

# **Mathematical Programming and Operations Research**

## Modeling, Algorithms, and Complexity

### Examples in Python and Julia

(Work in progress)

Edited by: Robert Hildebrand

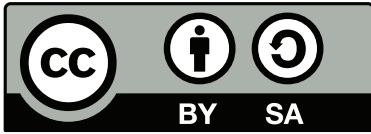
Contributors: Robert Hildebrand, Laurent Poirrier

Version v0.0.7-g08587b6-2020-1-18





Copyright 2019 by the contributors listed on the title page.



This work is licensed under a [Creative Commons “Attribution-ShareAlike 4.0 International”](https://creativecommons.org/licenses/by-sa/4.0/) license.

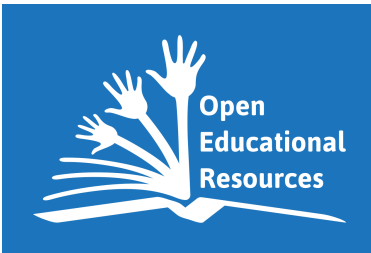
<https://creativecommons.org/>



This license allows everyone to remix, tweak, and build upon this work, even for commercial purposes, ... as long as they license their new creations under identical terms ...



... and if they credit the copyright holders.



This work aligns with the mission of UNESCO Open Educational Resources.

<https://en.unesco.org/themes/building-knowledge-societies/oer>



The source code of this book is available.

<https://github.com/open-optimization/>

---

<sup>1</sup>[https://en.wikipedia.org/wiki/Open\\_educational\\_resources#/media/File:Global\\_Open\\_Educational\\_Resources\\_Logo.svg](https://en.wikipedia.org/wiki/Open_educational_resources#/media/File:Global_Open_Educational_Resources_Logo.svg)

---

# Preface

## Open Optimization

Welcome to the Open Optimization - an ecosystem for open-source materials for teaching optimization and operations research. This ecosystem is being formed to host open-source lecture notes, lecture slides, examples, code, figures, and textbooks on material and courses related to optimization. All material will be licensed under Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0) that permits free reuse and alteration of the material provided the proper attribution is given. All material posted will be not just open-source, but open-source code as well - including LaTeX, tikz, and other means of generating content. This allows those interested in reusing material an easy way to change and adapt the material as needed.

Your contributions to this endeavor are greatly valued and appreciated. You are being contacted directly due to the excellent material that you have on your website. We hope you will help with this project and also use it as a resource for future courses, lectures, and presentations.

Content posted here may be adapted into freely available open-source published textbooks.

Goals:

1. Create freely available content to make easier teaching, designing courses, writing presentations, and finding reusable content.
2. Create free textbooks for courses on optimization and operations research that are:
  - Modern (up to date with current techniques and approaches)
  - Flexible (easy to adapt to the user's choice of presentation of material)
  - Connected to code examples (get students up and running faster)
3. Community collaboration on content authoring and revisions
4. Collect figures and images with source code for quality reproducibility
5. Host instructive code for optimization

## Notation

- $\mathbb{1}$  - a vector of all ones (the size of the vector depends on context)
- $\forall$  - for all
- $\exists$  - there exists
- $\in$  - in
- $\therefore$  - therefore
- $\Rightarrow$  - implies
- s.t. - such that (or sometimes "subject to".... from context?)
- $\{0,1\}$  - the set of numbers 0 and 1
- $\mathbb{Z}$  - the set of integers (e.g.  $1, 2, 3, -1, -2, -3, \dots$ )
- $\mathbb{Q}$  - the set of rational numbers (numbers that can be written as  $p/q$  for  $p, q \in \mathbb{Z}$  (e.g.  $1, 1/6, 27/2$ ))
- $\mathbb{R}$  - the set of all real numbers (e.g.  $1, 1.5, \pi, e, -11/5$ )
- $\setminus$  - setminus, (e.g.  $\{0, 1, 2, 3\} \setminus \{0, 3\} = \{1, 2\}$ )
- $\cup$  - union (e.g.  $\{1, 2\} \cup \{3, 5\} = \{1, 2, 3, 5\}$ )
- $\cap$  - intersection (e.g.  $\{1, 2, 3, 4\} \cap \{3, 4, 5, 6\} = \{3, 4\}$ )
- $\{0, 1\}^4$  - the set of 4 dimensional vectors taking values 0 or 1, (e.g.  $[0, 0, 1, 0]$  or  $[1, 1, 1, 1]$ )
- $\mathbb{Z}^4$  - the set of 4 dimensional vectors taking integer values (e.g.,  $[1, -5, 17, 3]$  or  $[6, 2, -3, -11]$ )
- $\mathbb{Q}^4$  - the set of 4 dimensional vectors taking rational values (e.g.  $[1.5, 3.4, -2.4, 2]$ )
- $\mathbb{R}^4$  - the set of 4 dimensional vectors taking real values (e.g.  $[3, \pi, -e, \sqrt{2}]$ )
- $\sum_{i=1}^4 i = 1 + 2 + 3 + 4$
- $\sum_{i=1}^4 i^2 = 1^2 + 2^2 + 3^2 + 4^2$
- $\sum_{i=1}^4 x_i = x_1 + x_2 + x_3 + x_4$
- $\square$  - this is a typical Q.E.D. symbol that you put at the end of a proof meaning "I proved it."
- For  $x, y \in \mathbb{R}^3$ , the following are equivalent (note, in other contexts, these notations can mean different things)
  - $x^\top y$      *matrix multiplication*
  - $x \cdot y$      *dot product*

–  $\langle x, y \rangle$     *inner product*

and evaluate to  $\sum_{i=1}^3 x_i y_i = x_1 y_1 + x_2 y_2 + x_3 y_3$ .

A sample sentence:

$$\forall x \in \mathbb{Q}^n \exists y \in \mathbb{Z}^n \setminus \{0\}^n \text{ s.t. } x^\top y \in \{0, 1\}$$

"For all non-zero rational vectors  $x$  in  $n$ -dimensions, there exists a non-zero  $n$ -dimensional integer vector  $y$  such that the dot product of  $x$  with  $y$  evaluates to either 0 or 1."



# Contents





## **Part I**

# **Introduction to Optimization**



# Chapter 1

## Mathematical Programming

We will state main general problem classes to be associated with in these notes. These are Linear Programming (LP), Mixed-Integer Linear Programming (MILP), Non-Linear Programming (NLP), and Mixed-Integer Non-Linear Programming (MINLP).

Along with each problem class, we will associate a complexity class for the general version of the problem. See [chapter 2](#) for a discussion of complexity classes. Although we will often state that input data for a problem comes from  $\mathbb{R}$ , when we discuss complexity of such a problem, we actually mean that the data is rational, i.e., from  $\mathbb{Q}$ , and is given in binary encoding.

### 1.1 Linear Programming (LP)

Some linear programming background, theory, and examples will be provided in ??.

#### **Linear Programming (LP):**

*Polynomial time (P)*

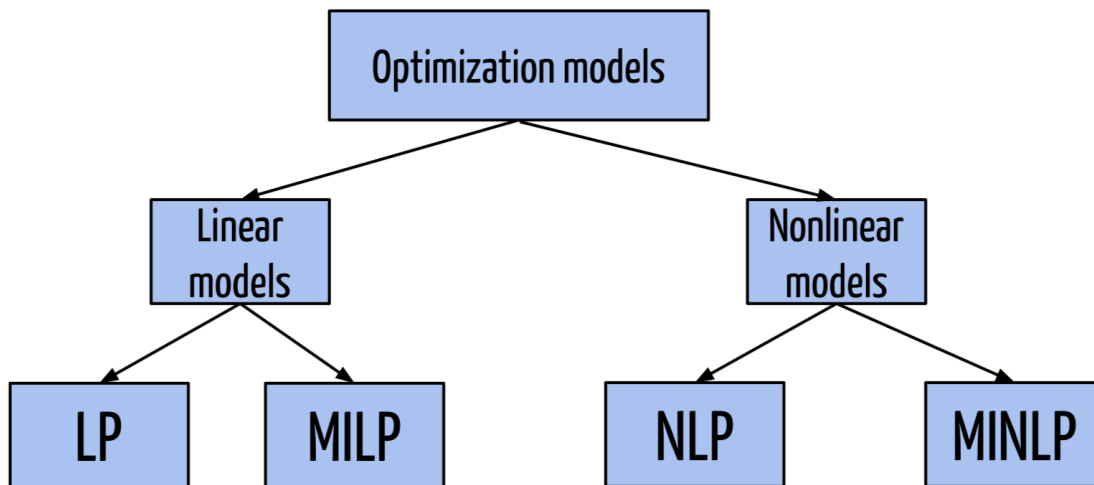
Given a matrix  $A \in \mathbb{R}^{m \times n}$ , vector  $b \in \mathbb{R}^m$  and vector  $c \in \mathbb{R}^n$ , the *linear programming* problem is

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \\ & x \geq 0 \end{aligned} \tag{1.1.1}$$

Linear programming can come in several forms, whether we are maximizing or minimizing, or if the constraints are  $\leq$ ,  $=$  or  $\geq$ . One form commonly used is *Standard Form* given as

---

<sup>1</sup>Tree of optimization problems. from . Diego Moran [CCO.](#), 2017.



