Analysis of Mathematical Optimization: Some Classes, Problems, and Algorithms

Liam Wrubleski

April 2020

Abstract

This paper aims to provide an introduction to the field of mathematical optimization requiring little background knowledge. The high-level background and theory of mathematical optimization are discussed, with a focus on convex optimization. The theory and advantages of convex optimization are then discussed. The theory of linear programming is then discussed in detail, as is a simple implementation of Dantzig's simplex algorithm for linear programming problems, with a worked example. The theory of mixed-integer linear programming is then discussed, as well as an overview of the branch-and-cut method used to solve these problems. High-level explanations of quadratic programming, quadratically constrained quadratic programming, second-order cone programming, geometric programming, and semidefinite programming are provided, as are some areas in which each type of program is useful. Finally, in the interest of showing some items outside of convex optimization, the global black-box stochastic optimization algorithm developed by Wang et. al. is explained, although as the provided analysis in the paper is comprehensive no further analysis is provided. All material is presented either with proof where doing so provides useful insight to the beginner, or with references to external proofs where such proofs do not help with a basic understanding.

1 Introduction

Many problems, in mathematics and industry both, involve finding a configuration of items that minimize some cost, maximize some benefit, or both. However, beyond the simplest kinds of problems, this is often very difficult to do, especially in a computationally efficient fashion. The field of mathematical optimization aims to develop methods and algorithms for finding these optimal configurations in a relatively efficient manner.

1.1 Notation

Optimization programs are generally denoted in the following fashion

$$\begin{aligned} &\text{minimize} && f(\mathbf{x}), \\ &\text{subject to} && \mathbf{x} \in S \end{aligned}$$

where S is some set of configurations referred to as the **feasible space** or the **feasible region**, and f is a function $f : S \to \mathbb{R}$ that determines how good a given configuration is. The problem also has a **objective sense**, which is either *maximization* or *minimization*. In the formulation above, the objective sense is minimization. A **solution** to this problem

is generally a value $v^* \in \mathbb{R} \cup \{-\infty, \infty\}$ such that $v^* = \inf\{f(x)|x \in S\}$, or a configuration $x^* \in \text{cl}(S)$ such that $f(x^*) = \inf\{f(x)|x \in S\}$ or $\lim_{x \in S \to x^*} f(x) = \inf\{f(x)|x \in S\}$.

A problem is described as **infeasible** if $S = \emptyset$, so there are no configurations in the feasible region, or if $v^* = \infty$. A problem is described as **unbounded** if $v^* = -\infty$, so for each configuration there is some configuration that performs better. A particular configuration x is **feasible** if and only if $x \in S$, and it is **infeasible otherwise**.

If the feasible space is a non-empty connected set, then the optimization problem is **continuous**. If the feasible space is disconnected, then the problem is **discrete** or **combinatorial**, and are related to **decision problems**, as the algorithm must decide which non-empty open set it should terminate in. Generally the feasible space is assumed to be a subset of $\mathbb{R}^n$, as problems on other feasible spaces can be converted to problems on a subset of $\mathbb{R}^n$.

1.2 Calculus Optimization

Upon introduction to optimization problems, many students will recall from early calculus courses some simple optimization problems solved using the first and second derivatives of a function. For example, a problem that should sound familiar to students who have taken some calculus goes as follows.

Calculus Optimization Example

You are on a beach next to the ocean. A lighthouse is placed in the ocean at the point (100,100), you are standing on the coastline at the point (0,0), and the coastline follows the line $y = 0$. You travel at 3 m/s on the beach, and at 1 m/s in the water. What is the minimum amount of time it takes you to reach the lighthouse?

This is a problem that can quite easily be solved with single-variable calculus, but even slight modifications to this problem become unreasonable to attempt. For example, if the coastline just follows a sloped line instead of a horizontal line, the equations are far beyond the capabilities of an early calculus student. For more complicated problems still, they are intractable to solve by hand. For these, mathematical optimization is the best tool to use.

1.3 No Free Lunch Theorems

In this paper, we discuss only a few optimization algorithms. There are many different algorithms, which are used in different situations, and this is also often confusing to persons who are new to optimization. This is because all algorithms that perform well on certain problems must perform poorly on some other problems. This was shown by Wolpert and Macready in [1], and is known as the "no free lunch" theorem, presented here without proof.

Theorem 1.1. *Given a finite set of configurations V and a finite set $S \subset \mathbb{R}$, assume that $f : V \to S$ is chosen according to the uniform distribution over the set of all functions that map V to S. For the problem of optimizing f over V, no algorithm performs better on average than blind search (which at each step chooses $v \in V$ according to the uniform distribution over all elements of V that haven't been chosen before.)*

This does not generally play into the considerations for optimization algorithms [2], but it does formalize the idea that if an algorithm performs well on some problems then it must

perform poorly on others (so that its average performance is no better than blind search). These problems are often nonsensical, but as discussed in section 3.3, there are algorithms with generally good performance that have poor performance on reasonable problems.

1.4 Notes on conventions, notations, and the standard form

As noted above, the sense of an optimization problem is either maximization or minimization. However, the convention is generally (with the exception of Linear Programming, see section 3) to phrase problems as minimization problems. This is because the maximization problem

$$\begin{array}{ll} \text{maximize} & f(\mathbf{x}), \\ \text{subject to} & \mathbf{x} \in S \end{array}$$

is equivalent[3] to

$$\begin{array}{ll} \text{minimize} & -f(\mathbf{x}), \\ \text{subject to} & \mathbf{x} \in S \end{array}$$

and having a consistent convention of minimization allows for easier algorithm development and explanation.

The feasible region may also be denoted using functions mapping $\mathbb{R}^n$ to $\mathbb{R}$, which are required to have certain values. These are referred to as **constraint functions**, and typically take one of two forms. **Inequality constraints** are typically denoted using the indexed functions $g_i(\mathbf{x}) \leq 0, i = 1, ..., m$, while **equality constraints** are typically denoted using the indexed functions $h_i(\mathbf{x}) = 0, i = 1, ..., p$. Normally, n refers to the number of variables in the problem, m refers to the number of inequality constraints, and p refers to the number of equality constraints. When constraint functions are used, we consider the objective function f to also be defined on $\mathbb{R}^n$. If the maximal domain of f is $\mathbb{R}^n$, then f is used on $\mathbb{R}^n$. If the maximal domain of f is $D \subset \mathbb{R}^n$, then we consider optimizing $\tilde{f} : \mathbb{R}^r \to \mathbb{R} \cup \{+\infty\}$, where

$$\tilde{f}(\mathbf{x}) = \begin{cases} +\infty, & \mathbf{x} \notin D \\ f(\mathbf{x}), & \mathbf{x} \in D \end{cases}$$

and the problem is also infeasible if $v^* = +\infty$. Problems phrased using constraint functions as follows are in **standard form**[3]

$$\begin{array}{lll} \text{minimize} & f(\mathbf{x}) & \\ \text{subject to} & g_i(\mathbf{x}) \leq 0, & i = 1, ..., m \\ & h_i(\mathbf{x}) = 0, & i = 1, ..., p \end{array}$$

The feasible region is then the set $S = \{\mathbf{x} \in \mathbb{R}^n | \forall i = 1, ..., m, g_i(\mathbf{x}) \leq 0, \forall i = 1, ..., p, h_i(\mathbf{x}) = 0\}$.

Although the feasible region does not by the definition above need to include its boundary, for continuous objective functions bounded on the closure of S, $v^* = \inf\{f(x) | x \in \text{cl}(S)\}$ is equivalent to $v^* = \min\{f(x) | x \in \text{cl}(S)\}$, so for the sake of convenience in this paper we assume S contains its boundary (denoted ∂S) and use min.

2 Convex Optimization

2.1 Introduction

Convex optimization encompasses many different problems, and many different kinds of problems. Further sections analyze particular subclasses of convex optimization, but the

analysis would be incomplete without first discussing convex optimization as a general field, so here follows an extremely simplified overview of convex optimization.

2.2 Theory

The foundation of convex optimization is convex sets and convex functions, so below are given the definitions of convex sets and functions.

Definition (Convex Set)**.** *A set $S \subseteq \mathbb{R}^n$ is **convex** if and only if for any two points $\mathbf{x}_1, \mathbf{x}_2 \in S$ and any $a \in [0,1]$, $a\mathbf{x}_1 + (1-a)\mathbf{x}_2 \in S$. [3]*

Definition (Epigraph)**.** *The **epigraph** of a function $f : S \to \mathbb{R}, S \subseteq \mathbb{R}^n$ is the set* epi$f = \{(\mathbf{x}, v) | \mathbf{x} \in S, v \geq f(\mathbf{x})\} \subset \mathbb{R}^{n+1}$*. [3]*

Definition (Convex Function)**.** *A **convex function** is a function $f : S \to \mathbb{R}$, where S is a convex set, such that* epi$f$ *is a convex set. Equivalently, f is a convex function if and only if for any two points $\mathbf{x}_1, \mathbf{x}_2 \in S$ and any $a \in [0,1]$, $f(a\mathbf{x}_1+(1-a)\mathbf{x}_2) \leq af(\mathbf{x}_1)+(1-a)f(\mathbf{x}_1)$.[3]*

Definition (Concave Function)**.** *A **concave function** is a function f such that $-f$ is a convex function.[3]*

Theorem 2.1. *The intersection of two convex sets is a convex set.*

Proof. Take any two convex sets $C_1, C_2 \subseteq \mathbb{R}^n$. Then let $\mathbf{x}_1, \mathbf{x}_2 \in C_1 \cap C_2, a \in [0,1]$. Then $a\mathbf{x}_1 + (1-a)\mathbf{x}_2 \in C_1$, as $\mathbf{x}_1, \mathbf{x}_2 \in C_1$ by construction and C_1 is convex. Similarly, $a\mathbf{x}_1 + (1-a)\mathbf{x}_2 \in C_2$, so $a\mathbf{x}_1 + (1-a)\mathbf{x}_2 \in C_1 \cap C_2$, so $C_1 \cap C_2$ is convex. □

Equipped with these definitions, a convex optimization problem may be defined as follows:

Definition (Convex Optimization Problem)**.** *A **convex optimization problem** is an optimization problem with feasible region S and objective function f of the form*

$$\begin{aligned} &\textit{minimize} && f(\mathbf{x}), \\ &\textit{subject to} && \mathbf{x} \in S \end{aligned}$$

where S is a convex set, and f is a convex function.

