)	Time(s) (when converged)
50	2	26.4	10.2	**25.7**	0	-	-
100	2.4	**46.4**	14.6	50.6	0	-	-
200	3	183.8	16.0	**102.6**	0	-	-
400	-	-	25.2	**508.6**	0	-	-
600	-	-	29.2	**1673.1**	0	-	-

Table 7.6: Results for the case with $m = 600$

8 Conclusions

We started this work with a motivation to consider uncertainty in optimization problems. A classic study performed by Ben-Tal and Nemirovski [1] shows that even slight perturbations of about 0.1% on the data served as input for *NETLIB* optimization problems can turn solutions that were previously optimal into highly infeasible points. We then focused on how the Robust Optimization framework allows the practitioner to provide uncertainty sets in which the parameters can vary, and then performs an optimization of the so-called *robust counterpart* of the nominal problem, that finds a solution feasible for *all* possible realizations of such sets.

While the approach that reformulates the problem into its robust counterpart has been successfully applied in diverse fields, it can suffer from a serious drawback: it, in general, belongs to a different class of optimization problems than that of the nominal problem, and its tractability heavily depends on the shape of the uncertainty sets used. We have seen that the robust counterpart of a QCQP with a single ellipsoidal uncertainty, for example, becomes an SDP. This drawback became very clear in Section 7.2, in which we have shown that the reformulation into an SDP failed to find a solution for the robust QCQP problem in all but the smallest instances we generated, because the problem becomes too big and the solver runs out of memory.

This fact motivated the idea of approximately solving robust problems with iterative methods, as seen in works such as [33,38,39]. Their goal is to find solutions that are approximately robust feasible without the need to solve the robust counterpart. While Mutapcic and Boyd's [38] cutting-set method only has guarantees that are exponential in n, Ben-Tal et al.'s Dual-Subgradient method [33] and Ho-Nguyen and Kilinc-Karzan's Online First Order method [39] were able to use concepts from the Online Convex Optimization framework to achieve tighter guarantees on the required number of rounds.

After providing an overview of the Online Convex Optimization framework on Chapter 3 (and, more specifically, the OGD on Section 3.2.3 and the MWU on Sections 3.2.4 and 3.3), we developed the MWU-Based Robust Optimization algorithm on Chapter 5. This alternative algorithm takes Ben-Tal et al.'s idea [33] further, as we mix concepts from the Dual-Subgradient method and the MWU-Based CP (Section 3.3) to **prove** that instead of solving in each iteration the whole nominal problem, we can solve a simplified version that aggregates all the uncertain constraints into one, and we do not require much more iterations for convergence. In fact, our algorithm needs essentially the maximum required number of iterations between the Dual-Subgradient method and the MWU application on Convex Programming, up to a small constant factor. This alternative can be useful when the nominal problem has many constraints and solving it in each iteration can be too costly.

Then, based on the empirical observation that Mutapcic and Boyd's cutting-set method seems to generally outperform the other two approaches we studied, despite having worse guarantees, on Chapter 6 we developed a different version of our algorithm that is more suited for practical cases. Unfortunately, it loses the theoretical guarantees proved in Chapter 5, but is able to achieve interesting computational results when the nominal problem is itself too costly and when some of the constraints are more slack than others.

We performed two computational experiments: one is approximately solving the robust version of linear programming problems from the *NETLIB* library. In it, we show that while Mutapcic and Boyd's cutting-set algorithm shines (because the nominal problem can be solved very efficiently, even with relatively many constraints), we can already see the motivation for our optimized algorithm: the biggest version of the nominal problem we have to solve, in terms of number of constraints, is typically much smaller than that of other methods. Our second experiment is better suited to illustrate this advantage. Since we work with the robust version of a QCQP with many constraints which have naturally different levels of slackness, the nominal problem is itself not easy to solve, and the main advantage of our method can be observed in the results, since the algorithm can be able to converge without the need to solve modified versions of the nominal problem that are too big.

There are several possible paths for future works. One is to apply the algorithms introduced in this work to different problems, ideally ones that display favourable characteristics for our practical algorithm: when the nominal problem is not easy to solve, and the constraints have different levels of slackness. Some ideas are:

- Robust SDP problems. Solving this class of problems with iterative algorithms has been discussed in [33]. This setting could be interesting the our Cutting-set with constraint aggregation method, precisely because the an SDP with too many constraints can be too costly.