© Diego Moran CC0.<sup>1</sup>

**Figure 1.1:** Tree of optimization problems.

### Linear Programming (LP) Standard Form:

*Polynomial time (P)*

Given a matrix  $A \in \mathbb{R}^{m \times n}$ , vector  $b \in \mathbb{R}^m$  and vector  $c \in \mathbb{R}^n$ , the *linear programming* problem in *standard form* is

$$\begin{aligned}
 \max \quad & c^\top x \\
 \text{s.t.} \quad & Ax = b \\
 & x \geq 0
 \end{aligned} \tag{1.1.2}$$

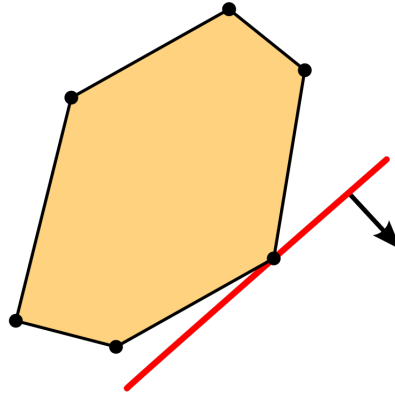
Figure 1.2

**Exercise 1.** Start with a problem in form given as (1.1.1) and convert it to standard form (1.1.2) by adding at most  $m$  many new variables and by enlarging the constraint matrix  $A$  by at most  $m$  new columns.

## 1.2 Mixed-Integer Linear Programming (MILP)

Mixed-integer linear programming will be the focus of Sections 3.4, ??, and ??. Recall that the notation  $\mathbb{Z}$  means the set of integers and the set  $\mathbb{R}$  means the set of real numbers.

<sup>2</sup>A pictorial representation of a simple linear program with two variables and six inequalities. The set of feasible solutions is depicted in yellow and forms a polygon, a 2-dimensional polytope. The linear cost function is represented by the red line and the arrow: The red line is a level set of the cost function, and the arrow indicates the direction in which we are optimizing. from [https://en.wikipedia.org/wiki/Linear\\_programming#/media/File:Linear\\_optimization\\_in\\_a\\_2-dimensional\\_polytope.svg](https://en.wikipedia.org/wiki/Linear_programming#/media/File:Linear_optimization_in_a_2-dimensional_polytope.svg). Ylooh CC0., 2012.

© Ylloh CC0.<sup>2</sup>

**Figure 1.2:** Linear programming constraints and objective.

The first problem of interest here is a *binary integer program* (BIP) where all  $n$  variables are binary (either 0 or 1).

**Binary Integer programming (BIP):**

*NP-Complete*

Given a matrix  $A \in \mathbb{R}^{m \times n}$ , vector  $b \in \mathbb{R}^m$  and vector  $c \in \mathbb{R}^n$ , the *binary integer programming* problem is

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \\ & x \in \{0, 1\}^n \end{aligned} \tag{1.2.1}$$

A slightly more general class is the class of *Integer Linear Programs* (ILP). Often this is referred to as *Integer Program* (IP), although this term could leave open the possibility of non-linear parts.

Figure 1.3

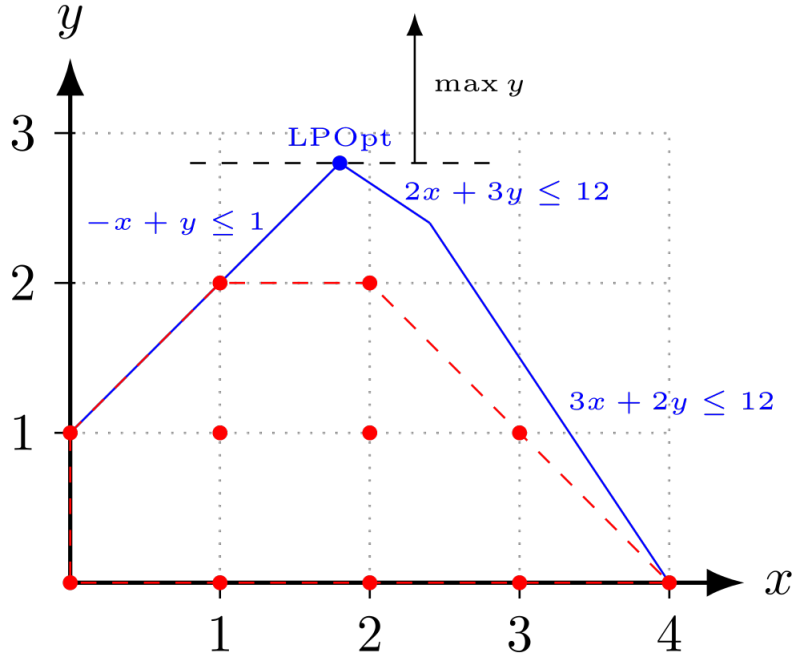
**Integer Linear Programming (ILP):**

*NP-Complete*

Given a matrix  $A \in \mathbb{R}^{m \times n}$ , vector  $b \in \mathbb{R}^m$  and vector  $c \in \mathbb{R}^n$ , the *integer linear programming* problem is

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \\ & x \in \mathbb{Z}^n \end{aligned} \tag{1.2.2}$$

<sup>3</sup>IP polytope with LP relaxation, from [https://en.wikipedia.org/wiki/Integer\\_programming#/media/File:IP\\_polytope\\_with\\_LP\\_relaxation.svg](https://en.wikipedia.org/wiki/Integer_programming#/media/File:IP_polytope_with_LP_relaxation.svg). Fanosta CC BY-SA 4.0., 2019.



© Fanosta CC BY-SA 4.0.<sup>3</sup>

**Figure 1.3:** Comparing the LP relaxation to the IP solutions.

An even more general class is *Mixed-Integer Linear Programming (MILP)*. This is where we have  $n$  integer variables  $x_1, \dots, x_n \in \mathbb{Z}$  and  $d$  continuous variables  $x_{n+1}, \dots, x_{n+d} \in \mathbb{R}$ . Succinctly, we can write this as  $x \in \mathbb{Z}^n \times \mathbb{R}^d$ , where  $\times$  stands for the *cross-product* between two spaces.

Below, the matrix  $A$  now has  $n + d$  columns, that is,  $A \in \mathbb{R}^{m \times (n+d)}$ . Also note that we have not explicitly enforced non-negativity on the variables. If there are non-negativity restrictions, this can be assumed to be a part of the inequality description  $Ax \leq b$ .

#### **Mixed-Integer Linear Programming (MILP):**

*NP-Complete*

Given a matrix  $A \in \mathbb{R}^{m \times (n+d)}$ , vector  $b \in \mathbb{R}^m$  and vector  $c \in \mathbb{R}^{n+d}$ , the *mixed-integer linear programming* problem is

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \\ & x \in \mathbb{Z}^n \times \mathbb{R}^d \end{aligned} \tag{1.2.3}$$

## 1.3 Non-Linear Programming (NLP)

### NLP:

#### *NP-Hard*

Given a function  $f(x): \mathbb{R}^d \rightarrow \mathbb{R}$  and other functions  $f_i(x): \mathbb{R}^d \rightarrow \mathbb{R}$  for  $i = 1, \dots, m$ , the *nonlinear programming* problem is

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & f_i(x) \leq 0 \quad \text{for } i = 1, \dots, m \\ & x \in \mathbb{R}^d \end{aligned} \tag{1.3.1}$$

Nonlinear programming can be separated into convex programming and non-convex programming. These two are very different beasts and it is important to distinguish between the two.