Remark. *In order to perform convex optimization with the objective sense being maximization, $-f$ must be convex, so f must be concave.*

To explain why convex optimization is especially useful, several more definitions are required.

Definition (Local Optimum)**.** *A point $\mathbf{x}^* \in \bar{S}$ is a **local optimum** for the optimization problem (S, f) if $\exists \varepsilon > 0 : \forall \mathbf{x} \in \bar{S} : \|\mathbf{x} - \mathbf{x}^*\| < \varepsilon \implies f(\mathbf{x}) \geq f(\mathbf{x}^*)$.*

Definition (Global Optimum)**.** *A point $\mathbf{x}^*$ is a **global optimum** for the optimization problem with feasible region S and objective function f if $\forall \mathbf{x} \in S, f(\mathbf{x}) \geq f(\mathbf{x}^*)$.*

Theorem 2.2. *Any local optimum to a convex optimization problem is a global optimum to that optimization problem.*

Proof. Suppose (S, f) describes a convex optimization problem, and suppose for the sake of contradiction that f has a local optimum $\mathbf{x}_0 \in \bar{S}$ which is not a global optimum of f. Then there exists some point $\mathbf{x}^*$ such that $f(\mathbf{x}^*) < f(\mathbf{x}_0)$. As $\mathbf{x}_0$ is a local minimum of this problem, there exists $\varepsilon > 0$ such that $\forall \mathbf{x} \in \bar{S} : \|\mathbf{x} - \mathbf{x}_0\| < \varepsilon \implies f(\mathbf{x}) \geq f(\mathbf{x}_0)$. Take such an epsilon, then consider $a = \frac{\varepsilon}{2\|\mathbf{x}^* - \mathbf{x}_0\|}$ and $\mathbf{x}_1 = a\mathbf{x}^* + (1-a)\mathbf{x}_0$. As S is a convex set, we know $x_1 \in S$. This gives that $\|\mathbf{x}_1 - \mathbf{x}_0\| = \|(a\mathbf{x}^* + (1-a)\mathbf{x}_0) - \mathbf{x}_0\| = \|a(\mathbf{x}^* - \mathbf{x}_0) + \mathbf{x}_0 - \mathbf{x}_0\| = \frac{\varepsilon\|\mathbf{x}^* - \mathbf{x}_0\|}{2\|\mathbf{x}^* - \mathbf{x}_0\|} = \frac{\varepsilon}{2} < \varepsilon$, so $f(\mathbf{x}_1) \geq f(\mathbf{x}_0)$. However, from the definition of a convex function, we know that $f(\mathbf{x}_1) = f(a\mathbf{x}^* + (1-a)\mathbf{x}_0) \leq af(\mathbf{x}^*) + (1-a)f(\mathbf{x}_0) < af(\mathbf{x}_0) + (1-a)f(\mathbf{x}_0) = f(\mathbf{x}_0)$. So we have that $f(\mathbf{x}_1) \geq f(\mathbf{x}_0)$ and $f(\mathbf{x}_1) < f(\mathbf{x}_0)$, which is a contradiction. □

This property of convex optimization problems makes them very attractive, as it is only required to find a single local minimum in order to globally optimize the function.

As with mathematical optimization in general, the feasible space is usually expressed in standard form using constraint functions[3]. For convex optimization problems, the standard form looks as follows:

$$\begin{array}{ll} \text{minimize} & f(\mathbf{x}) \\ \text{subject to} & g_i(\mathbf{x}) \leq 0, \quad i = 1, ..., m \\ & h_i(\mathbf{x}) = 0, \quad i = 1, ..., p \end{array}$$

where f is a convex function, g_i are all convex functions, and h_i are all affine functions (see section 3). This makes the feasible space

$$S = \left(\bigcap_{i=0}^{m} \text{epi} g_i\right) \cap \left(\bigcap_{i=0}^{p} \{(\mathbf{x}, 0) \in \mathbb{R}^{n+1} | h_i(\mathbf{x}) = 0\}\right) \cap \{(\mathbf{x}, v) \in \mathbb{R}^{n+1} | v \leq 0\}$$

As this is the intersection of convex spaces, it is also convex.

3 Linear Programming

3.1 Introduction

Linear programming is a relatively simple subclass of convex optimization, limiting the ways in which the feasible space can be defined, and which objective functions are permitted to be used. This makes them much easier to solve, as it is possible to take advantage of structure in the problem, but it also limits the real-world problems that can be described. For example, linear programming cannot use Euclidean distance between points as a constraint, which means that it cannot be used to solve problems involving the placement of items.

It should be noted that the word programming in an optimization context does not relate to the idea of computer programming, but rather exists for historical reasons. It largely derives from the work of George B. Dantzig, who developed the simplex algorithm for solving linear programming problems, as he was working for the United States military, which used the term "program" to describe proposed schedules [4]. As his work was seminal in the field, the terminology has become standard.

Definition (Halfspace)**.** *A **halfspace** $H_{\mathbf{v},b} \subset \mathbb{R}^n$ is the union of all points on one side of the $n-1$ dimensional hyperplane defined by $\mathbf{v} \in \mathbb{R}^n \setminus \mathbf{0}_{[n]}, b \in \mathbb{R}$ and all points in that hyperplane: for some vector $\mathbf{v} \in \mathbb{R}^n \setminus \mathbf{0}_{[n]}, b \in \mathbb{R}, S = \mathbf{x} \in \mathbb{R}^n | \mathbf{v} \cdot \mathbf{x} \geq b$.*

Theorem 3.1. *All halfspaces are convex.*

Proof. Take any halfspace $H_{\mathbf{v},b} \subset \mathbb{R}^n$. Take any $\mathbf{x}_1, \mathbf{x}_2 \in H_{\mathbf{v},b}$. Then for any $a \in [0,1], (a\mathbf{x}_1 + (1-a)\mathbf{x}_2) \cdot \mathbf{v} = a\mathbf{x}_1 \cdot \mathbf{v} + (1-a)\mathbf{x}_2 \cdot \mathbf{v} \geq ab + (1-a)b = b$, so we have that $a\mathbf{x}_1 + (1-a)\mathbf{x}_2 \in H_{\mathbf{v},b}$, so $H_{\mathbf{v},b}$ is convex. □

Remark. *An alternative definition of a convex set in $\mathbb{R}^n$ is the intersection of (potentially infinitely many) halfspaces in $\mathbb{R}^n$.*

Linear programming is optimization of a linear objective function in $\mathbb{R}^n$ where the feasible space is the intersection of finitely many halfspaces in $\mathbb{R}^n$. Phrased equivalently, the feasible space is a (possibly unbounded) convex polytope in n dimensions, with faces defined by the bounding hyperplanes of the halfspaces. However, this definition is difficult to use in the construction of an algorithm, so we typically write an LP problem as a constrained problem in $\mathbb{R}^n$, where the constraint functions are all affine.

Definition (Affine function)**.** *An affine function is a function $f : \mathbb{R}^n \to \mathbb{R}$ of the form $f(\mathbf{x}) = \mathbf{c}^T\mathbf{x} + b$, for some $\mathbf{c} \in \mathbb{R}^n, b \in \mathbb{R}$. In other words, an affine function is a linear function plus a constant.*

3.2 Theory

Transforming this definition to the standard form is straightforward. Note that here the convention is to maximize the objective function, which is historical in nature and simplifies some notation later, but is equivalent to a minimization problem as stated earlier.

$$\begin{array}{ll} \text{maximize} & f(\mathbf{x}) = \mathbf{c}^T\mathbf{x} \\ \text{subject to} & g_i(\mathbf{x}) = \mathbf{a}_i^T\mathbf{x} - b_i \leq 0, \quad i = 1, ..., m \\ & h_i(\mathbf{x}) = \mathbf{c}_i^T\mathbf{x} - d_i = 0, \quad i = 1, ..., p \end{array}$$

While this is in standard form as described above, for linear programming problems there is a slight variation to the standard form that is typically used, called the **canonical form** of the LP problem. This is obtained from the standard form using transformations discussed in section 3.3, and appears as follows:

$$\begin{array}{ll} \text{maximize} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{array}$$

where the inequality signs here are comparing vectors elementwise.

3.3 Simplex Algorithm

The simplex algorithm was developed by George B. Dantzig. It is an algorithm for solving LP problems, and has been shown to be fairly efficient in real-world applications, and on random problems. Its worst case performance was shown by Klee and Minty to be exponential time in the number of dimensions, but on real-world problems and in general, it tends to be polynomial time in the number of dimensions. This has been shown in several fashions, which are unfortunately beyond the scope of this analysis. We will explore the specific results obtained from these sources, though, after analyzing the algorithm itself.

The simplex algorithm accepts linear programming problems in canonical form.[5]

$$\begin{array}{ll} \text{maximize} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}_n \end{array}$$

If the problem is not in canonical form, it is transformed into canonical form as follows.[5] If there are any equality constraints $h(\mathbf{x}) = 0$, they are each transformed into two inequality

constraints $h(\mathbf{x}) \leq 0, -h(\mathbf{x}) \leq 0$. If any variable x_i is unbounded, it is replaced with two variables $x_i^+, x_i^- \geq 0$, substituting $x_i = x_i^+ - x_i^-$ in each of the other functions. If any variable x_i is not bounded from below, but it is bounded from above (i.e. $x_i \leq b_i$), then substitute $x_i = b_i - y_i$, with $y_i \geq 0$. Finally, if any variable x_i is bounded from below, but not by 0 (i.e. $x_i \geq k_i \neq 0$), then substitute $x_i = y_i + k_i$, with $y_i \geq 0$.