- In [23], Bertsimas et al. show how to apply concepts from Statistics in order to use available data to design uncertainty sets for robust problems. As they discuss, for some of the sets their formulation involves complex nonlinear constraints, such as exponential cone constraints. Since a direct optimization can be challenging in these situations, iterative methods such as [38] are recommended by the authors. Our algorithm could be a good option in this situation, as it often requires solving smaller problems than other iterative algorithms.
- Works such as [65] and [66] employ ideas from Mutapcic and Boyd's Cutting-set method [38] on high-dimensional machine learning problems. A method with constraint-aggregation as we suggested could help the computational performance in these high-dimensional settings.
- Specific robust QCQP problems. While our method was successful on the QCQP instances we tested, it can be interesting to apply it to known QCQP problems in the literature. Some of these are briefly discussed in [60], and include learning the kernel matrix in discriminant analysis [67], finance [68], signal processing [69] and the alignment of kernels in semi-supervised learning [70]

Another idea is to use frameworks such as the (AB)-Prod presented by Sani et al. [71] to improve the practical performance of the theoretical algorithm while maintaining guarantees. Other possible research questions have been suggested. Ben-Tal et al. [33] discuss that one interesting contribution would be to remove the need for convexity in algorithms such as the Dual-Subgradient [33] and the OFO [39]. Currently, they need the convexity assumption in order to ensure that the average $\bar{x}$ of the solutions x^t is in the domain $\mathcal{X}$, but if this requirement could be removed in some case, it could open the possibility to solve robust combinatorial problems as well. Finally, both [33] and [39] argue that these iterative approaches could also be adapted to solve multi-stage robust decision problems such as Markov decision process

References

[1] A. Ben-Tal and A. Nemirovski, “Robust solutions of linear programming problems contaminated with uncertain data,” *Mathematical programming*, vol. 88, no. 3, pp. 411–424, 2000.

[2] A. Saltelli, S. Tarantola, F. Campolongo, and M. Ratto, “Sensitivity analysis in practice: a guide to assessing scientific models,” *Chichester, England*, 2004.

[3] J. M. Mulvey, R. J. Vanderbei, and S. A. Zenios, “Robust optimization of large-scale systems,” *Operations research*, vol. 43, no. 2, pp. 264–281, 1995.

[4] G. B. Dantzig, “Linear programming under uncertainty,” in *Stochastic programming*. Springer, 2010, pp. 1–11.

[5] G. Infanger, “Planning under uncertainty solving large-scale stochastic linear programs,” Stanford Univ., CA (United States). Systems Optimization Lab., Tech. Rep., 1992.

[6] J. R. Birge and F. Louveaux, *Introduction to stochastic programming*. Springer Science & Business Media, 2011.

[7] P. Kall, S. W. Wallace, and P. Kall, *Stochastic programming*. Springer, 1994.

[8] A. Ben-Tal and A. Nemirovski, “Robust truss topology design via semidefinite programming,” *SIAM journal on optimization*, vol. 7, no. 4, pp. 991–1016, 1997.

[9] F. J. Fabozzi, P. N. Kolm, D. A. Pachamanova, and S. M. Focardi, *Robust portfolio optimization and management*. John Wiley & Sons, 2007.

[10] L. E. Ghaoui, M. Oks, and F. Oustry, “Worst-case value-at-risk and robust portfolio optimization: A conic programming approach,” *Operations research*, vol. 51, no. 4, pp. 543–556, 2003.

[11] D. Goldfarb and G. Iyengar, “Robust portfolio selection problems,” *Mathematics of operations research*, vol. 28, no. 1, pp. 1–38, 2003.

[12] D. Bienstock and N. ÖZbay, “Computing robust basestock levels,” *Discrete Optimization*, vol. 5, no. 2, pp. 389–414, 2008.

[13] D. Bertsimas and A. Thiele, "A robust optimization approach to supply chain management," in *International Conference on Integer Programming and Combinatorial Optimization*. Springer, 2004, pp. 86–100.

[14] A. B. Tal, B. Golany, A. Nemirovski, and J.-P. Vial, "Supplier-retailer flexible commitments contracts: A robust optimization approach," 2003.

[15] A. Fredriksson, A. Forsgren, and B. Hårdemark, "Minimax optimization for handling range and setup uncertainties in proton therapy," *Medical physics*, vol. 38, no. 3, pp. 1672–1684, 2011.

[16] X. J. Wang and D. J. Curry, "A robust approach to the share-of-choice product design problem," *Omega*, vol. 40, no. 6, pp. 818–826, 2012.

[17] S. Yan and C.-H. Tang, "Inter-city bus scheduling under variable market share and uncertain market demands," *Omega*, vol. 37, no. 1, pp. 178–192, 2009.

[18] Y. C. Eldar, A. Ben-Tal, and A. Nemirovski, "Robust mean-squared error estimation in the presence of model uncertainties," *IEEE Transactions on Signal Processing*, vol. 53, no. 1, pp. 168–181, 2005.

[19] L. El Ghaoui and H. Lebret, "Robust solutions to least-squares problems with uncertain data," *SIAM Journal on matrix analysis and applications*, vol. 18, no. 4, pp. 1035–1064, 1997.

