1 Introduction

This course is an introduction to the concept of discrete optimization. I should emphasize that what we cover in this course is a very brief overview. We only study popular problems and techniques that you are more likely to encounter in future. In fact, a comprehensive course on this topics requires a much greater deal of time and effort than what we spend in this course. The reason is that mathematical optimization, especially in the discrete case, lacks a generic approach. This means that there is no small set of principles available, such that one can easily[1] solve every type of problems by referring to them. On the contrary, there has been a group of different ideas in the course of history, which has been one by one applied to a large group of problems, and as a result, some ideas are found to be successful for some problems and some not. A comprehensive course, should go through all of these ideas and their suitable applications. Besides, there are a number of challenging problems, for which no suitable idea has been found. It is not even known whether these problems may be "easily" solved or not, though there is a strong disbelief about it! These topics are what you may find in a typical discrete optimization course book. However, as I have already mentioned, we are not going to deal with all of them and simply stick to the ones that we afford.

In spite of the above mentioned difficulties, mathematical optimization is one of the most useful disciplines in our modern world. Industrial planning, organizational management, traffic management, and many other tasks are hardly conceivable without the advanced optimization techniques. We also have a very good understanding about the approximation techniques and when they are not approximate anymore! But how did it all happen? By the rapid growth of industrialization in the 19th and 20th century, many organizations eventually turned their attention to improving their methods in order to increase their chance of success. Driven by the industrial needs, mathematicians started to formulate problems, in order to find the maximum amount of profit or the minimum amount of undesired cost, etc. They called these problems mathematical **optimization** or **programming**. Next, they devised algorithms to find the maximal or minimal values in their problems, which has been since then substantally improved. Formulating a real-world problem as a mathematical programming is often referred to as **modeling**, while the algorithms are sometimes called **optimization methods** and sometimes just **algorithms**. It took a while until the importance of mathematical programming was understood by the society. After a long course of research, the computerized optimization methods became so efficient in some cases that they could easily lead to significant improvements in different industries, without a significant knowledge about the nature of those indutries. At this time, it was not possible anymore to neglect the role of optimization in practice. Nowadays, optimization is not used only for planning and management. Machine learning, for example, is one of those areas, where optimization plays an important role. I cannot further explain this topic as it requires a different background, which I do not assume you have. However, I will point out to some of these applications later in the course.

Now, let us talk in more details about a general optimization problem. Indeed, we need a good notational convention to express our problems. Nowadays, there is a standard way to denote a mathematical optimization, which often looks like the following:

$$\begin{aligned} &\min_{\mathbf{x}\in\Psi} f(\mathbf{x}) \\ &\text{subject to} \\ &P[\mathbf{x}] \end{aligned} \tag{1}$$

In (1), $\mathbf{x}$ denotes the parameter(s) that we can adjust and is called the optimization **variable**. The set Ψ is the set of all possible values that $\mathbf{x}$ admits and is called the **variable space**. Further, $f(\mathbf{x})$ denotes a function that assigns to each value in the variable space a real number, known as the cost. The function $f(\mathbf{x})$ is itself referred to as the **cost function**. Finally, $P[\mathbf{x}]$ represents a set of **constraints**. The constraints restrict the variables to satisfy certain specifications. They

[1] This refers to the number of calculations an algorithm needs to solve a problem. We will shortly talk about it.

can be included simply by a number of statements like "$P[\mathbf{x}]$: $\mathbf{x}$ contains at least one zero.", but it is more popular to use numerical equality and inequalities to represent them. Here is how it generally looks like:

$$\begin{gathered}\min_{\mathbf{x}\in\Psi} f(\mathbf{x}) \\ \text{subject to} \\ \begin{array}{ll} g_1(\mathbf{x}) \leq 0 & h_1(\mathbf{x}) = 0 \\ g_2(\mathbf{x}) \leq 0 & h_2(\mathbf{x}) = 0 \\ \vdots \quad , & \vdots \\ g_m(\mathbf{x}) \leq 0 & h_p(\mathbf{x}) = 0 \end{array}\end{gathered} \tag{2}$$

In (2), each term $g_i(\mathbf{x})$ or $h_j(\mathbf{x})$ for $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, p$ is a real function. So, a valid choice of $\mathbf{x}$ should satisfy all the **equality constraints** $h_j(\mathbf{x}) = 0$ as well as the inequality constraints $g_i(\mathbf{x}) \leq 0$. An optimization, which does not have any constraint is called **unconstrained**. Sounds a bit abstract!? Let us make a concrete example:

Example 1. A pharmaceutical company has discovered and is the only producer of a new type of lung cancer drug. Hence, the company can control the market price of this drug by adjusting its daily production. The amount of production is measured in the unit of boxes per day. The company delivers its entire daily production to the market. The goal is to maximize the profit of the company.

Suppose that the company produces and delivers to the market x boxes per day. According to the demand curve, the market price of the drug p SEK/box is given by

$$p = 900 - 0.4x \tag{3}$$

On the other hand, the supply curve shows that the cost c SEK/box of producing x boxes is given by

$$c = 100 + 0.6x \tag{4}$$

For technical reasons, the company may not produce more than 500 boxes per day.