It is also worth discussing the matrix inequality. If we consider the original inequality constraints $g_i(\mathbf{x}) = \mathbf{a}_i^T\mathbf{x} - b_i$, then we construct the matrix A and the vector $\mathbf{b}$ from these constraints by setting the i-th row of A to be $\mathbf{a}_i^T$, and the i-th element of $\mathbf{b}$ to be b_i, so that using the elementwise inequality $A\mathbf{x} - b \leq \mathbf{0}_m$ encapsulates the same information as the original inequality constraints. Finally, we rearrange to obtain $A\mathbf{x} \leq \mathbf{b}$.

When given an LP problem in canonical form, the simplex algorithm first begins by taking each inequality constraint encapsulated in the matrix A and the vector $\mathbf{b}$, and converting it to an equality constraint using a **slack variable**. This variable does not appear in the objective function, and only appears in one of the inequality constraints. For the inequality constraint $g_i(\mathbf{x}) = \mathbf{a}_i^T\mathbf{x} - b_i \leq 0$, we add the slack variable s_i so that we obtain $\mathbf{a}_i^T - b_i + s_i = 0, s_i \geq 0$.

Remark. *It may seem counter intuitive to convert each equality constraint to two inequality constraints, and then to convert each inequality constraint to an equality constraint. Indeed, as touched on in section 3.3.2, the initial equality constraints may be used. Here we perform this conversion so that we only have inequality constraints, which simplifies our explanation. For further information, see section 3.3.2.*

Using these slack variables, the constraints then become $A\mathbf{x} + \mathbf{s} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}_n, \mathbf{s} \geq \mathbf{0}_m$, while the objective function takes the form $\mathbf{c}^T\mathbf{x} + \mathbf{0}_m^T\mathbf{s}$. The algorithm then arranges the problem into a particular kind of matrix called a **tableau**:

$$\begin{bmatrix} 1 & \mathbf{c}^T & \mathbf{0}_m^T & 0 \\ 0 & A & I_m & \mathbf{b}. \end{bmatrix} \tag{1}$$

Remark. *This method of construction of the tableau implicitly assumes that $\mathbf{x} = \mathbf{0}_n$ is a feasible solution, which is not necessarily true. If this does not hold, then the initial tableau must be constructed in an alternative fashion, which is to run the simplex algorithm on a modified version of the problem. This is explained in section 3.3.2.*

The algorithm then performs operations called **pivots** on this tableau to obtain possible solutions[6]. At any given point during the operation of the algorithm, the variables corresponding to columns of the matrix which are columns of the $m + 1$ dimensional identity matrix are called **basic variables**, and all other variables are **nonbasic variables**. All nonbasic variables are 0, and basic variables take on values from the rightmost column (i.e. if the column corresponding to x_i has zeroes in every row except row $j > 1$, and has value 1 in row j, then $x_i = b_{j-1}$). The first column corresponds to the value of the objective function $Z = -f(\mathbf{x})$, or $Z + f(\mathbf{x}) = 0$.

3.3.1 Pivot Operations

With an understanding of the current solution given a configuration of the tableau, we can now explain the pivot operation. First, recall from the introduction of LPs that the feasible space is a convex polytope. The method of construction of the initial tableau ensures that the initial solution always sits on a vertex of this polytope, and pivot operations move the

algorithm from one feasible vertex of the polytope to another. This algorithm works because if the feasible space is not empty, and the objective function has a maximum on any point in the feasible region, then there exists at least one vertex of the polytope at which the objective function takes on its maximum.

Lemma 3.1. *If a nonzero linear function $f : \mathbb{R}^n \to \mathbb{R}$ takes on a maximum M at a point $\mathbf{x}^*$ in a space S, then $\mathbf{x} \in \partial S$.*

Proof. Suppose $f(\mathbf{x}^*) = \max_{\mathbf{x}\in S}\{f(\mathbf{x})\}$ (i.e. f takes on its maximum over S at $\mathbf{x}^*$). Note that as f is linear and nonzero, there exists some $\mathbf{c} \in \mathbb{R}^n \setminus \mathbf{0}_n$ such that $f(\mathbf{x}) = \mathbf{c}^T\mathbf{x}$. Then for any $\varepsilon > 0, f(\mathbf{x}^* + \frac{\varepsilon}{\|\mathbf{c}\|}\mathbf{c}) = \mathbf{c}^T(\mathbf{x}^* + \frac{\varepsilon}{\|\mathbf{c}\|}\mathbf{c}) = \mathbf{c}^T\mathbf{x}^* + \frac{\varepsilon}{\|\mathbf{c}\|}\|\mathbf{c}\|^2 = \mathbf{c}^T\mathbf{x}^* + \varepsilon\|\mathbf{c}\| > \mathbf{c}^T\mathbf{x}^*$, so we have that $\mathbf{x}^* + \frac{\varepsilon}{\|\mathbf{c}\|}\mathbf{c} \notin S$, as $\mathbf{x}^*$ is the maximum over S. However, $\mathbf{x}^* + \frac{\varepsilon}{\|\mathbf{c}\|}\mathbf{c} \in N_\varepsilon(\mathbf{x}^*)$, so $\forall\varepsilon > 0, \exists\mathbf{x} \in N_\varepsilon(\mathbf{x}^*) : \mathbf{x} \notin S$, and $\mathbf{x}^* \in S$, so we have that $\mathbf{x}^*$ is a boundary point of S, so $\mathbf{x}^* \in \partial S$. □

Theorem 3.2 (Fundamental theorem of linear programming)**.** *If a linear programming problem in canonical form with objective function f has a maximum over its feasible region, then f takes on that maximum for at least one vertex of the feasible region.*

Proof. If the objective function $f(\mathbf{x}) = 0$, then the function takes on its maximum at all points in the feasible region. As the feasible region is a nonempty polytope, it has at least one vertex, so f takes on its maximum at at least one vertex of the feasible region.

If the objective function $f(\mathbf{x}) = \mathbf{c}^T\mathbf{x}, \mathbf{c} \neq \mathbf{0}_n$, then by lemma 3.1, we know that $\exists\mathbf{x}_0$ in the boundary of the feasible region that maximizes f. Then $\mathbf{x}_0$ must lie in some "hyperface" of the feasible region (i.e. in 3 dimensions, it can lie in a 0-face (vertex), 1-face (edge), or 2-face (face)). Let F represent the lowest dimensional face of the feasible region that $\mathbf{x}_0$ lies in, and j represent the dimension of F. If j is 0, then $\mathbf{x}_0$ is a vertex of the feasible region. Otherwise, if $j > 0$, then there exist j orthogonal vectors $\mathbf{v}_i$ such that for each of these vectors, there exists some $\varepsilon_i > 0 : 0 < \delta < \varepsilon \implies \mathbf{x}_0 + \delta\mathbf{v}_i \in F \wedge \mathbf{x}_0 - \delta\mathbf{v}_i \in F$. As $\mathbf{x}_0$ maximizes f over the entire feasible region, we must then have that $\mathbf{c}^T\mathbf{v}_i = 0$ for each $\mathbf{v}_i$, which implies that F lies entirely in the hyperplane $\{x \in \mathbb{R}^n : \mathbf{c}^T\mathbf{x} = \mathbf{c}^T\mathbf{x}_0\}$. Then, as F has at least one vertex, there is at least one vertex $\mathbf{x}^*$ with $f(\mathbf{x}^*) = f(\mathbf{x}_0)$. □

At any given step, if the slack variable associated with an inequality is a nonbasic variable, then that inequality is currently true as an equality. Additionally, the values of the first row of the tableau are the partial derivatives of the objective function with respect to each variable. A pivot operation takes a nonbasic variable (called the **entering variable**) and makes it into a basic variable, and simultaneously takes a basic variable (called the **leaving variable**) and makes it into a nonbasic variable. It proceeds as follows:

1. If all entries in the first row are negative, the current solution is optimal, so return Z.
2. Otherwise, take c to be the leftmost column with a positive element in the first row [7] (this is the entering variable choice rule, others can be used).
3. If all entries in column c, other than the first one, are negative, the problem is unbounded, so return $+\infty$.
4. Otherwise, let a_{rc} denote the element in row r of column c, and b_r denote the element in row r of the rightmost column of the tableau.

5. Take r to be the row that minimizes b_r/a_{rc}. If multiple rows have the same value, take the first one whose basic variable is the furthest to the left [7] (this is the dropping variable choice rule, others can be used).

6. Divide row r by a_{rc}, then add multiples of row r to each other row in the tableau such that column c is a column of the identity matrix.