### 1.3.1 Convex Programming

Here the functions are all **convex**!

### Convex Programming:

#### *Polynomial time (P)* (typically)

Given a convex function  $f(x): \mathbb{R}^d \rightarrow \mathbb{R}$  and convex functions  $f_i(x): \mathbb{R}^d \rightarrow \mathbb{R}$  for  $i = 1, \dots, m$ , the *convex programming* problem is

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & f_i(x) \leq 0 \quad \text{for } i = 1, \dots, m \\ & x \in \mathbb{R}^d \end{aligned} \tag{1.3.2}$$

**Example 1:** Convex programming is a generalization of linear programming. This can be seen by letting  $f(x) = c^\top x$  and  $f_i(x) = A_i x - b_i$ .

### 1.3.2 Non-Convex Non-linear Programming

When the function  $f$  or functions  $f_i$  are non-convex, this becomes a non-convex nonlinear programming problem. There are a few complexity issues with this.

**IP as NLP** As seen above, quadratic constraints can be used to create a feasible region with discrete solutions. For example

$$x(1 - x) = 0$$

has exactly two solutions:  $x = 0, x = 1$ . Thus, quadratic constraints can be used to model binary constraints.

**Binary Integer programming (BIP) as a NLP:**

*NP-Hard*

Given a matrix  $A \in \mathbb{R}^{m \times n}$ , vector  $b \in \mathbb{R}^m$  and vector  $c \in \mathbb{R}^n$ , the *binary integer programming* problem is

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \\ & \cancel{x \in \{0, 1\}^n} \\ & x_i(1 - x_i) = 0 \quad \text{for } i = 1, \dots, n \end{aligned} \tag{1.3.3}$$

## 1.4 Mixed-Integer Non-Linear Programming (MINLP)

### 1.4.1 Convex Mixed-Integer Non-Linear Programming

### 1.4.2 Non-Convex Mixed-Integer Non-Linear Programming

## 1.5 Models

- [INFORMS - Impact](#)
- [INFORMS - Success Story - Bus Routing](#)
- [AIMMS - Employee Training](#)
- [AIMMS - Media Selection](#)
- [AIMMS - Diet Problem](#)
- [AIMMS - Farm Planning Problem](#)
- [AIMMS - Pooling Problem](#)



## Chapter 2

# Algorithms and Complexity

[Youtube! - P versus NP](#)

When considering a problem class, we want to know roughly how difficult the problem is. When considering an algorithm, we want to know roughly how fast the algorithm will run. These are questions that we can answer with complexity theory.

### 2.1 Big-O Notation

**Definition 2** (Big-O). For two functions  $f(n)$  and  $g(n)$ , we say that  $f(n) = O(g(n))$  if there exist positive constants  $c$  and  $n_0$  such that

$$0 \leq f(n) \leq c g(n) \text{ for all } n \geq n_0. \quad (2.1.1)$$

We can also use Big-O to denote a set as

$$O(g(n)) := \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq c g(n), \text{ for all } n \geq n_0\}. \quad (2.1.2)$$

**Example 2:** Consider  $f(n) = 5n^2 + 10n + 7$  and  $g(n) = n^2$ . We want to show that  $f(n) = O(g(n))$ .

Let's try  $c = 22$  and  $n_0 = 1$ . We need to show that Equation 2.1.1 is satisfied.

Note first that we always have

1.  $n^2 \leq n^2$  and therefore  $5n^2 \leq 5n^2$

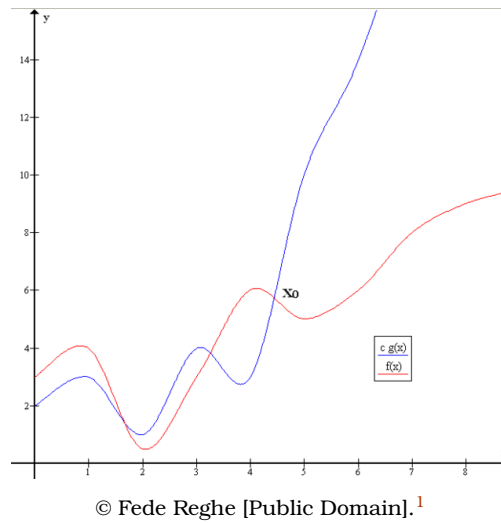
Note that if  $n \geq 1$ , then

2.  $n \leq n^2$  and therefore  $10n \leq 10n^2$

3.  $1 \leq n^2$  and therefore  $7 \leq 7n^2$

---

<sup>1</sup>Big-O Notation, from <https://commons.wikimedia.org/wiki/File:Big-O-notation.png>. Fede Reghe [Public Domain], 2010.



**Figure 2.1:** Example of Big O notation:  $f(x) \in O(g(x))$  as there exists  $c > 0$  (e.g.  $c = 1$ ) and  $x_0$  (e.g.  $x_0 = 5$ ) such that  $f(x) \leq cg(x)$  whenever  $x \geq x_0$ .

Since all inequalities 1,2, and 3 are valid for  $n \geq 1$ , by adding them, we obtain a new inequality that is also valid for  $n \geq 1$ , which is

$$5n^2 + 10n + 7 \leq 5n^2 + 10n^2 + 7n^2 \quad \text{for all } n \geq 1, \quad (2.1.3)$$

$$\Rightarrow 5n^2 + 10n + 7 \leq 22n^2 \quad \text{for all } n \geq 1. \quad (2.1.4)$$

Hence, we have shown that Equation 2.1.1 holds for  $c = 22$  and  $n_0 = 1$ . Hence  $f(n) = O(g(n))$ .

Correct uses:

- $2^n + n^5 + \sin(n) = O(2^n)$
- $2^n = O(n!)$
- $n! + 2^n + 5n = O(n!)$
- $n^2 + n = O(n^3)$
- $n^2 + n = O(n^2)$
- $\log(n) = O(n)$
- $10 \log(n) + 5 = O(n)$

Notice that not all examples above give a tight bound on the asymptotic growth. For instance,  $n^2 + n = O(n^3)$  is true, but a tighter bound is  $n^2 + n = O(n^2)$ .

In particular, the goal of big O notation is to give an upper bound on the asymptotic growth of a function. But we would prefer to give a strong upper bound as opposed to a weak upper bound. For instance, if you order a package online, you will typically be

given a bound on the latest date that it will arrive. For example, if it will arrive within a week, you might be guaranteed that it will arrive by next Tuesday. This sounds like a reasonable bound. But if instead, they tell you it will arrive before 1 year from today, this may not be as useful information. In the case of big O notation, we would like to give a least upper bound that most simply describes the growth behavior of the function.

In that example,  $n^2 + n = O(n^2)$ , this literally means that there is some number  $c$  and some value  $n_0$  that  $n^2 + n \leq cn^2$  for all  $n \geq n_0$ , that is, for all values of  $n_0$  larger than  $n$ , the function  $cn^2$  dominates  $n^2 + n$ .

For example, a valid choice is  $c = 2$  and  $n_0 = 1$ . Then it is true that  $n^2 + n \leq 2n^2$  for all  $n \geq 1$ .

But it is also true that  $n^2 + n = O(n^3)$ . For example, a valid choice is again  $c = 2$  and  $n_0 = 1$ , then

$$n^2 + n \leq 2n^3 \text{ for all } n \geq 1.$$

In this example,  $O(n^3)$  is the case where the internet tells you the package will arrive before 1 year from today. The bound is true, but it is not as useful information as we would like to have. Let's compare these upper bounds. Let  $f(n) = n^2 + n$ ,  $g(n) = 2n^2$ ,  $h(n) = 2n^3$ .

Then we have

	n = 10 .	n = 100 .	n = 1000 .	n = . 10000
f(n)	110,	10100,	1001000,	100010000
g(n)	200,	20000,	2000000,	200000000
h(n) .	2000,	2000000,	2000000000,	2000000000000

So, here we see that  $g(n)$  and  $h(n)$  are both upper bounds on  $f(n)$ , but the nice part about  $g(n)$  is that it is growing at a similar rate to  $f(n)$ . In particular, it is always within a factor of 2 of  $f(n)$ .

Alternatively, the bound  $h(n)$  is true, but it grows so much faster than  $f(n)$  that it doesn't give a good idea of the asymptotic growth of  $f(n)$ .