Now, the total profit of the company per day is given by $f(x) = x(p - c) = x(800 - x)$ and we can write down the following optimization:

$$\begin{gathered}\max_{x\in\mathbb{Z}} x(800 - x) \\ \text{subject to} \\ 0 \leq x \leq 500\end{gathered} \tag{5}$$

You may notice that the optimization in (5) does not totally agree with the template we introduced in (2):

1. First, notice that I used max, while the template in (2) employs min. This is indeed a valid and very popular notation, but many books avoid one of the min or max when they explain the theory. The reason is that one of them is enough to explain even the other one; You can easily turn any maximization problem into an equivalent minimization by changing the sign of the cost function. Take a look at Figure 1. As seen, the maximum point of $f(\mathbf{x})$ is always the same as the minimum point of $-f(\mathbf{x})$. So, if one changes the sign of the cost function the optimization remains unchanged, except for the sign of the optimal value. However, it is occasionally more convenient to use the max notation. In this case, it is not appropriate anymore to call $f(\mathbf{x})$ cost function. Instead, it is often called **score** or **utility** function.

2. We also used $0 \leq x \leq 500$ as a short hand notation for two constraints, namely $0 \leq x$ and $x \leq 500$. However, these two are not in the form of "$g_i(\mathbf{x}) \leq 0$", but can be easily modified to be. We can multiply $0 \leq x$ by minus one to obtain $-x \leq 0$ and subtract 500 from $x \leq 500$ to obtain $x - 500 \leq 0$. So, we get $g_1(x) = -x$ and $g_2(x) = x - 500$.

Hence, we can write (5) as

$$\begin{aligned} &\min_{x\in\mathbb{Z}} -x(800-x) \\ &\text{subject to} \\ &-x \le 0 \qquad (6) \\ &x-500 \le 0, \qquad (7) \end{aligned}$$

which agrees with (2).

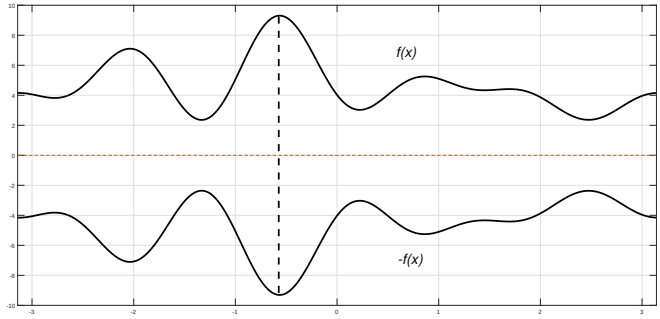

Figure 1: Maximizing the function $f(\mathbf{x})$

Of course, the main task after modeling is to solve the optimization. The solution consists of two parts: A value (or a point) $\mathbf{x}^*$ in the variable space, which satisfies the constraints and minimizes the cost function and the minimum value $f^* = f(\mathbf{x}^*)$. Any point in the variable space, which satisfies the constraints is called a **feasible solution** and $\mathbf{x}^*$ is referred to as an **optimal feasible solution** or simply an **optimal solution**. The set of all feasible solutions is called the **feasible region**. I often use Ω to refer to the feasible region. Moreover, $f^* = f(\mathbf{x}^*)$ is called the **optimal value**. There might be more than one optimal solution, but clearly there is a unique optimal value. As a convention, $f^* = \min f(\mathbf{x})$ means that f^* is the optimal value. To refer to the optimal solution, we use $\mathbf{x}^* = \arg\min f(\mathbf{x})$. Technically, $\arg\min f(\mathbf{x})$ is the set of all optimal solutions. So, it is technically correct to write $\mathbf{x}^* \in \arg\min f(\mathbf{x})$, but the former notation is also popular. In the form of (2), an individual inequality constraint $g_i(\mathbf{x}) \le 0$ is said to be **active** at the optimal point if $g_i(\mathbf{x}^*) = 0$. Otherwise, it is called **inactive**.

An optimization might not have any solution. This is because either there is no existing feasible solution (the feasible region is empty), in which case the optimization is called **infeasible**, or no minimal value exists. The latter case often happens when the cost function can be decreased as much as we desire. It can get less than -1, $-1,000,000$, -10^{100} or even smaller! In this case, the optimization is said to be **unbounded**. There are more sophisticated situations, where the minimal value does not exist. Take the following example.

Example 2. Consider the optimization

$$\begin{aligned} &\min_{x\in\mathbb{R}} \tfrac{1}{x} \\ &\text{subject to} \\ &x \ge 1 \qquad (8) \end{aligned}$$

The cost is positive and can get as small as desired by increasing x to infinity, but it never reaches zero. So, there is actually no minimum point for this optimization. Technically, the value 0 is not the optimal value, since it is not attainable. It is instead called the **infimum**, but this is too theoretical to be considered here.