The entering variable is the variable associated with column c, and the leaving variable is the basic variable that previously had a 1 in row r. By selecting the row which minimizes the value of b_r/a_{rc}, when the row operations are performed to make column c into a column of the identity matrix, we ensure that the values in the rightmost column always stay positive, ensuring that the resulting solution is still feasible. Another interpretation is that when the variable enters, it goes from 0 to whichever constraint is most tightly bounding the value of that variable, which ensures that the resulting solution is feasible. [4]

3.3.2 Initial Tableau and Solution

The problem arises of how to find an initially feasible vertex of the polytope. If the origin is a feasible point, then it is a feasible vertex, but if the origin is not feasible then a different method of finding a feasible vertex is required. This method also allows for the direct usage of equality constraints. Suppose we are given a linear programming problem for which the origin is not a feasible point after all of the aforementioned transformations, or which is not in canonical form

$$\begin{aligned} &\text{maximize} && \mathbf{c}^T\mathbf{x} \\ &\text{subject to} && A_s\mathbf{x} \leq \mathbf{b}_s \\ & && A_v\mathbf{x} \leq \mathbf{b}_v \\ & && C\mathbf{x} = \mathbf{d} \\ & && \mathbf{x} \geq \mathbf{0}_n \end{aligned}$$

where $A_s, \mathbf{b}_s$ correspond to the m inequality constraints which are satisfied at the origin, $A_v, \mathbf{b}_v$ correspond to the p inequality constraints which are not satisfied at the origin, and C, d correspond to q equality constraints. Then construct the non-canonical tableau

$$\begin{bmatrix} 1 & \mathbf{c}^T & \mathbf{0}_m^T & \mathbf{0}_p^T & 0 \\ \mathbf{0}_m & A_s & I_m & \mathbf{0}_{p,p} & \mathbf{b}_s \\ \mathbf{0}_p & A_v & \mathbf{0}_{m,m} & I_p & \mathbf{b}_v \\ \mathbf{0}_q & C & \mathbf{0}_{m,m} & \mathbf{0}_{p,p} & d \end{bmatrix}$$

This tableau does not correspond to a valid solution to the linear programming problem, as all the inequality constraints which are violated at the origin are violated, and no attempt has been made to address the equality constraints. The first step in solving these issues is that if any entries in the rightmost column are negative, that entire row is negated (this is valid as each row corresponds to an equality). Then, a new, canonical tableau is constructed by adding new virtual artificial variables, and adding a new objective function, which is the sum of these virtual variables. We attempt to minimize this new objective function, so when adding it to the tablea u it is negated. Finally, in order to get the tableau to be canonical, each of the artificial variables is made into a basic variable, by adding its corresponding row

to the first row (this is not shown in the tableau below).

$$\begin{bmatrix} 1 & 0 & \mathbf{0}_n^T & \mathbf{0}_m^T & \mathbf{0}_p^T & -\mathbf{1}_p^T & -\mathbf{1}_q^T & 0 \\ 0 & 1 & \mathbf{c}^T & \mathbf{0}_m^T & \mathbf{0}_p^T & \mathbf{0}_p^T & \mathbf{0}_q^T & 0 \\ 0 & \mathbf{0}_m & A_s & I_m & \mathbf{0}_{p,p} & \mathbf{0}_{p,p} & \mathbf{0}_{q,q} & \mathbf{b}_s \\ 0 & \mathbf{0}_p & A_v & \mathbf{0}_{m,m} & I_p & I_p & \mathbf{0}_{q,q} & \mathbf{b}_v \\ 0 & \mathbf{0}_q & C & \mathbf{0}_{m,m} & \mathbf{0}_{p,p} & \mathbf{0}_{p,p} & I_q & d \end{bmatrix}$$

The simplex method is then run on this canonical tableau, and if this returns a minimum value of 0 then the internally stored tableau is a canonical representation of the original LP problem. Otherwise, if this returns a positive minimum value, then the original LP problem is infeasible (i.e. the feasible space is empty). This is referred to as the $Phase1$ computation.[5]

3.3.3 Worked Example

The above information is difficult to understand without a particular reference, so a simple worked example is presented here. Consider the following problem:

Linear Programming Example Problem

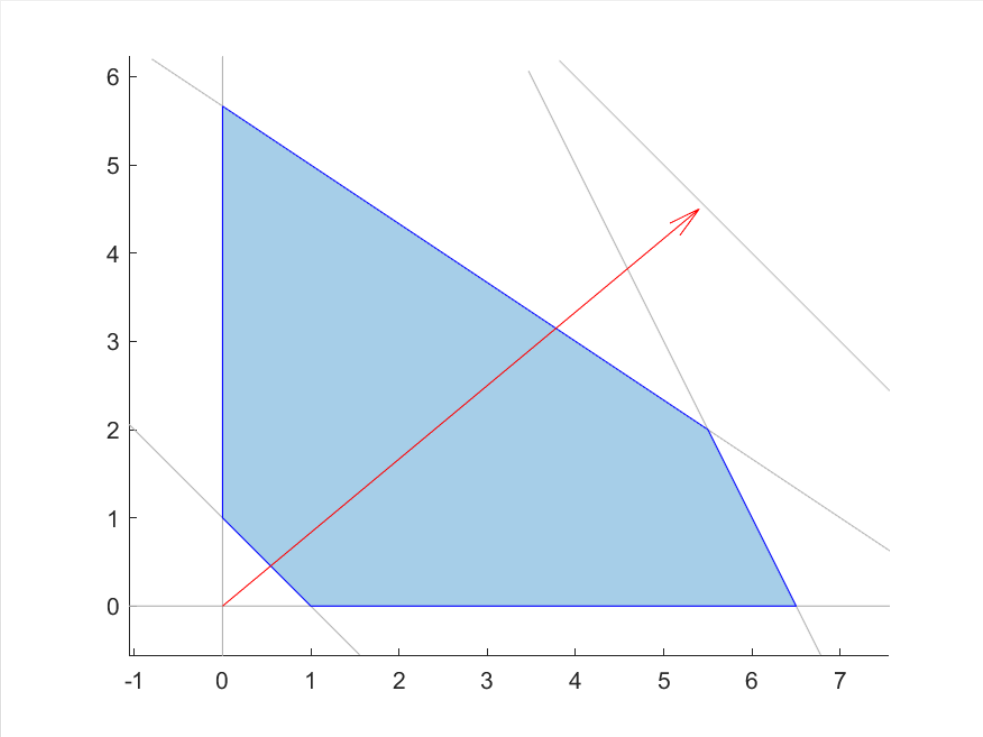
A farmer has 10 square kilometers of land on which she can grow wheat and barley, in any positive combination of the two. She also has 17 kg of fertilizer, and 13 kg of pesticide. For each square kilometer of wheat she grows, she needs 2 kg of fertilizer and 2 kg of pesticide, and she can sell the wheat for $6000 of profit per square kilometer. For each square kilometer of barley she grows, she needs 3 kg of fertilizer, and 1 kg of pesticide, and she can sell the barley for $5000 of profit per square kilometer. Finally, suppose she needs to plant at least 1 square kilometer of crops, regardless of any other consideration, in order to meet her quota. How should she plant these crops to maximize her profit?

Linear Programming Example Evaluation

This problem is obviously contrived, and can be solved quite simply using far simpler methods. However, here we go through the entire simplex method for the purpose of illustration. First, convert the constraints set using words to inequalities, letting x represent the amount of land allocated to wheat in km^2, and y represent the amount of land allocated to barley in km^2. Then we obtain the following constraints:

1. The total amount of land available is 10 square kilometers $\implies x + y \leq 10$.
2. The total amount of fertilizer is 17 kilograms $\implies 2x + 3y \leq 17$.
3. The total amount of pesticide is 13 kilograms $\implies 2x + y \leq 13$.
4. She needs to plant at least 1 km^2 of crops $\implies x + y \geq 1$.
5. She wants to maximize her profit (in thousands of dollars) $\implies$ maximize $6x + 5y$.

Here we see the feasible space for this problem pictured, where the blue region is the feasible space, the blue line is its boundary, and the grey lines are the inequalities. The red arrow is the direction of maximization.

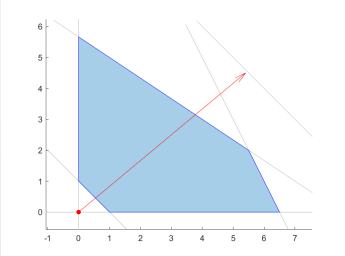


Adding slack variables and rearranging, we obtain the equalities

1. $x + y + s_1 = 10$
2. $2x + 3y + s_2 = 17$
3. $2x + y + s_3 = 13$
4. $-x - y + s_4 = -1$

We can then arrange these equalities into a tableau as described above. Note that here we include the variable names as the first row of the matrix, but these are not actual elements of the matrix. They are listed only to help the reader keep track of variables through different iterations. Not also that when a pivot operation is performed, the entering and leaving elements are bolded. This does not, in this context, correspond to vectors.

$$\begin{bmatrix} Z & x & y & s_1 & s_2 & s_3 & s_4 & b \\ 1 & 6 & 5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 10 \\ 0 & 2 & 3 & 0 & 1 & 0 & 0 & 17 \\ 0 & 2 & 1 & 0 & 0 & 1 & 0 & 13 \\ 0 & -1 & -1 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$



While this tableau is a canonical tableau, it does not correspond to a valid solution, as we have s_4 as a basic variable, with a value of -1, so this is not a feasible configuration. We can see this in the image, as the red point representing our current solution does not lie within the feasible region. Although it is trivial to guess where we could place the starting point, we go through the Phase 1 computation for the sake of illustration. We begin by negating the last row, then adding an artificial variable u to it, giving us the following canonical tableau, upon which we begin to execute the simplex method

$$\begin{bmatrix} W & Z & x & y & s_1 & s_2 & s_3 & s_4 & u & b \\ 1 & 0 & \mathbf{1} & 1 & 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 1 & 6 & 5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 10 \\ 0 & 0 & 2 & 3 & 0 & 1 & 0 & 0 & 0 & 17 \\ 0 & 0 & 2 & 1 & 0 & 0 & 1 & 0 & 0 & 13 \\ 0 & 0 & \mathbf{1} & 1 & 0 & 0 & 0 & -1 & 1 & 1 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} W & Z & x & y & s_1 & s_2 & s_3 & s_4 & u & b \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 6 & -6 & -6 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -1 & 9 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 2 & -2 & 15 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 2 & -2 & 11 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & -1 & 1 & 1 \end{bmatrix}$$

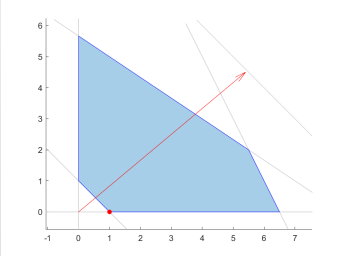
As all entries in the first row, except for the first, are negative, the algorithm returns with a minimum of 0. This means that it has found a suitable starting location, which is encoded in the internal tableau.

$$\begin{bmatrix} Z & x & y & s_1 & s_2 & s_3 & s_4 & b \\ 1 & 0 & -1 & 0 & 0 & 0 & 6 & -6 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 9 \\ 0 & 0 & 1 & 0 & 1 & 0 & 2 & 15 \\ 0 & 0 & -1 & 0 & 0 & 1 & 2 & 11 \\ 0 & 1 & 1 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}$$

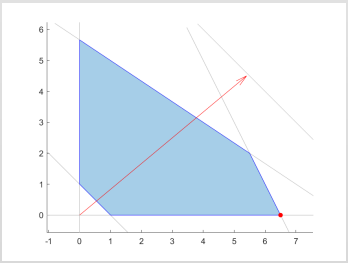
Examining the tableau, we can see that x is a basic variable with a value of 1, and y is a nonbasic variable, so it has a value of 0, and we can see that this is a vertex of the feasible region. Now with a feasible starting point in a canonical tableau, we

run the simplex method again.