Some common classes of functions:

$O(1)$	Constant
$O(\log(n))$	Logarithmic
$O(n)$	Linear
$O(n^c)$ (for $c > 1$ )	Polynomial
$O(c^n)$ (for $c > 1$ )	Exponential

**Exponential Time Algorithms do not currently solve reasonable-sized problems in reasonable time.**

Polynomial	$\log n$	3	4
	$n$	10	20
	$n \log n$	33	86
	$n^2$	100	400
	$n^3$	1,000	8,000
	$n^5$	100,000	3,200,000
	$n^{10}$	10,000,000,000	10,240,000,000,000
Exponential	$n$	10	20
	$n^{\log n}$	2,099	419,718
	$2^n$	1,024	1,048,576
	$5^n$	9,765,625	95,367,431,640,625
	$n!$	3,628,800	2,432,902,008,176,640,000
	$n^n$	10,000,000,000	104,857,600,000,000,000,000,000,000

## 2.2 Algorithms - Example with Bubble Sort

The following definition comes from Merriam-Webster's dictionary.

**Definition 3.** *An algorithm is a procedure for solving a mathematical problem in a finite number of steps that frequently involves repetition of an operation; broadly: a step-by-step procedure for solving a problem or accomplishing some end.*

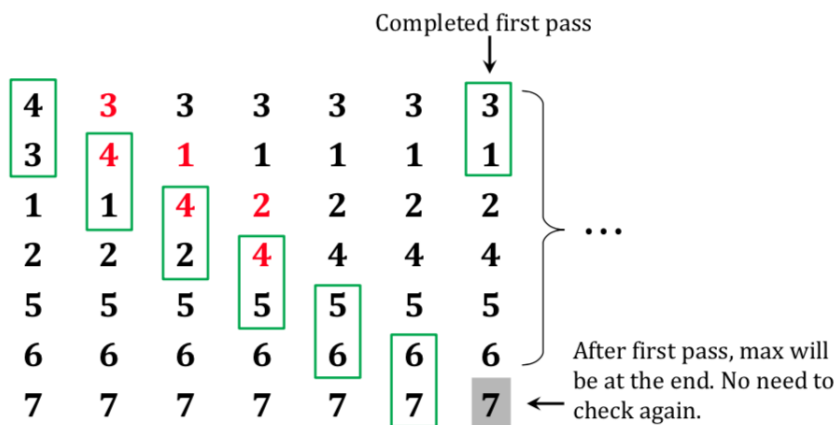
### 2.2.1 Sorting

[Wikipedia](#)

Bubble sort algorithm:.....

#### Bubble sort algorithm

- Compare two neighboring objects and swap if they are in the wrong order.



The algorithm will terminate when a pass is completed with no swaps.

How many steps did it take to sort the numbers in this example?

**15 steps**

This is around  $2n$  ( $n = 7$ ). Does this mean it is  $O(n)$ ?

**NO!**

The **worst case** must be examined.

In the worst case, the minimum element will be at the end of the list.

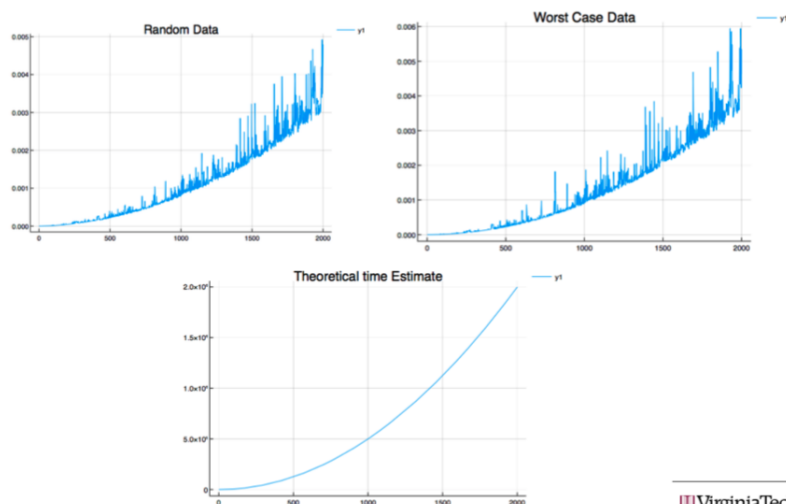
The number of steps in this case will be:

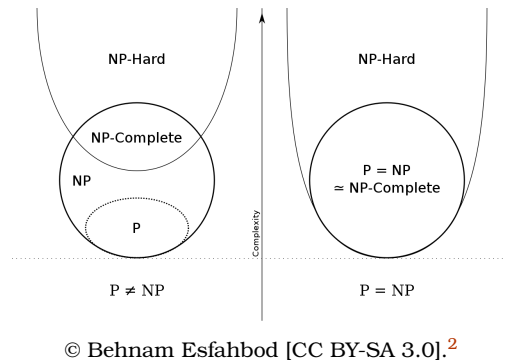
$$(n - 1) + (n - 2) + \cdots + 2 + 1 = \frac{n(n - 1)}{2} = O(n^2)$$

The Bubble Sort is an  $O(n^2)$  algorithm

These can be verified experimentally as seen in the following plot. The random case grows quadratically just as the worst case does.

**Time elapsed in computer for bubble sort**





**Figure 2.2:** Complexity class possibilities. Most academics agree that the case  $P \neq NP$  is more likely.

## 2.3 Complexity Classes

In this subsection we will discuss the complexity classes P, NP, NP-Complete, and NP-Hard. These classes help measure how difficult a problem is. **Informally**, these classes can be thought of as

- P - the class of efficiently solvable problems
- NP - the class of efficiently checkable problems
- NP-Hard - the class of problems that can solve any problem in NP
- NP-Complete - the class of problems that are in both NP and are NP-Hard.

It is not known if  $P$  is the same as NP, but it is conjectured that these are very different classes. This would mean that the NP-Hard problems (and NP-Complete problems) are necessarily much more difficult than the problems in P. See [Figure 2.2](#).

We will now discuss these classes more formally.

### 2.3.1 P

P is the class of polynomially solvable problems. P contains all problems for which there exists an algorithm that solves the problem in a run time bounded by a polynomial. That is,  $O(n^c)$  for some constant  $c$ .

**Example 3:** The minimum size spanning tree problem is in P. It can be solved, for instance, by Prim's algorithm, which runs in time  $O(m \log n)$ , where  $m$  is the number of edges in the graph and  $n$  is the number of nodes in the graph.

<sup>2</sup>Euler diagram for P, NP, NP-complete, and NP-hard set of problems. Under the assumption that  $P \neq NP$ , the existence of problems within NP but outside both P and NP-complete was established by Ladner. from <https://commons.wikimedia.org/wiki/File:TriangleInequality.svg>. Behnam Esfahbod [CC BY-SA 3.0]., 2007.

**Example 4:** Linear programming is in P. It can be solved by interior point methods in  $O(n^{3.5}\phi)$  where  $\phi$  represents the number of binary bits that are required to encode the problem. These bits describe the matrix  $A$ , and vectors  $c$  and  $b$  that define the linear program.

### 2.3.2 NP

NP is the class of nondeterministic polynomial problems. NP contains all problems in which membership can be verified in polynomial time from a certificate.

Thus, to show that a problem is in NP, you must do the following:

1. Describe a format for a certificate to the problem.
2. Show that given such a certificate, it is easy to verify the solution to the problem.

**Example 5:** Integer Linear Programming is in NP. More explicitly, the feasibility question of

"Does there exists an integer vector  $x$  that satisfies  $Ax \leq b$ "

is in NP.

Although it turns out to be difficult to find such an  $x$  or even prove that one exists, this problem is in NP for the following reason: if you are given a particular  $x$  and someone claims to you that it is a feasible solution to the problem, then you can easily check if they are correct. In this case, the vector  $x$  that you were given is called a *certificate*.

Note that it is easy to verify if  $x$  is a solution to the problem because you just have to

1. Check if  $x$  is integer.
2. Use matrix multiplication to check if  $Ax \leq b$  holds.

### 2.3.3 NP-Hard

The class of problems that are called *NP-Hard* are those that can be used to solve any other problem in the NP class. That is, problem A is NP-Hard provided that for any problem B in NP there is a transformation of problem B that preserves the size of the problem, up to a polynomial factor, into a new problem that problem A can be used to solve.

Here we think of "if problem A could be solved efficiently, then all problems in NP could be solved efficiently".