Now, let us take a brief look at the optimization algorithms. Generally speaking the optimization problems are divided into three different categories. Remember that the optimization

variable may consists of different entries, i.e. it can be written as $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, where n is the number of entries. An optimization is called **discrete** if all the entries of $\mathbf{x}$ admit discrete values. On the contrary, an optimization is called **continuous** if all the entries are real-valued and the variable space is $\mathbb{R}^n$. A discrete optimization is called **integer** if the entries of $\mathbf{x}$ are integer and the variable space is $\mathbb{Z}^n$. It is instead called **binary** if they only admit two values, for example when the variable space is $\{0, 1\}^n$. Discrete optimizations require different methods to the continuous ones, although there are certain links between them. A number of issues, related to the two classes of continuous and discrete optimizations, is summarized in Figure 2. If some entries of $\mathbf{x}$ are discrete and some other continuous, the optimization is referred to as **mixed**. One of the main issues with optimization algorithms is their speed. Speed is measured in terms

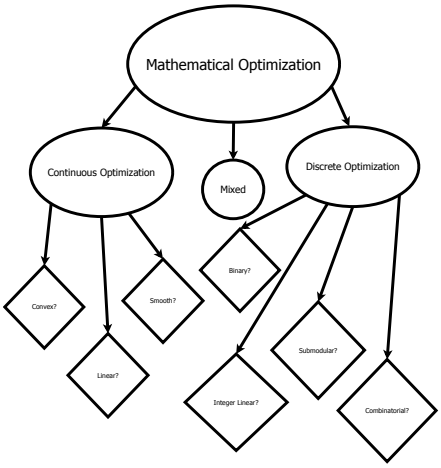

Figure 2: The general classification of different optimization methods.

of the number of operations that an algorithm performs before termination. This is often called the **computational complexity** of an algorithm. Algorithms are scarcely designed only for a very single problem, but they often cover a family of problems with an arbitrary size. Denote the size of a problem by n. Then, the computational complexity is a function $C(n)$ of the size n. In many cases, it is only important to know how fast $C(n)$ grows with n. Let us illustrate this with a simple example.

Example 3. A social network is planning to broadcast an advertisement. The network has n users and has already collected some information regarding the behavioral pattern of its users. This information is turned into a set of scores s_i for $i = 1, 2, \ldots, n$. The more the score s_i is, the more likely is that the i^{th} user will buy the product. For technical reasons, the network can only broadcast the advertisement over 1000 users. The problem is to select them to maximize the chance of selling the product (the total score).

It should be clear that the answer to the problem is to select 1000 users with the largest scores. To do this, we first select the largest element, which needs n comparisons, then the second largest, which needs $n - 1$ comparisons and so on. The total number of operations (comparisons) for this

algorithm is

$$C_1(n) = n + (n-1) + (n-2) + \ldots + (n-999) = \frac{1000 \times (2n - 999)}{2} = 1000n - 499500 \quad (9)$$

So, the computational complexity of this algorithm grows **linearly** with the number of users.

Now, let us try to formulate the problem as a standard optimization problem in (2). Let us introduce a set of indicators $x_i \in \{0, 1\}$ for $i = 1, 2, \ldots, n$, such that $x_i = 1$ means that the i^{th} user is selected. We can write our problem as

$$\begin{aligned} &\max_{\mathbf{x}=(x_1,x_2,\ldots,x_n)\in\{0,1\}^n} x_1 s_1 + x_2 s_2 + \ldots + x_n s_n \\ &\text{subject to} \\ &x_1 + x_2 + \ldots + x_n \leq 1000 \end{aligned} \quad (10)$$

Let us forget that the solution of this problem is the "1000 largest scores" and solve the problem by trying every possible vector $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \{0, 1\}^n$. This is called **exhaustive search** or **brute force**. We should first examine that this vector satisfy the constraint and then check if it produces a larger cost than the previously examined ones. Since, there are 2^n different vectors and each examination needs a number of operations (can you say how many ?), the complexity of this algorithm is higher than 2^n, i.e. it is **super-exponential**.

To get a sense of how the two algorithms are different in speed, consider the "facebook" case with 1.5 billion users. The complexity of the first algorithm is less than $1000 * n$, which is in the order of 10^{12} comparisons. With an ordinary desktop computer, this should take few hours to perform. Now take the second method. Its complexity is more than 2^{10^9} or around 10^{10^8}. To see how large this number is, notice that if one billion desktop computers work for one thousand years, they can only examine 10^{30} vectors! It is seen that exhaustive search cannot be a solution for large problems and "smarter" algorithms should be employed.

Another important aspect of the optimization algorithms is their precision. In general, it might be difficult to find the optimal solution and the optimal value of an optimization problem with a reasonable complexity, but many algorithms may offer an approximate solution in a more affordable time. Suppose that a minimization problem has an optimal solution $\mathbf{x}^*$ with the optimal value $f^* = f(\mathbf{x}^*) \geq 0$. It is said that an algorithm provides an $\alpha-$**approximate** solution to this problem, where $\alpha > 1$, if its output is a feasible solution $\bar{\mathbf{x}}$ such that $f(\mathbf{x}^*) \leq f(\bar{\mathbf{x}}) \leq \alpha f(\mathbf{x}^*)$. The solution $\mathbf{x}^*$ is called the **exact solution**. Later in the course, we consider algorithms with both exact and approximate solutions.