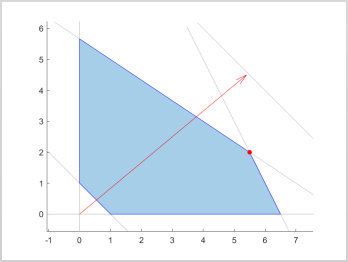
$$\begin{bmatrix} Z & x & y & s_1 & s_2 & s_3 & s_4 & b \\ 1 & 0 & -1 & 0 & 0 & 0 & \mathbf{6} & -6 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 9 \\ 0 & 0 & 1 & 0 & 1 & 0 & 2 & 15 \\ 0 & 0 & -1 & 0 & 0 & 1 & \mathbf{2} & 11 \\ 0 & 1 & 1 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}$$



$$\begin{bmatrix} Z & x & y & s_1 & s_2 & s_3 & s_4 & b \\ 1 & 0 & \mathbf{2} & 0 & 0 & -3 & 0 & -39 \\ 0 & 0 & 1/2 & 1 & 0 & -1/2 & 0 & 7/2 \\ 0 & 0 & \mathbf{2} & 0 & 1 & -1 & 0 & 4 \\ 0 & 0 & -1/2 & 0 & 0 & 1/2 & 1 & 11/2 \\ 0 & 1 & 1/2 & 0 & 0 & 1/2 & 0 & 13/2 \end{bmatrix}$$



$$\begin{bmatrix} Z & x & y & s_1 & s_2 & s_3 & s_4 & b \\ 1 & 0 & 0 & 0 & -1 & -2 & 0 & -43 \\ 0 & 0 & 0 & 1 & -1/4 & -1/4 & 0 & 5/2 \\ 0 & 0 & 1 & 0 & 1/2 & -1/2 & 0 & 2 \\ 0 & 0 & 0 & 0 & 1/4 & 1/4 & 1 & 13/2 \\ 0 & 1 & 0 & 0 & -1/4 & 3/4 & 0 & 11/2 \end{bmatrix}$$



The end result is $x = 11/2 = 5.5, y = 2$, giving the farmer a total profit of \$43,000.

3.4 Efficiency and Uses

The simplex algorithm is fairly efficient, and on average is polynomial time in the number of dimensions.[8] We can see that the algorithm cannot do better than polynomial time quite simply, by noting that for a single pivot operation in n dimensions with m inequality constraints, we require 1 row division for the leaving row, followed by m row multiplications and additions to make the entering variable a basic variable. Assuming each of multiplication and division are constant time per element, this gives a total of $n+m+2+2m(n+m+2) = (2m+1)(n+m+2)$ operations per pivot, which is polynomial in the number of dimensions / constraints.

Remark. *By using a mechanism called the* ***dual problem****, an equivalent problem can be*

constructed that has m variables and n constraints, so distinguishing between them is not particularly useful in this analysis.

However, it is the number of pivot operations needed that causes issues in complexity analysis. Presented above is one of the simplest rules for selection of entering and leaving variables (this is Bland's rule for entering[7]). This rule is quite inefficient in practice[9], and other rules with much better performance exist, such as the Devex rule[10]. In general, the simplex algorithm requires a number of pivots cubic in the number of dimensions[11]. However, it was shown by Klee and Minty that there is an infinite family of problems for which the simplex algorithm visits each of the 2^n vertices before terminating under the standard selection rules[12]. In n dimensions, the Klee-Minty cube is the linear programming problem

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^{n} 2^{n-i} x_i \\ \text{subject to} & (\sum_{i=1}^{k} 2^{k-i+1} x_i) - x_k \leq 5^k \quad k = 1, ..., n \\ & x_k \geq 0 \qquad\qquad\qquad\qquad\quad k = 1, ..., n \end{array}$$

[12] Since this was proven, it has been shown that for any selection rule, there is a family of problems on which the simplex algorithm visits every vertex. This means that the worst case performance of the simplex algorithm is exponential in the number of dimensions, but several subsequent analyses have shown that the problems on which the simplex algorithm performs very poorly are in some sense "rare". [13]

It should also be noted that the simplex algorithm as shown above is very space inefficient [14], as many real-world problems have thousands or millions of variables and constraints[15], rather than the 3 or 4 we have shown. Furthermore, in most industry applications, the matrix is sparse, so in industry the actual algorithm used is the revised simplex algorithm[14], which stores a sparse and invertible representation of the nonbasic variables, which reduces both the required storage space and the computational overhead.

The simplex algorithm is often used in applications where the variables in question are continuous, or are discrete but sufficiently large that a small error is acceptable (e.g. if one is planning the division of $1,000,000,000 to military departments, treating every dollar discretely is probably unnecessary, and a continuous approximation will suffice).

4 Mixed Integer Linear Programming

4.1 Introduction

While many problems have convenient continuous approximations, many others do not. For example, in scheduling contexts, where each person being scheduled has a fixed amount of time that they can contribute, and they cannot be doing two things at once. In these instances, we require that some variables have values that are either integers, or sufficiently close to integer values that the error is truly negligible. However, this problem is significantly harder than LP, which is already not trivial. Fully integer linear programming is NP-complete (and so NP-hard). In fact, even fully integer satisfiability (finding a single point satisfying all of the constraints) is NP-complete, and the special case where the variables are constrained to be binary is one of Karp's 21 NP-complete problems (proof of NP-hardness is shown by Karp via reduction from boolean satisfiability, which is known to be NP-complete)[16], [17].

For scale on how difficult these problems are, consider a scheduling problem assigning 10 people to 10 items. There are some constraints on how these assignments can occur and

some objective function, but for the moment we are interested in the worst case performance. Using a naïve implementation of this problem gives 100 binary variables, one per possible person-item assignment. If we brute-force checked every possible configuration with the world's fastest supercomputer (at time of writing the Summit supercomputer in the Oak Ridge National Laboratory), with 200 PFLOPS, and supposing that checking each potential assignment takes only one floating point operation, it would take over 200,000 years to complete. If we assume the fastest possible computer, checking one potential assignment each Planck time, this would take about 70 femtoseconds. However, if we increase the number of people and items to 15 each, for a total of 225 variables, then it would take over 92 quadrillion years. This is almost 7 million times the age of the universe.

This assumes that we are just attempting to brute force the computation, which is obviously not a very good way to do it. However, many of the best methods known to mathematicians currently devolve to brute force. As the problem is NP-complete, it's intractable to even verify a solution is optimal without checking each possible solution.

Here we discuss one of the techniques observed to have reasonable performance on real-world problems, known as branch-and-cut[18]. The exact operation of the branch-and-cut algorithm in commercial solvers is, unfortunately, a trade secret, so we discuss only the general technique.

Remark. *Note that MILP is not convex, as an affine combination of two integer solutions will give non-integer solutions, so the feasible space is not convex due to the integrality constraints.*

4.2 Theory

Mixed Integer Linear Programming problems take the form

$$\begin{aligned} &\text{maximize} && \mathbf{c}^T\mathbf{x} \\ &\text{subject to} && A\mathbf{x} \leq \mathbf{b} \\ & && \mathbf{x} \geq 0 \\ & && x_i \in \mathbb{Z}, \quad i = 1, ..., p \leq n \end{aligned}$$

The branch-and-cut technique is a combination of two simpler techniques, branch-and-bound and cutting planes[18]. Branch-and-bound increases the lower bound for the possible solution, and cutting planes decrease the upper bound. The lower bound is always initialized to $-\infty$, and the upper bound is set to the optimum for the relaxation of the problem. The relaxation is the linear programming problem obtained by simply ignoring the integrality constraints. Any solution satisfying the integrality constraints is going to be no better than the optimum for the relaxation.

4.2.1 Branch and bound

The branch-and-bound technique explained here takes inspiration from binary search algorithms, and goes as follows[19]:

1. Push the relaxation of the MILP problem to the queue.
2. Loop until the queue is empty:
   (a) Take a problem from the queue, and solve.
   (b) If the value of this problem is equal to the current upper bound, then return and exit.

(c) If the value of this problem is worse than the current lower bound, discard this problem, then continue to the next loop.

(d) Otherwise, if the solution for this problem satisfies the integrality constraint, store this solution, and set the current lower bound to the value for this problem, then continue to the next loop.

(e) Otherwise (noting the solution is better than the current lower bound, but doesn't satisfy the integrality constraint), take one of the variables that should be integral, but isn't. Call this variable x and its value in the solution to this problem x', then create two new problems, one with $x \geq \lceil x' \rceil$, and one with $x \leq \lfloor x' \rfloor$ and push them to the queue.

4.2.2 Cutting planes

The cutting planes method improves on the best known upper bound, by creating an additional linear constraint that removes the current solution without removing any potential integer solutions. In so doing, it removes whichever point previously produced the best result, forcing the solver to find a different point. This point should be slightly worse than the last point, thus the upper bound can be adjusted downward. If the solver ever finds an integer point by chance, then return that result, as that is guaranteed to be an integer point maximizing the MILP problem.

The cuts used are called Gomory cuts, and they were designed for fully integer problems. The adaptation to mixed integer problems is complex at best, and does not lend itself to intuitive understanding, so they will not be deeply explored here. For further information, see [20]

4.2.3 Branch and Cut

As the name implies, this is a combination of branch-and-bound and cutting plane techniques. This combination technique runs some number of cutting planes on the problem in step 2(a) of the branch-and-bound algorithm. It also runs some cutting planes on the top-level LP relaxation to improve the upper bound of the integer solution.