More specifically, we assume that we have an oracle for problem A that runs in polynomial time. An oracle is an algorithm that for the problem that returns the solution of

the problem in a time polynomial in the input. This oracle can be thought of as a magic computer that gives us the answer to the problem. Thus, we say that problem A is NP-Complete provided that given an oracle for problem A, one can solve any other problem B in NP in polynomial time.

Note: These problems are not necessarily in NP.

### 2.3.4 NP-Complete

The class of problems that are call *NP-Complete* are those which are in NP and also NP-Hard.

We know of many problems that are NP-Complete. For example, binary integer programming feasibility is NP-Complete. One can show that another problem is NP-complete by

1. showing that it can be used to solve binary integer programming feasibility,
2. showing that the problem is in NP.

The first problem proven to be NP-Complete is called 3-SAT []. 3-SAT is a special case of the *satisfiability problem*. In a satisfiability problem, we have variables  $X_1, \dots, X_n$  and we want to assign them values as either `true` or `false`. The problem is described with *AND* operations, denoted as  $\wedge$ , with *OR* operations, denoted as  $\vee$ , and with *NOT* operations, denoted as  $\neg$ . The *AND* operation  $X_1 \wedge X_2$  returns `true` if BOTH  $X_1$  and  $X_2$  are true. The *OR* operation  $X_1 \vee X_2$  returns `true` if AT LEAST ONE OF  $X_1$  and  $X_2$  are true. Lastly, the *NOT* operation  $\neg X_1$  returns there opposite of the value of  $X_1$ .

These can be described in the following table

$$\text{true} \wedge \text{true} = \text{true} \quad (2.3.1)$$

$$\text{true} \wedge \text{false} = \text{false} \quad (2.3.2)$$

$$\text{false} \wedge \text{false} = \text{false} \quad (2.3.3)$$

$$\text{false} \wedge \text{true} = \text{false} \quad (2.3.4)$$

$$\text{true} \vee \text{true} = \text{true} \quad (2.3.5)$$

$$\text{true} \vee \text{false} = \text{true} \quad (2.3.6)$$

$$\text{false} \vee \text{false} = \text{false} \quad (2.3.7)$$

$$\text{false} \vee \text{true} = \text{true} \quad (2.3.8)$$

$$\neg \text{true} = \text{false} \quad (2.3.9)$$

$$\neg \text{false} = \text{true} \quad (2.3.10)$$

For example, **Missing code here** A *logical expression* is a sequence of logical operations on variables  $X_1, \dots, X_n$ , such that

$$(X_1 \wedge \neg X_2 \vee X_3) \wedge (X_1 \vee \neg X_3) \vee (X_1 \wedge X_2 \wedge X_3). \quad (2.3.11)$$



A *clause* is a logical expression that only contains the operations  $\vee$  and  $\neg$  and is not nested (with parentheses), such as

$$X_1 \vee \neg X_2 \vee X_3 \vee \neg X_4. \quad (2.3.12)$$

A fundamental result about logical expressions is that they can always be reduced to a sequence of clauses that are joined by  $\wedge$  operations, such as

$$(X_1 \vee \neg X_2 \vee X_3 \vee \neg X_4) \wedge (X_1 \vee X_2 \vee X_3) \wedge (X_2 \vee \neg X_3 \vee \neg X_4 \vee X_5). \quad (2.3.13)$$

The satisfiability problem takes as input a logical expression in this format and asks if there is an assignment of `true` or `false` to each variable  $X_i$  that makes the expression `true`. The 3-SAT problem is a special case where the clauses have only three variables in them.

**3-SAT:**

*NP-Complete*

Given a logical expression in  $n$  variables where each clause has only 3 variables, decide if there is an assignment to the variables that makes the expression `true`.

**Binary Integer Programming:**

*NP-Complete*

Binary Integer Programming can easily be shown to be in NP, since verifying solutions to BIP can be done by checking a linear system of inequalities.

Furthermore, it can be shown to be NP-Complete since it can be used to solve 3-SAT. That is, given an oracle for BIP, since 3-SAT can be modeled as a BIP, the 3-SAT could be solved in oracle-polynomial time.

## 2.4 Relevant Terminology

We will discuss the following concepts:

- Feasible solutions
- Optimal solutions
- Approximate solutions
- Heuristics
- Exact Algorithms
- Approximation Algorithms
- Complexity class relations

## 2.5 Matching Problem

**Definition 4.** Given a graph  $G = (V, E)$ , a matching is a subset  $E' \subseteq E$  such that no vertex  $v \in V$  is contained in more than one edge in  $E'$ .

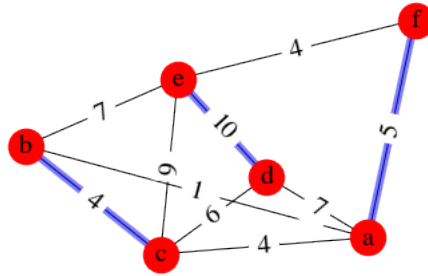
A perfect matching is a matching where every vertex is connected to an edge in  $E'$ .

A maximal matching is a matching  $E'$  such that there is no matching  $E''$  that strictly contains it.

**To Do #1:** INCLUDE PICTURES OF MATCHINGS

**Figure 2.3:** Two possible matchings. On the left, we have a perfect matching (all nodes are matched). On the right, a feasible matching, but not a perfect matching since not all nodes are matched.

**Definition 5.** Maximum Weight Matching Given a graph  $G = (V, E)$ , with associated weights  $w_e \geq 0$  for all  $e \in E$ , a maximum weight matching is a matching that maximizes the sum of the weights in the matching.



### 2.5.1 Greedy Algorithm for Maximal Matching

The greedy algorithm iteratively adds the edge with largest weight that is feasible to add.

**Greedy Algorithm for Maximal Matching:**

Complexity:  $O(|E| \log(|V|))$

1. Begin with an empty graph ( $M = \emptyset$ )
2. Label the edges in the graph such that  $w(e_1) \geq w(e_2) \geq \dots \geq w(e_m)$
3. For  $i = 1, \dots, m$   
 If  $M \cup \{e_i\}$  is a valid matching (i.e., no vertex is incident with two edges), then set  $M \leftarrow M \cup \{e_i\}$  (i.e., add edge  $e_i$  to the graph  $M$ )
4. Return  $M$

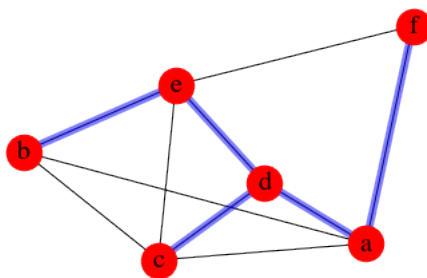
**Theorem 6** ([6]). *The greedy algorithm finds a 2-approximation of the maximum weighted matching problem. That is,  $w(M_{\text{greedy}}) \geq \frac{1}{2}w(M^*)$ .*

### 2.5.2 Other algorithms to look at

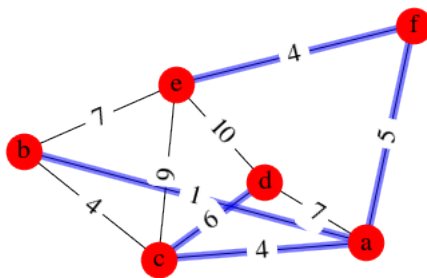
1. Improved Algorithm [9]
2. Blossom Algorithm [Wikipedia](#)

## 2.6 Minimum Spanning Tree

**Definition 7.** *Given a graph  $G = (V, E)$ , a spanning tree connected, acyclic subgraph  $T$  that contains every node in  $V$ .*



**Definition 8.** *Given a graph  $G = (V, E)$ , with associated weights  $w_e \geq 0$  for all  $e \in E$ , a maximum weight spanning tree is a spanning tree maximizes the sum of the edge weights.*



**Lemma 9.** *Let  $G$  be a connected graph with  $n$  vertices.*

1.  *$T$  is a spanning tree of  $G$  if and only if  $T$  has  $n - 1$  edges and is connected.*
2. *Any subgraph  $S$  of  $G$  with more than  $n - 1$  edges contains a cycle.*

See Section 4.2 for integer programming formulations of this problem.