[20] H. Xu, C. Caramanis, and S. Mannor, "Robustness and regularization of support vector machines," *Journal of Machine Learning Research*, vol. 10, no. Jul, pp. 1485–1510, 2009.

[21] ——, "Robust regression and lasso," in *Advances in Neural Information Processing Systems*, 2009, pp. 1801–1808.

[22] Z. Wang, P. W. Glynn, and Y. Ye, "Likelihood robust optimization for data-driven problems," *Computational Management Science*, vol. 13, no. 2, pp. 241–261, 2016.

[23] D. Bertsimas, V. Gupta, and N. Kallus, "Data-driven robust optimization," *Mathematical Programming*, vol. 167, no. 2, pp. 235–292, 2018.

[24] D. Bertsimas, D. B. Brown, and C. Caramanis, "Theory and applications of robust optimization," *SIAM review*, vol. 53, no. 3, pp. 464–501, 2011.

[25] B. L. Gorissen, İ. Yanıkoğlu, and D. den Hertog, "A practical guide to robust optimization," *Omega*, vol. 53, pp. 124–137, 2015.

[26] A. Ben-Tal and A. Nemirovski, "Robust optimization–methodology and applications," *Mathematical Programming*, vol. 92, no. 3, pp. 453–480, 2002.

[27] Z. Li and C. A. Floudas, "Robust counterpart optimization: Uncertainty sets, formulations and probabilistic guarantees," in *proceedings of the 6th conference on foundations of computer-aided process operations, Savannah (Georgia)*, 2012.

[28] A. L. Soyster, "Convex programming with set-inclusive constraints and applications to inexact linear programming," *Operations research*, vol. 21, no. 5, pp. 1154–1157, 1973.

[29] A. Ben-Tal and A. Nemirovski, "Robust solutions of uncertain linear programs," *Operations research letters*, vol. 25, no. 1, pp. 1–13, 1999.

[30] ——, "Robust convex optimization," *Mathematics of operations research*, vol. 23, no. 4, pp. 769–805, 1998.

[31] L. El Ghaoui, F. Oustry, and H. Lebret, "Robust solutions to uncertain semidefinite programs," *SIAM Journal on Optimization*, vol. 9, no. 1, pp. 33–52, 1998.

[32] T. Dokka and M. Goerigk, "An experimental comparison of uncertainty sets for robust shortest path problems," *arXiv preprint arXiv:1704.08470*, 2017.

[33] A. Ben-Tal, E. Hazan, T. Koren, and S. Mannor, "Oracle-based robust optimization via online learning," *Operations Research*, vol. 63, no. 3, pp. 628–638, 2015.

[34] D. Bertsimas and M. Sim, "Robust discrete optimization and network flows," *Mathematical programming*, vol. 98, no. 1-3, pp. 49–71, 2003.

[35] S. Sra, S. Nowozin, and S. J. Wright, *Optimization for machine learning*. Mit Press, 2012.

[36] D. Bertsimas, I. Dunning, and M. Lubin, "Reformulation versus cutting-planes for robust optimization," *Computational Management Science*, vol. 13, no. 2, pp. 195–217, 2016.

[37] S. Arora, E. Hazan, and S. Kale, "The multiplicative weights update method: a meta-algorithm and applications," *Theory of Computing*, vol. 8, no. 1, pp. 121–164, 2012.

[38] A. Mutapcic and S. Boyd, "Cutting-set methods for robust convex optimization with pessimizing oracles," *Optimization Methods & Software*, vol. 24, no. 3, pp. 381–406, 2009.

[39] N. Ho-Nguyen and F. Kılınç-Karzan, "Online first-order framework for robust convex optimization," *Operations Research*, vol. 66, no. 6, pp. 1670–1692, 2018.

[40] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[41] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," in *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 2003, pp. 928–936.

[42] N. Littlestone and M. K. Warmuth, "The weighted majority algorithm," *Information and computation*, vol. 108, no. 2, pp. 212–261, 1994.

[43] Y. Freund and R. E. Schapire, "Adaptive game playing using multiplicative weights," *Games and Economic Behavior*, vol. 29, no. 1-2, pp. 79–103, 1999.

[44] T. Lattimore and C. Szepesvári, "Bandit algorithms," *preprint*, 2018.

[45] S. Shalev-Shwartz *et al.*, "Online learning and online convex optimization," *Foundations and Trends® in Machine Learning*, vol. 4, no. 2, pp. 107–194, 2012.

[46] S. Shalev-Shwartz and S. M. Kakade, "Mind the duality gap: Logarithmic regret algorithms for online optimization," in *Advances in Neural Information Processing Systems*, 2009, pp. 1457–1464.