As already mentioned, the best methods of balancing these techniques are industry secrets, so a thorough explanation cannot be done here. It should, however, be noted that branch-and-bound devolves to brute force[21], and cutting planes are not particularly effective in finding solutions in and of themselves, but tend to make branch-and-bound better. This shows that even the best known techniques are very difficult.

4.3 Uses

MILP is particularly useful in preference-based scheduling contexts, as the preferences can be continuous, but the schedules themselves are discrete[22][23]. Other logistical decision problems are also easily translated to MILP problems[24], so MILP solvers are very useful.

As MILP is NP-complete, all other NP problems can be converted to a MILP problem in polynomial time, including some well known problems like the knapsack problem and the travelling salesman problem[16].

5 Quadratic Programming

5.1 Introduction

Quadratic programming is a type of convex optimization that generalizes linear programming slightly. It allows the objective function to be a convex quadratic function, although the constraint functions still must be affine functions. This is useful as it allows for types of optimization like ordinary least squares, as well as for optimizing the variance of a random variable (which often arises in portfolio optimization)[3].

5.2 Theory

The objective function f is permitted to take the form $f(\mathbf{x}) = \mathbf{x}^T Q\mathbf{x} + \mathbf{c}^T\mathbf{x}$, where this function is convex. On this type of function, requiring f to be convex is equivalent to requiring Q to be positive semidefinite.[25]

Theorem 5.1. *$f(x) = \mathbf{x}^T Q\mathbf{x} + \mathbf{c}^T\mathbf{x}$ is convex if and only if Q is positive semidefinite.*

Proof. Suppose f is convex, then for any $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^n, a \in [0,1]$:

$$f(a\mathbf{x}_1 + (1-a)\mathbf{x}_2) \leq af(\mathbf{x}_1) + (1-a)f(\mathbf{x}_2)$$

Consider the left hand side. We see

$$\begin{aligned} LHS &= f(a\mathbf{x}_1 + (1-a)\mathbf{x}_2) \\ &= (a\mathbf{x}_1 + (1-a)\mathbf{x}_2)^T Q(a\mathbf{x}_1 + (1-a)\mathbf{x}_2) + \mathbf{c}^T(a\mathbf{x}_1 + (1-a)\mathbf{x}_2) \\ &= a^2\mathbf{x}_1^T Q\mathbf{x}_1 + 2a(1-a)\mathbf{x}_1^T Q\mathbf{x}_2 + (1-a)^2\mathbf{x}_2^T Q\mathbf{x}_2 + a\mathbf{c}^T\mathbf{x}_1 + (1-a)\mathbf{c}^T\mathbf{x}_2 \end{aligned}$$

On the right hand side, we see

$$\begin{aligned} RHS &= af(\mathbf{x}_1) + (1-a)f(\mathbf{x}_2) \\ &= a\mathbf{x}_1^T Q\mathbf{x}_1 + (1-a)\mathbf{x}_2^T Q\mathbf{x}_2 + a\mathbf{c}^T\mathbf{x}_1 + (1-a)\mathbf{c}^T\mathbf{x}_2 \end{aligned}$$

As we had $LHS \leq RHS$, it must be the case that $RHS - LHS \geq 0$. Therefore,

$$a(1-a)\left[\mathbf{x}_1^T Q\mathbf{x}_1 - 2\mathbf{x}_1^T TQ\mathbf{x}_2 + \mathbf{x}_2^T Q\mathbf{x}_2\right] \geq 0$$

This is trivially true for $a = 0, 1$. Now supposing $a \in (0,1)$,

$$\begin{aligned} \mathbf{x}_1^T Q\mathbf{x}_1 - 2\mathbf{x}_1^T TQ\mathbf{x}_2 + \mathbf{x}_2^T Q\mathbf{x}_2 &\geq 0 \\ (\mathbf{x}_1 - \mathbf{x}_2)^T Q(\mathbf{x}_1 - \mathbf{x}_2) &\geq 0 \end{aligned}$$

As $\mathbf{x}_1$ and $\mathbf{x}_2$ were arbitrary vectors, $\mathbf{x}_1 - \mathbf{x}_2$ can be any vector in $\mathbb{R}^n$, so Q must be positive semidefinite.

Now suppose Q is positive semidefinite. Then for any vector $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{x}^T Q\mathbf{x} \geq 0$. Take $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^n$ such that $\mathbf{x} = \mathbf{x}_1 - \mathbf{x}_2$. Then $(\mathbf{x}_1 - \mathbf{x}_2)^T Q(\mathbf{x}_1 - \mathbf{x}_2) \geq 0$. Take any $a \in [0,1]$. As $a(1-a) \geq 0$, we obtain that

$$a(1-a)\left[\mathbf{x}_1^T Q\mathbf{x}_1 - 2\mathbf{x}_1^T TQ\mathbf{x}_2 + \mathbf{x}_2^T Q\mathbf{x}_2\right] \geq 0$$

To each side of this inequality, add $a^2\mathbf{x}_1^TQ\mathbf{x}_1 + 2a(1-a)\mathbf{x}_1^TQ\mathbf{x}_2 + (1-a)^2\mathbf{x}_2^TQ\mathbf{x}_2 + a\mathbf{c}^T\mathbf{x}_1 + (1-a)\mathbf{c}^T\mathbf{x}_2$. This then gives that the left hand side is $a\mathbf{x}_1^TQ\mathbf{x}_1 + (1-a)\mathbf{x}_2^TQ\mathbf{x}_2 + a\mathbf{c}^T\mathbf{x}_1 + (1-a)\mathbf{c}^T\mathbf{x}_2$, or equivalently $af(\mathbf{x}_1) + (1-a)f(\mathbf{x}_2)$. The right hand side is then $a^2\mathbf{x}_1^TQ\mathbf{x}_1 + 2a(1-a)\mathbf{x}_1^TQ\mathbf{x}_2 + (1-a)^2\mathbf{x}_2^TQ\mathbf{x}_2 + a\mathbf{c}^T\mathbf{x}_1 + (1-a)\mathbf{c}^T\mathbf{x}_2$, or equivalently $f(a\mathbf{x}_1 + (1-a)\mathbf{x}_2)$. So we then have that $af(\mathbf{x}_1) + (1-a)f(\mathbf{x}_2) \geq f(a\mathbf{x}_1 + (1-a)\mathbf{x}_2)$, so f is convex. □

There exist algorithms that solve QP problems in polynomial time when Q is positive definite[26], but if Q is not positive definite then solving this problem is NP-hard[27].

5.3 Quadratically Constrained Quadratic Programming

Quadratically constrained quadratic programming is a further generalization of quadratic programming that allows the constraint functions to also be convex quadratic functions.[3] This is also widely used in statistical applications, as it often occurs that one wants to maximize the expectation of some random variable, while limiting the maximum variance. As an example, suppose you are trying to optimize your investment portfolio. You have n different stocks to choose from, where the current status of the stocks is given by the random variable $\mathbf{p} \in \mathbb{R}^n$, which has known mean $\bar{p}$ and covariance Σ. The random variable p is scaled to its present value, so that p_i represents the price of stock i at some fixed point in the future divided by its price now. Let x_i denote the amount you invest in stock i, B your total investment budget, and σ_{max} the maximum risk you're willing to accept. Finally, suppose you don't want to short any stocks (equivalent to investing a negative amount in a stock). Then you can phrase the problem of maximizing your expected return subject to this maximum risk and budget as follows[3]:

$$\begin{array}{ll} \text{minimize} & -\bar{p}^T\mathbf{x} \\ \text{subject to} & \mathbf{x}^T\Sigma\mathbf{x} - \sigma_{max} \leq 0 \\ & \mathbf{1}^T\mathbf{x} \leq B \\ & \mathbf{x} \geq 0 \end{array}$$

6 More Complex Convex Programs

6.1 Second-Order Cone Programming

6.1.1 Introduction

Second-order cone programming sets the feasible space to be a second-order cone, but in effect it generalizes QCQP programs by allowing the matrices used to have a single negative eigenvalue, rather than forcing all of the eigenvalues to be greater than or equal to zero. This allows for limiting distances by another variable, which is often useful in engineering and space planning applications.

6.1.2 Theory

As stated above, the feasible space is a second-order cone, which is generally denoted as a set in $\mathbb{R}^{n+1}, S = \{(\mathbf{x}, t) | \|\mathbf{x}\|_2 \leq t\}$, or equivalently

$$S = \left\{ \begin{bmatrix} \mathbf{x} \\ t \end{bmatrix} | \begin{bmatrix} \mathbf{x} \\ t \end{bmatrix}^T \begin{bmatrix} I_n & \mathbf{0}_n \\ \mathbf{0}_n^T & -1 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ t \end{bmatrix} \leq 0, t \geq 0 \right\}$$

[3]. It can clearly be seen that the matrix in this constraint is not positive semidefinite, as the eigenvalue for the eigenvector with $\mathbf{x} = \mathbf{0}_n, t = 1$ is -1. However, with the addition of the condition $t \geq 0$, this is a convex set. [3]

6.2 Geometric Programming

6.2.1 Introduction

Definition. *A **monomial** in the context of geometric programming is a function $f : \mathbb{R}^n_{++} \to \mathbb{R}$, defined as $\mathbf{x} \mapsto cx_1^{a_1}x_2^{a_2}...x_n^{a_n}, c > 0$.*

Definition. *A **posynomial** is any sum of monomials as defined above.*

A geometric programming problem takes the form

$$\begin{array}{ll} \text{minimize} & f_0(\mathbf{x}) \\ \text{subject to} & f_i(\mathbf{x}) \leq 1, \quad i = 1, ..., p \\ & g_i(\mathbf{x}) = 1, \quad i = 1, ..., q \end{array}$$

where f_i are posynomials, and g_i are monomials.[28][29]
These problems are actually not generally convex, but they are what is called log-log-convex[30], which is to say that if you substitute every variable for its logarithm, and take the logarithm of the objective function and the constraint functions, then the resulting problem is convex, and corresponds uniquely to the original problem.