### 2.6.1 Kruskal's algorithm

[Wikipedia](#)

**Kruskal - for Minimum Spanning tree:**Complexity:  $O(|E| \log(|V|))$ 

1. Begin with an empty tree ( $T = \emptyset$ )
2. Label the edges in the graph such that  $w(e_1) \leq w(e_2) \leq \dots \leq w(e_m)$
3. For  $i = 1, \dots, m$   
If  $T \cup \{e_i\}$  is acyclic, then set  $T \leftarrow T \cup \{e_i\}$
4. Return  $T$

**2.6.2 Prim's Algorithm**[Wikipedia](#)[TeXample](#)**2.7 Traveling Salesman Problem**

Some videos:

- [Youtube!](#)
- [Youtube!](#)
- [Youtube!](#)

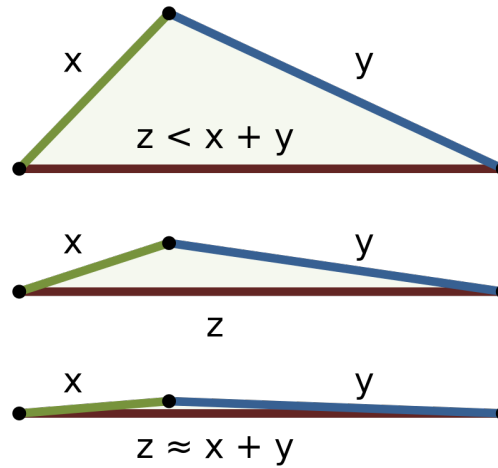
See Section 4.3 for integer programming formulations of this problem. Also, hill climbing algorithms for this problem such as 2-Opt, simulated annealing, and tabu search will be discussed in Section ??.

**2.7.1 Nearest Neighbor - Construction Heuristic**

[https://en.wikipedia.org/wiki/Nearest\\_neighbour\\_algorithm](https://en.wikipedia.org/wiki/Nearest_neighbour_algorithm) Starting from any node, add the edge to the next closest node. Continue this process.

**Nearest Neighbor:**Complexity:  $O(n^2)$ 

1. Start from any node (lets call this node 1) and label this as your current node.
2. Pick the next current node as the one that is closest to the current node that has not yet been visited.
3. Repeat step 2 until all nodes are in the tour.



© WhiteTimberwolf [CC BY-SA 3.0].<sup>3</sup>

**Figure 2.4:** The triangle inequality: the sum of the lengths of two sides of a triangle exceeds the length of the third side.

### 2.7.2 Double Spanning Tree - 2-Apx

Graphs with nice properties are often easier to handle and typically graphs found in the real world have some nice properties. The *triangle inequality* comes from the idea of a triangle that the sum of the lengths of two sides always are longer than the length of the third side. See [Figure 2.4](#)

**Definition 10.** A complete, weighted graph  $G$  (i.e., a graph that has all possible edges and a weight assigned to each edge) satisfies the triangle inequality provided that for every triple of vertices  $a, b, c$  and edges  $e_{ab}, e_{bc}, e_{ac}$ , we have that

$$w(e_{ab}) + w(e_{bc}) \geq w(e_{ac}).$$

---

#### Algorithm 1 Double Spanning Tree

---

**Input:** A graph  $G = (V, E)$  that satisfies the triangle inequality

**Output:** A tour that is a 2-Apx of the optimal solution

- 1: Compute a minimum spanning tree  $T$  of  $G$ .
  - 2: Double each edge in the minimum spanning tree (i.e., if edge  $e_{ab}$  is in  $T$ , add edge  $e_{ba}$ ).
  - 3: Compute an Eulerian Tour using these edges.
  - 4: Return tour that visits vertices in the order the Eulerian Tour visits them, but without repeating any vertices.
- 

Let  $S$  be the resulting tour and let  $S^*$  be an optimal tour. Since the resulting tour is feasible, it will satisfy

$$w(S^*) \leq w(S).$$

---

<sup>3</sup>The triangle inequality: the sum of the lengths of two sides of a triangle exceeds the length of the third side. from <https://commons.wikimedia.org/wiki/File:TriangleInequality.svg>. WhiteTimberwolf [CC BY-SA 3.0]., 2013.

But we also know that the weight of a minimum spanning tree  $T$  is less than that of the optimal tour, hence

$$w(T) \leq w(S^*).$$

Lastly, due to the triangle inequality we know that

$$w(S) \leq 2w(T),$$

since replacing any edge in the Eulerian tour with a more direct edge only reduces the total weight.

Putting this together, we have

$$w(S) \leq 2w(T) \leq 2w(S^*)$$

and hence,  $S$  is a 2-approximation of the optimal solution.

### 2.7.3 Christofides - Approximation Algorithm - $(3/2)$ -Apx

If we combine algorithms for minimum spanning tree and matching, we can find a better approximation algorithm. This is given by Christofides. Again, this is in the case where the graph satisfies the triangle inequality. See [Wikipedia - Christofides Algorithm](#) or [Ola Svensson Lecture Slides](#) for more detail.

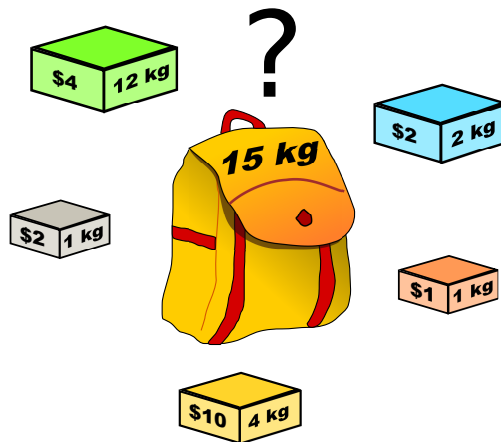
## Chapter 3

# Integer Programming Formulations

### 3.1 Knapsack Problem

Video! - Michel Belaire (EPFL) teaching knapsack problem

Knapsack problem can take different forms depending on if the variables are binary or integer. The binary version means that there is only one item of each item type that can be taken. This is typically illustrated as a backpack (knapsack) and some items to put into it (see [Figure 3.1](#)), but has applications in many contexts.



© See page for author [CC BY-SA 2.5].<sup>1</sup>

**Figure 3.1:** *Knapsack Problem: which items should we choose take in the knapsack that maximizes the value while respecting the 15kg weight limit?*

<sup>1</sup>Image of backpack and four blocks with different weight and value. from [https://en.wikipedia.org/wiki/Knapsack\\_problem#/media/File:Knapsack.svg](https://en.wikipedia.org/wiki/Knapsack_problem#/media/File:Knapsack.svg). See page for author [CC BY-SA 2.5]., 2007.

**Binary Knapsack Problem:***NP-Complete*

Given an non-negative weight vector  $a \in \mathbb{Q}_+^n$ , a capacity  $b \in \mathbb{Q}_+$ , and objective coefficients  $c \in \mathbb{Q}^n$ ,

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & a^\top x \leq b \\ & x \in \{0,1\}^n \end{aligned} \tag{3.1.1}$$

**Example: Knapsack**

[Code: ??]

You have a knapsack (bag) that can only hold  $W = 15$  kgs. There are 5 items that you could possibly put into your knapsack. The items (weight, value) are given as: (12 kg, \$4), (2 kg, \$2), (1kg, \$2), (1kg, \$1), (4kg, \$10). Which items should you take to maximize your value in the knapsack? See Figure 3.1.

**Variables:**

- let  $x_i = 0$  if item  $i$  is in the bag
- let  $x_i = 1$  if item  $i$  is not in the bag

**Model:**

$$\begin{aligned} \max \quad & 4x_1 + 2x_2 + 2x_3 + 1x_4 + 10x_5 \\ \text{s.t.} \quad & 12x_1 + 2x_2 + 1x_3 + 1x_4 + 4x_5 \leq 15 \\ & x_i \in \{0,1\} \text{ for } i = 1, \dots, 5 \end{aligned} \tag{3.1.2}$$

In the integer case, we typically require the variables to be non-negative integers, hence we use the notation  $x \in \mathbb{Z}_+^n$ . This setting reflects the fact that instead of single individual items, you have item types of which you can take as many of each type as you like that meets the constraint.