[47] E. Hazan, A. Agarwal, and S. Kale, "Logarithmic regret algorithms for online convex optimization," *Machine Learning*, vol. 69, no. 2-3, pp. 169–192, 2007.

[48] D. P. Bertsekas, A. Nedi, A. E. Ozdaglar *et al.*, "Convex analysis and optimization," 2003.

[49] N. Merentes and K. Nikodem, "Remarks on strongly convex functions," *Aequationes mathematicae*, vol. 80, no. 1-2, pp. 193–199, 2010.

[50] E. Hazan, A. Rakhlin, and P. L. Bartlett, "Adaptive online gradient descent," in *Advances in Neural Information Processing Systems*, 2008, pp. 65–72.

[51] H. B. McMahan, "Follow-the-regularized-leader and mirror descent: Equivalence theorems and l1 regularization," 2011.

[52] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[53] S. A. Plotkin, D. B. Shmoys, and É. Tardos, "Fast approximation algorithms for fractional packing and covering problems," *Mathematics of Operations Research*, vol. 20, no. 2, pp. 257–301, 1995.

[54] J. E. Kelley, Jr, "The cutting-plane method for solving convex programs," *Journal of the society for Industrial and Applied Mathematics*, vol. 8, no. 4, pp. 703–712, 1960.

[55] J. Abernethy, K. A. Lai, K. Y. Levy, and J.-K. Wang, "Faster rates for convex-concave games," *arXiv preprint arXiv:1805.06792*, 2018.

[56] C. Kroer, N. Ho-Nguyen, G. Lu, and F. Kılınç-Karzan, "Performance evaluation of iterative methods for solving robust convex quadratic problems."

[57] J. Nocedal and Y.-x. Yuan, "Combining trust region and line search techniques," in *Advances in nonlinear programming*. Springer, 1998, pp. 153–175.

[58] A. Wächter and L. T. Biegler, "Line search filter methods for nonlinear programming: Motivation and global convergence," *SIAM Journal on Optimization*, vol. 16, no. 1, pp. 1–31, 2005.

[59] S. Browne, J. Dongarra, E. Grosse, and T. Rowan, "The netlib mathematical software repository," *D-lib Magazine*, vol. 1, no. 9, 1995.

[60] K. Basu, A. Saha, and S. Chatterjee, "Large-scale quadratically constrained quadratic program via low-discrepancy sequences," in *Advances in Neural Information Processing Systems*, 2017, pp. 2297–2307.

[61] V. Jeyakumar and G. Li, "Trust-region problems with linear inequality constraints: exact sdp relaxation, global optimality and robust optimization," *Mathematical Programming*, vol. 147, no. 1-2, pp. 171–206, 2014.

[62] F. Furini, E. Traversi, P. Belotti, A. Frangioni, A. Gleixner, N. Gould, L. Liberti, A. Lodi, R. Misener, H. Mittelmann *et al.*, "Qplib: A library of quadratic programming instances," *Mathematical Programming Computation*, vol. 11, no. 2, pp. 237–265, 2019.

[63] X. Bao, N. V. Sahinidis, and M. Tawarmalani, "Semidefinite relaxations for quadratically constrained quadratic programming: A review and comparisons," *Mathematical programming*, vol. 129, no. 1, p. 129, 2011.

[64] S. Bose, D. F. Gayme, K. M. Chandy, and S. H. Low, "Quadratically constrained quadratic programs on acyclic graphs with application to power flow," *IEEE Transactions on Control of Network Systems*, vol. 2, no. 3, pp. 278–287, 2015.

[65] W. Li, L. Duan, D. Xu, and I. W. Tsang, "Learning with augmented features for supervised and semi-supervised heterogeneous domain adaptation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, no. 6, pp. 1134–1148, 2013.

[66] M. Tan, I. W. Tsang, and L. Wang, "Towards ultrahigh dimensional feature selection for big data," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1371–1429, 2014.

[67] J. Ye, S. Ji, and J. Chen, "Learning the kernel matrix in discriminant analysis via quadratically constrained quadratic programming," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2007, pp. 854–863.

[68] M. S. Lobo, "Robust and convex optimization with applications in finance," Ph.D. dissertation, stanford university, 2000.

[69] Y. Huang and D. P. Palomar, "Randomized algorithms for optimal solutions of double-sided qcqp with applications in signal processing," *IEEE Transactions on Signal Processing*, vol. 62, no. 5, pp. 1093–1108, 2014.

[70] X. Zhu, J. Kandola, J. Lafferty, and Z. Ghahramani, "Graph kernels by spectral transforms," *Semi-supervised learning*, pp. 277–291, 2006.

[71] A. Sani, G. Neu, and A. Lazaric, "Exploiting easy data in online optimization," in *Advances in Neural Information Processing Systems*, 2014, pp. 810–818.