These types of problems often arise in integrated circuit design[31], where the volumes and areas of various components need to be optimized, while keeping solid connections with sufficient cross sectional area for current flow, etc. All of those constraints involve muliplying several variables together, and possibly adding those products together, perfectly suited to the above description of monomials and posynomials.

6.2.2 Theory

As mentioned above, these problems can be converted to convex problems by substituting the variables for their logarithms and taking the logarithms of the functions. Consider first a monomial $g(\mathbf{x}) = cx_1^{a_1}x_2^{a_2}...x_n^{a_n} = \exp(\log(c) + a_1\log(x_1) + a_2\log(x_2) + ... + a_n\log(x_n))$. Substituting $y_i = \log(x_i)$ gives $g'(\mathbf{y}) = \exp(\log(c) + a_1y_1 + a_2y_2 + ... + a_ny_n)$, then taking the logarithm of this gives $\tilde{g}(\mathbf{y}) = \mathbf{a}^T\mathbf{y} + \log(c)$. As we had that $g_i(\mathbf{x}) = 1$, we have that $\tilde{g}(\mathbf{y}) = 0$. For the posynomials, this is slightly more complex. Consider $f(\mathbf{x}) = \sum_{i=1}^p c_i \prod_{j=1}^n x_j^{a_{ij}}$. By substituting $y_i = \log(x_i)$, we get $f'(\mathbf{y}) = \sum_{i=1}^p \exp\left(\log(c_i) + \sum_{j=1}^n a_{ij}y_j\right)$. Taking the logarithm, we obtain $\tilde{f}(\mathbf{y}) = \log\left(\sum_{i=1}^p \exp\left(\log(c_i) + \sum_{j=1}^n a_{ij}y_j\right)\right)$. This is the composition of a function called log-sum-exp with an affine function. As noted by Boyd, [3] this is also a convex function if log-sum-exp is convex.

Theorem 6.1. *$f(x) = \log\left(\sum_{i=1} \exp(x_i)\right)$ is a convex function.*

Proof. Take $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$. Then take $u_i = \exp(x_i), v_i = \exp(y_i), a \in [0, 1]$. Then $f(a\mathbf{x} + (1-a)\mathbf{y}) = \log\left(\sum_{i=1}^n u_i^a v_i^{(1-a)}\right)$. Then by Hölder's inequality, as $u_i, v_i > 0$ we have that $\sum_{i=1}^n u_i^a v_i^{(1-a)} \leq (\sum_{i=1}^n u_i^{a\frac{1}{a}})^a (\sum_{i=1}^n v_i^{(1-a)\frac{1}{1-a}})^{(1-a)}$, so then we have that $\log\left(\sum_{i=1}^n u_i^a v_i^{(1-a)}\right) \leq a\log\left(\sum_{i=1}^n u_i\right) + (1-a)\log\left(\sum_{i=1}^n v_i\right)$, or equivalently that $f(a\mathbf{x} + (1-a)\mathbf{y}) \leq af(\mathbf{x}) + (1-a)f(\mathbf{y})$. □

Then as log-sum-exp is convex, and the function $\tilde{f}$ is the composition of log-sum-exp and an affine mapping, $\tilde{f}$ is convex. Once this transformation is performed, the problem can be solved much more easily, then transforming back is straightforward as well, using $x_i = \exp(y_i)$.

6.3 Semidefinite Programming

6.3.1 Introduction

Semidefinite programming is one of the most general kinds of convex optimization, encompassing LP, QP, QCQP, and SOCP. Many other optimization problems can be well approximated by semidefinite programming problems. In fact, semidefinite programming is so general that many commercial solvers that solve all of the above problems actually convert them first to an SDP problem, then solve that SDP problem.

6.3.2 Theory

Definition. *Given a $n \times n$ matrix A, $tr(A)$ is the sum of the diagonal elements of A.*

Definition. *Given two symmetric $n \times n$ matrices $A, B \in \mathbb{S}^n$, define $\langle A, B \rangle_{\mathbb{S}^n} = tr(A^T B)$, where $\mathbb{S}^n$ represents the space of all symmetric $n \times n$ matrices.*

Semidefinite programming problems take the following form

$$\begin{aligned} \text{minimize} \quad & \langle C, X \rangle_{\mathbb{S}^n} \\ \text{subject to} \quad & \langle A_k, X \rangle_{\mathbb{S}^n} \leq b_k, \quad k = 1, ..., m \\ & X \succeq 0 \end{aligned}$$

where $A_k, C, X \in \mathbb{S}^n$, and $X \succeq 0$ means X is positive semidefinite[32]. This actually encapsulates polynomial constraints as well. This comes in with the constraint of positive semidefiniteness for X. This constrains its eigenvalues to all be positive. As the eigenvalues are solutions to the characteristic equation $|X - \lambda I| = 0$, this constrains the n-th order polynomial to only have positive solutions, which is sufficient to have polynomial constraints of order less than n.

6.4 Closing Comments On Convex Optimization

For most of these types of optimization, we did not go into much detail, touching only on their representation and some vague techniques for problem conversions and solutions. That is because for most applications, LP and MILP will be sufficient for the beginning optimizer, and as the advanced solving techniques become significantly more complex in presumed background, they tend to confuse more than enlighten. It is worth noting again that all of these different techniques because you should always use the least complicated model that encapsulates all of the necessary information, as using a more complex model makes it significantly harder to solve.

7 An Example of Non-Convex Optimization

7.1 Introduction

While many problems can be expressed as a convex optimization problem, or can be converted to an equivalent convex optimization problem, there are many other problems that

cannot. This is especially the case in many engineering problems, where even if the problem itself were technically convex, finding a closed form for the feasible region or the objective function is nigh on impossible. One specific example of this is 3D modelling, where mechanical engineers want to have a program design a mechanical part fitting some parameters, and optimize its performance under certain circumstances, as evaluated by a simulation. This is referred to as **black-box optimization**, which is where the objective function is a black box. In all previous algorithms discussed, the algorithm has information about the objective function that it can use in optimizing. In black-box optimization, the algorithm can only give inputs to the objective function, and the function gives back a singular value. This makes the problem significantly harder. In this particular case, evaluations of the objective function are also expensive, both in terms of resources and time, so we want to find our optimum with as few evaluations of the objective function as possible. Finally, we would like to find a global optimum, even though our objective function may not be convex (recall that the reason convex optimization is used is because all local optima are also global optima). Given the information we have from convex optimization, this seems like an impossible task, but algorithms do exist with reasonable performance on problems of exactly this type.

7.2 Mode-Pursuing Sampling

The problem we are attempting to optimize is

$$\begin{aligned} &\text{minimize} && f(\mathbf{x}) \\ &\text{subject to} && \mathbf{x} \in S(f) \subset \mathbb{R}^n \end{aligned}$$

where $S(f)$ is a compact set. The paper we reference uses the simplifying supposition that $S(f) = [a, b]^n$ for some known $-\infty < a < b < \infty$, and we do the same here. The method we will be discussing is the method proposed by Liqun Wang, Songqing Shan, and G Wang in Mode-Pursuing Sampling Method for Global Optimization on Expensive Black-box Functions[33]. The proposed method begins with a method called the mode-pursuing sampling algorithm, which clusters sampled points around the mode of an arbitrary probability density function. It begins with a modification of a sampling algorithm provided by Fu and Wang[34], which is summarized very well in the paper, and so will not be again summarized here. For the purposes of this analysis, it suffices to know that the method generates a discrete probability distribution from a given continuous probability density function, which approximates the initial PDF asymptotically well as the number of sampled points increases.

The mode-pursuing sampling algorithm uses this samplying in a 4 step loop, where P_s is the set of sampled points and the function's value at those points. There are two observations that should be made first though; 1. minimizing f is, as with convex optimization, the same as maximizing $-f$, and 2. Either f is unbounded below on $S(f)$, or there exists some c such that $f(\mathbf{x}) \geq c \forall \mathbf{x} \in S(f)$, so f is assumed to be positive on $S(f)$ without loss of generality, as either a constant may be added to make that true, or there is no solution anyway.

1. Generate a starting set of m initial points $x^{(1)}, x^{(2)}, ..., x^{(m)}$ distributed uniformly on $S(f)$. Add these points and their function values to P_s.

2. Use all of the points and function values in P_s to generate a continuous approximation $\hat{f}$ to f such that $\hat{f}(\mathbf{x}^(i)) = f(\mathbf{x}^{(i)})$ for each $\mathbf{x}^{(i)} \in P_s$.

3. Define $g(\mathbf{x}) = c_0 - \hat{f}(\mathbf{x})$, where c_0 is any constant such that $c_0 \geq \hat{f}(x) \forall \mathbf{x} \in S(f)$. This makes g a probability density function, up to some normalizing constant. The most

probable points are the points close to the maxima of g, which occur at the minima of $\hat{f}$. Use Fu and Wang's sampling method to sample m more points according to g.

4. Add the newly generated points and their function values to P_s. If some stopping criterion is met, then exit with the best answer so far, otherwise go to step 2.

In step 2, the authors use a linear spline function, for reasons detailed in their paper. While these reasons are interesting, they are a bit too in-depth for this analysis, so here we will simply note that they generate an approximation that works well for this algorithm.

This method is quite nice in its relative simplicity, as they note that if no stopping criterion is applied then this algorithm converges to the true optimum of f, and by construction it ensures that it continues to sample the global function even as it approaches the minimum around the current mode, so it ensures that it does not miss the global optimum while exploring near a local optimum.

7.3 Global Optimization Strategy

While we have already expressed that the algorithm so far is reasonable, and converges to the optimum eventually, as described it can be quite slow to converge, especially if the objective function is relatively flat around the global optimum. To address this, the authors introduce two items. One switches the algorithm to a local search mode if a certain criterion is met, and the other is referred to as a speed control factor. We explore the local search mode first.