**Integer Knapsack Problem:***NP-Complete*

Given an non-negative weight vector  $a \in \mathbb{Q}_+^n$ , a capacity  $b \in \mathbb{Q}_+$ , and objective coefficients  $c \in \mathbb{Q}^n$ ,

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & a^\top x \leq b \\ & x \in \mathbb{Z}_+^n \end{aligned} \tag{3.1.3}$$



We can also consider an equality constrained version

**Equality Constrained Integer Knapsack Problem:**

*NP-Hard*

Given an non-negative weight vector  $a \in \mathbb{Q}_+^n$ , a capacity  $b \in \mathbb{Q}_+$ , and objective coefficients  $c \in \mathbb{Q}^n$ ,

$$\max \quad c^\top x \quad (3.1.4)$$

$$\text{s.t.} \quad a^\top x = b \quad (3.1.5)$$

$$x \in \mathbb{Z}_+^n \quad (3.1.6)$$

**Example 7:** Using pennies, nickels, dimes, and quarters, how can you minimize the number of coins you need to make up a sum of 83¢?

**Variables:**

- Let  $p$  be the number of pennies used
- Let  $n$  be the number of nickels used
- Let  $d$  be the number of dimes used
- Let  $q$  be the number of quarters used

**Model**

$$\begin{array}{ll} \min & p + n + d + q & \text{total number of coins used} \\ \text{s.t.} & p + 5n + 10d + 25q = 83 & \text{sums to 83¢} \\ & p, d, n, q \in \mathbb{Z}_+ & \text{each is a non-negative integer} \end{array}$$

## 3.2 Set Covering

Video! - Michel Belaire (EPFL) explaining set covering problem

**Set Covering:**

*NP-Complete*

Given a set  $V$  with subsets  $V_1, \dots, V_l$ , determine the smallest subset  $S \subseteq V$  such that  $S \cap V_i \neq \emptyset$  for all  $i = 1, \dots, l$ .

The set cover problem can be modeled as

$$\begin{aligned}
 \min \quad & \mathbb{1}^\top x \\
 \text{s.t.} \quad & \sum_{v \in V_i} x_v \geq 1 \text{ for all } i = 1, \dots, l \\
 & x_v \in \{0, 1\} \text{ for all } v \in V
 \end{aligned} \tag{3.2.1}$$

where  $x_v$  is a 0/1 variable that takes the value 1 if we include item  $j$  in set  $S$  and 0 if we do not include it in the set  $S$ .

**To Do #2:** Add flight crew scheduling example and images.

One specific type of set cover problem is the *vertex cover* problem.

**Vertex Cover:**

*NP-Complete*

Given a graph  $G = (V, E)$  of vertices and edges, we want to find a smallest size subset  $S \subseteq V$  such that every for every  $e = (v, u) \in E$ , either  $u$  or  $v$  is in  $S$ .

We can write this as a mathematical program in the form:

$$\begin{aligned}
 \min \quad & \mathbb{1}^\top x \\
 \text{s.t.} \quad & x_u + x_v \geq 1 \text{ for all } (u, v) \in E \\
 & x_v \in \{0, 1\} \text{ for all } v \in V.
 \end{aligned} \tag{3.2.2}$$

**To Do #3:** Add fire station cover example.

**Set Covering - Matrix description:**

*NP-Complete*

Given a non-negative matrix  $A \in \{0, 1\}^{m \times n}$ , a non-negative vector, and an objective vector  $c \in \mathbb{R}^n$ , the set cover problem is

$$\begin{aligned}
 \max \quad & c^\top x \\
 \text{s.t.} \quad & Ax \geq \mathbb{1} \\
 & x \in \{0, 1\}^n.
 \end{aligned} \tag{3.2.3}$$

**Example: Vertex Cover with matrix**

[Code: ??]

An alternate way to solve ?? is to define the adjacency matrix  $A$  of the graph. The adjacency matrix is a  $|E| \times |V|$  matrix with  $\{0,1\}$  entries. The each row corresponds to an edge  $e$  and each column corresponds to a node  $v$ . For an edge  $e = (u, v)$ , the corresponding row has a 1 in columns corresponding to the nodes  $u$  and  $v$ , and a 0 everywhere else. Hence, there are exactly two 1's per row. Applying the formulation above in **Set Covering - Generalized** solves the problem.

**General Set Covering**

We could also allow for a more general type of set covering where we have non-negative integer variables and a right hand side that has values other than 1.

**Set Covering - Generalized:***NP-Complete*

Given a non-negative matrix  $A \in \mathbb{Z}_+^{m \times n}$ , a non-negative vector  $b \in \mathbb{Z}^m$ , and an objective vector  $c \in \mathbb{R}^n$ , the set cover problem is

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \geq b \\ & x \in \mathbb{Z}_+^n. \end{aligned} \tag{3.2.4}$$

**To Do #4:** Add Nurse scheduling problem.

**Example: Nurse Scheduling**

[Code: ??]

**3.3 Machine Assignment**

Example Model: Machine assignment

- Given  $m$  machines and  $n$  jobs, find a least cost assignment of jobs to machines not exceeding the machine capacities.
- Each job  $j$  requires  $a_{ij}$  units of machine  $i$ 's capacity  $b_i$ .
- Cost of assigning job  $j$  to machine  $i$  is  $c_{ij}$ .

Here is a model.... **To be added....**

**To Do #5:** Include picture

### 3.4 Facility Location

**To Do #6:** Add discussion on Facility Location Problems and pictures.

[Wikipedia - Facility Location Problem](#)

#### 3.4.1 Capacitated Facility Location

**Capacitated Facility Location:**

*NP-Complete*

Given costs connections  $c_{ij}$  and fixed building costs  $f_i$ , demands  $d_j$  and capacities  $u_i$ , the capacitated facility location problem is

$$\begin{aligned}
 \min \quad & \sum_{i=1}^n \sum_{j=1}^m c_{ij} y_{ij} + \sum_{i=1}^n f_i x_i \\
 \text{s.t.} \quad & \sum_{i=1}^n y_{ij} = 1 \text{ for all } j = 1, \dots, m \\
 & \sum_{j=1}^m d_j y_{ij} \leq u_i x_i \text{ for all } i = 1, \dots, n \\
 & y_{ij} \geq 0 \text{ for all } i = 1, \dots, n \text{ and } j = 1, \dots, m \\
 & x_i \in \{0, 1\} \text{ for all } i = 1, \dots, n
 \end{aligned} \tag{3.4.1}$$

Here  $M$  is a large number and can be chosen as  $M = m$ , but could be refined smaller if more context is known.

#### 3.4.2 Uncapacitated Facility Location

**Uncapacitated Facility Location:**

*NP-Complete*

Given costs connections  $c_{ij}$  and fixed building costs  $f_i$ , the uncapacitated facility location problem is

$$\begin{aligned}
 \min \quad & \sum_{i=1}^n \sum_{j=1}^m c_{ij} z_{ij} + \sum_{i=1}^n f_i x_i \\
 \text{s.t.} \quad & \sum_{i=1}^n z_{ij} = 1 \text{ for all } j = 1, \dots, m \\
 & \sum_{j=1}^m z_{ij} \leq M x_i \text{ for all } i = 1, \dots, n \\
 & z_{ij} \in \{0, 1\} \text{ for all } i = 1, \dots, n \text{ and } j = 1, \dots, m \\
 & x_i \in \{0, 1\} \text{ for all } i = 1, \dots, n
 \end{aligned} \tag{3.4.2}$$

Here  $M$  is a large number and can be chosen as  $M = m$ , but could be refined smaller if more context is known.

### 3.5 Capital Budgeting

A firm has  $n$  projects it could undertake to maximize revenue, but budget limitations require that not all can be completed.

Project  $j$  expects to produce revenue  $c_j$

Project  $j$  requires investment  $a_{ij}$  in time period  $i$  for  $i = 1, \dots, m$

In time period  $i$ , capital  $b_i$  is available

Let  $x_i$  be a binary variable such that  $x_i = 1$  if we choose investment  $i$  and  $x_i = 0$  otherwise. The the model can be given as:

$$\begin{aligned} \max \quad & \sum_{j=1}^n c_j x_j \\ \text{s.t.} \quad & \sum_{j=1}^n a_{ij} x_j \leq b_i, \quad i = 1, \dots, m \\ & x_j \in \{0, 1\}, \quad j = 1, \dots, n \end{aligned}$$

Consider the example given in the following table.