The authors note that any smooth function can be locally and accurately approximated by a quadratic function (by Taylor's Theorem). They suppose that if the best quadratic approximation around the current mode then it is a reasonable indicator that the current mode is near the global optimum. Using a quadratic approximation of the objective function makes optimizing in a local area trivial. If the neighbourhood of the current best mode does not well fit a quadratic model, then the algorithm emphasizes searching the rest of the space, hoping to find a mode that is better approximated by a quadratic model. If the current mode does fit a quadratic model sufficiently well, then the algorithm performs inexpensive quadratic optimization to find its minimum. If that minimum is sufficiently close, then the algorithm returns that minimum, otherwise it again emphasizes searching the rest of the space.

The way that the algorithm controls whether it explores near the current mode or emphasizes the rest of the search space is using the speed control factor. It modifies the functioning of Fu and Wang's sampling method, to make the current modes more or less probable, and the surrounding area correspondingly less or more probable.

By using these mechanisms, the algorithm converges to a value significantly quicker, but it also has a much higher probability of getting stuck on, or returning the value of, a local minimum rather than the global minimum, as the emphasis on local searching makes it significantly easier to overlook the global minimum.

7.4 Closing comments on non-convex optimization

I, personally, think this is quite an ingenious method of optimization. That said, although the authors report that it is effective even on functions for which the objective function is convex, previously known, or not expensive, the detriments of potentially returning a local minimum instead of the desired global minimum are reason enough to stick with well

established solutions when they work. The method as described in this paper also does not allow for mixed integer optimization, although some of the authors have continued working on variants of this method, including methods accommodating discrete variables[35].

Non-convex optimization is generally about balancing the time spent searching for a solution with minimizing the risk of getting the wrong answer, and I think this method does rather well at that.

8 Conclusion

We have discussed many different types of optimization, primarily convex optimization, and with an emphasis on linear programming. At this point I want to note again that for most real-world problems, a mixed-integer linear program is more than sufficient to approximate the problem, so explaining linear programming and mixed integer linear programming in greater detail is likely to prove more beneficial to the future optimizer than explaining the general optimization subclasses in significantly more detail than presented here. To those whose interest in these classes has been piqued, the textbook Convex Optimization by Stephen Boyd and Lieven Vandenberghe is a fantastic resource for further reading, especially on the theory. However, in this paper I have provided an overview of some of the most common or interesting items in mathematical optimization, balancing mathematical rigour and ease of comprehension.

References

[1] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[2] D. Whitley and J. Watson, "Complexity theory and the no free lunch theorem," in. Jan. 1970, pp. 317–339. DOI: 10.1007/0-387-28356-0_11.

[3] S. Boyd, S. Boyd, L. Vandenberghe, and C. U. Press, *Convex Optimization*, ser. Berichte über verteilte messysteme. Cambridge University Press, 2004, ISBN: 9780521833783. [Online]. Available: https://books.google.ca/books?id=mYm0bLd3fcoC.

[4] G. B. Dantzig, "Origins of the simplex method," in *A History of Scientific Computing*. New York, NY, USA: Association for Computing Machinery, 1990, pp. 141–151, ISBN: 0201508141. [Online]. Available: https://doi.org/10.1145/87252.88081.

[5] K. Murty, *Linear programming*. New York: Wiley, 1983, ISBN: 978-0-471-09725-9.

[6] G. Dantzig and M. Thapa, *Linear Programming 1: Introduction*, ser. Springer Series in Operations Research and Financial Engineering. Springer New York, 1997, ISBN: 9780387948331. [Online]. Available: https://books.google.ca/books?id=I-aluupuDEMC.

[7] R. G. Bland, "New finite pivoting rules for the simplex method," *Mathematics of Operations Research*, vol. 2, no. 2, pp. 103–107, 1977, ISSN: 0364765X, 15265471. [Online]. Available: http://www.jstor.org/stable/3689647.

[8] A. Schrijver, *Theory of linear and integer programming*. Chichester New York: Wiley, 1998, ISBN: 0-471-98232-6.

[9] J. Matoušek, *Understanding and using linear programming*. Berlin: Springer, 2007, ISBN: 3-540-30697-8.

[10] P. M. J. Harris, "Pivot selection methods of the devex LP code," *Mathematical Programming*, vol. 5, no. 1, pp. 1–28, Dec. 1973. DOI: 10.1007/bf01580108. [Online]. Available: https://doi.org/10.1007/bf01580108.

[11] K. Borgwardt, *The simplex method, a probabilistic analysis*. Berlin New York: Springer-Verlag, 1987, ISBN: 3-540-17096-0.

[12] V. Klee and G. J. Minty, "How good is the simplex algorithm," 1970.

[13] D. A. Spielman and S. Teng, "Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time," *CoRR*, vol. cs.DS/0111050, 2001. [Online]. Available: https://arxiv.org/abs/cs/0111050.

[14] S. S. Morgan, "A comparison of simplex method algorithms," Archived from the original on 2011-08-07, MSc thesis, University of Florida, 1997. [Online]. Available: https://web.archive.org/web/20110807134509/http://www.cise.ufl.edu/research/sparse/Morgan/index.htm (visited on 2011).

[15] (2020). Blue yonder gmbh pricing decisions - gurobi, [Online]. Available: https://www.gurobi.com/case_study/blue-yonder-gmbh/.

[16] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department*, R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, Eds. Boston, MA: Springer US, 1972, pp. 85–103, ISBN: 978-1-4684-2001-2. DOI: 10.1007/978-1-4684-2001-2_9. [Online]. Available: https://doi.org/10.1007/978-1-4684-2001-2_9.

[17] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, ser. STOC '71, Shaker Heights, Ohio, USA: Association for Computing Machinery, 1971, pp. 151–158, ISBN: 9781450374644. DOI: 10.1145/800157.805047. [Online]. Available: https://doi.org/10.1145/800157.805047.

[18] J. Mitchell, "Branch-and-cut algorithms for combinatorial optimization problems1," *Handbook of Applied Optimization*, Jan. 2002.

[19] A. H. Land and A. G. Doig, "An automatic method of solving discrete programming problems," *Econometrica*, vol. 28, no. 3, pp. 497–520, 1960, ISSN: 00129682, 14680262. [Online]. Available: http://www.jstor.org/stable/1910129.

[20] H. Marchand, A. Martin, R. Weismantel, and L. Wolsey, "Cutting planes in integer and mixed integer programming," *Discrete Applied Mathematics*, vol. 123, no. 1, pp. 397–446, 2002, ISSN: 0166-218X. DOI: https://doi.org/10.1016/S0166-218X(01)00348-1. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166218X01003481.

[21] G. Pataki and M. Tural, *Basis reduction, and the complexity of branch-and-bound*, 2009. arXiv: 0907.2639 [math.OC].

[22] (2020). German first division basketball league scheduling - gurobi, [Online]. Available: https://www.gurobi.com/case_study/bbl-english/.

[23] (2020). National football league scheduling, [Online]. Available: https://www.gurobi.com/case_study/nfl-english/.

[24] (2020). Hewlett-packard project portfolio optimization, [Online]. Available: https://www.gurobi.com/case_study/ppo-hp-english/.

[25] J. Nocedal, *Numerical optimization*. New York: Springer, 2006, ISBN: 978-0-387-30303-1.

[26] L. M.K. Kozlov S.P. Tarasov, "[the polynomial solvability of convex quadratic programming]," English, trans. Russian by J. Berry, *USSR Computational Mathematics and Mathematical Physics*, vol. 20, pp. 223–228, 5 Feb. 5, 1979. DOI: 10.1016/0041-5553(80)90098-1.

[27] S. Sahni, "Computationally related problems," *SIAM Journal on Computing*, vol. 3, no. 4, pp. 262–279, 1974. DOI: 10.1137/0203021. eprint: https://doi.org/10.1137/0203021. [Online]. Available: https://doi.org/10.1137/0203021.

[28] R. Duffin, *Geometric programming : theory and application*. New York: John Wiley, 1967, ISBN: 0-471-22370-0.

[29] S. Boyd, S.-J. Kim, L. Vandenberghe, and A. Hassibi, "A tutorial on geometric programming," *Optimization and Engineering*, vol. 8, no. 1, pp. 67–127, Apr. 2007. DOI: 10.1007/s11081-007-9001-7. [Online]. Available: https://doi.org/10.1007/s11081-007-9001-7.

[30] A. Agrawal, S. Diamond, and S. Boyd, *Disciplined geometric programming*, 2018. arXiv: 1812.04074 [math.OC].

[31] M. d. Hershenson, S. P. Boyd, and T. H. Lee, "Optimal design of a cmos op-amp via geometric programming," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 1, pp. 1–21, 2001.

[32] R. M. Freund, "Introduction to semidefinite programming (sdp)," 2004.

[33] L. Wang, S. Shan, and G. G. Wang, "Mode-pursuing sampling method for global optimization on expensive black-box functions," *Engineering Optimization*, vol. 36, no. 4, pp. 419–438, 2004. DOI: 10.1080/03052150410001686486. eprint: https://doi.org/10.1080/03052150410001686486. [Online]. Available: https://doi.org/10.1080/03052150410001686486.

[34] J. Fu and L. Wang, "A random-discretization based monte carlo sampling method and its applications," *Methodology And Computing In Applied Probability*, vol. 4, pp. 5–25, Jan. 2002. DOI: 10.1023/A:1015790929604.

[35] B. Sharif, G. G. Wang, and T. Y. ElMekkawy, "Mode Pursuing Sampling Method for Discrete Variable Optimization on Expensive Black-Box Functions," *Journal of Mechanical Design*, vol. 130, no. 2, Jan. 2008, 021402, ISSN: 1050-0472. DOI: 10.1115/1.2803251. eprint: https://asmedigitalcollection.asme.org/mechanicaldesign/article-pdf/130/2/021402/5617466/021402_1.pdf. [Online]. Available: https://doi.org/10.1115/1.2803251.