Project	$\mathbb{E}[\text{Revenue}]$	Resources required in week 1	Resources required in week 2
1	10	3	4
2	8	1	2
3	6	2	1
Resources available		5	6

$$\max \quad 10x_1 + 8x_2 + 6x_3$$

subject to

$$\begin{aligned} 3x_1 + 1x_2 + 2x_3 &\leq 5 \\ 4x_1 + 2x_2 + 1x_3 &\leq 6 \\ x_j &\in \{0, 1\}, \quad j = 1, 2, 3 \end{aligned}$$

### 3.6 Network Flow

**To Do #7:** Add discussion of network flow problem and picture.

### 3.7 Transportation Problem

**To Do #8:** Add discussion of transportation problem and picture.

Youtube! - TRANSPORTATION PROBLEM with PuLP in PYTHON

### 3.8 Makespan Minimization

**To Do #9:** Add discussion of some makespan minimization problems.

### 3.9 Other examples

- [Sudoku](#)

### 3.10 Modeling Tricks

In this section, we describe ways to model a variety of constraints that commonly appear in practice. The goal is changing constraints described in words to constraints defined by math.

#### 3.10.1 Either Or Constraints

“At least one of these constraints holds” is what we would like to model. Equivalently, we can phrase this as an *inclusive or* constraint.

**Either Or:**

$$\text{Either } a^\top x \leq b \text{ or } c^\top x \leq d \text{ holds} \quad (3.10.1)$$

can be modeled as

$$\begin{aligned} a^\top x - b &\leq M_1 \delta \\ c^\top x - d &\leq M_2(1 - \delta) \\ \delta &\in \{0, 1\}, \end{aligned} \quad (3.10.2)$$

where  $M_1$  is an upper bound on  $a^\top x - b$  and  $M_2$  is an upper bound on  $c^\top x - d$ .

**Example 10:** Either 2 buses or 10 cars are needed shuttle students to the football game.

- Let  $x$  be the number of buses we have and
- let  $y$  be the number of cars that we have.

Suppose that there are at most  $M_1 = 5$  buses that could be rented and at most  $M_2 = 20$  cars that could be available.

This constraint can be modeled as

$$\begin{aligned} x - 2 &\leq 5\delta \\ y - 10 &\leq 20(1 - \delta) \\ \delta &\in \{0, 1\}, \end{aligned} \tag{3.10.3}$$

### If then implications

If then implications are extremely useful in models. For instance, if we have more than 5 passengers, then we need to take two cars. Most if then statements can be modeled with by a constraint and an on/off flag. For example

$$\text{If } \delta = 1, \text{ then } a^\top x \leq b. \tag{3.10.4}$$

By letting  $M$  be an upper bound on the quantity  $a^\top x - b$ , we can model this condition as

$$\begin{aligned} a^\top x - b &\leq M(1 - \delta) \\ \delta &\in \{0, 1\} \end{aligned} \tag{3.10.5}$$

On the other hand, if we want to model the reverse implication, we have to be slightly more careful. We let  $m$  be a lower bound on the quantity  $a^\top x - b$  and we let  $\epsilon$  be a tiny number that is an error bound in verifying if an inequality is violated. **If the data  $a, b$  are integer and  $x$  is an integer, then we can take  $\epsilon = 1$ .**

Now

$$\text{If } a^\top x \leq b \text{ then } \delta = 1 \tag{3.10.6}$$

can be modeled as

$$a^\top x - b \geq \epsilon(1 - \delta) + m\delta. \tag{3.10.7}$$

A simple way to understand this constraint is to consider the *contrapositive* of the if then statement that we want to model. The contrapositive says that

$$\text{If } \delta = 0, \text{ then } a^\top x - b > 0. \tag{3.10.8}$$

To show the contrapositive, we set  $\delta = 0$ . Then the inequality becomes

$$a^\top x - b \geq \epsilon(1 - 0) + m0 = \epsilon > 0.$$

Thus, the contrapositive holds.

**If instead we wanted a direct proof:**

Case 1: Suppose  $a^\top x \leq b$ . Then  $0 \geq a^\top x - b$ , which implies that

$$\delta(a^\top x - b) \geq a^\top x - b$$

Therefore

$$\delta(a^\top x - b) \geq \epsilon(1 - \delta) + m\delta$$

After rearranging

$$\delta(a^\top x - b - m) \geq \epsilon(1 - \delta)$$

Implication	Constraint
If $\delta = 0$ , then $a^\top x \leq b$	$a^\top x \leq b + M\delta$
If $a^\top x \leq b$ , then $\delta = 1$	$a^\top x \geq m\delta + \epsilon(1 - \delta)$

**Table 3.1:** Short list: If/then models with a constraint and a binary variable. Here  $M$  and  $m$  are upper and lower bounds on  $a^\top x - b$  and  $\epsilon$  is a small number such that if  $a^\top x > b$ , then  $a^\top x \geq b + \epsilon$ .

Implication	Constraint
If $\delta = 0$ , then $a^\top x \leq b$	$a^\top x \leq b + M\delta$
If $\delta = 0$ , then $a^\top x \geq b$	$a^\top x \geq b + m\delta$
If $\delta = 1$ , then $a^\top x \leq b$	$a^\top x \leq b + M(1 - \delta)$
If $\delta = 1$ , then $a^\top x \geq b$	$a^\top x \geq b + m(1 - \delta)$
If $a^\top x \leq b$ , then $\delta = 1$	$a^\top x \geq b + m\delta + \epsilon(1 - \delta)$
If $a^\top x \geq b$ , then $\delta = 1$	$a^\top x \leq b + M\delta - \epsilon(1 - \delta)$
If $a^\top x \leq b$ , then $\delta = 0$	$a^\top x \geq b + m(1 - \delta) + \epsilon\delta$
If $a^\top x \geq b$ , then $\delta = 0$	$a^\top x \leq b + m(1 - \delta) - \epsilon\delta$

**Table 3.2:** Long list: If/then models with a constraint and a binary variable. Here  $M$  and  $m$  are upper and lower bounds on  $a^\top x - b$  and  $\epsilon$  is a small number such that if  $a^\top x > b$ , then  $a^\top x \geq b + \epsilon$ .

Since  $a^\top x - b - m \geq 0$  and  $\epsilon > 0$ , the only feasible choice is  $\delta = 1$ .

Case 2: Suppose  $a^\top x > b$ . Then  $a^\top x - b \geq \epsilon$ . Since  $a^\top x - b \geq m$ , both choices  $\delta = 0$  and  $\delta = 1$  are feasible.

By the choice of  $\epsilon$ , we know that  $a^\top x - b > 0$  implies that  $a^\top x - b \geq \epsilon$ .

—

Since we don't like strict inequalities, we write the strict inequality as  $a^\top x - b \geq \epsilon$  where  $\epsilon$  is a small positive number that is a smallest difference between  $a^\top x - b$  and 0 that we would typically observe. As mentioned above, if  $a, b, x$  are all integer, then we can use  $\epsilon = 1$ .

Now we want an inequality with left hand side  $a^\top x - b \geq$  and right hand side to take the value

- $\epsilon$  if  $\delta = 0$ ,
- $m$  if  $\delta = 1$ .

This is accomplished with right hand side  $\epsilon(1 - \delta) + m\delta$ .

Many other combinations of if then statements are summarized in the following table: These two implications can be used to derive the following longer list of implications.

Lastly, if you insist on having exact correspondance, that is, " $\delta = 0$  if and only if  $a^\top x \leq b$ " you can simply include both constraints for "if  $\delta = 0$ , then  $a^\top x \leq b$ " and if " $a^\top x \leq b$ , then  $\delta = 0$ ". Although many problems may be phrased in a way that suggests



you need "if and only if", it is often not necessary to use both constraints due to the objectives in the problem that naturally prevent one of these from happening.

For example, if we want to add a binary variable  $\delta$  that means

$$\begin{cases} \delta = 0 \text{ implies } a^\top x \leq b \\ \delta = 1 \text{ Otherwise} \end{cases}$$

If  $\delta = 1$  does not effect the rest of the optimization problem, then adding the constraint regarding  $\delta = 1$  is not necessary. Hence, typically, in this scenario, we only need to add the constraint  $a^\top x \leq b + M\delta$ .

### 3.10.2 Binary reformulation of integer variables

If an integer variable has small upper and lower bounds, it can sometimes be advantageous to recast it as a sequence of binary variables - for either modeling, the solver, or both. Although there are technically many ways to do this, here are the two most common ways.

**Full reformulation:**

*u* many binary variables

For a non-negative integer variable  $x$  with upper bound  $u$ , modeled as

$$0 \leq x \leq u, \quad x \in \mathbb{Z}, \quad (3.10.9)$$

this can be reformulated with  $u$  binary variables  $z_1, \dots, z_u$  as

$$\begin{aligned} x &= \sum_{i=1}^u iz_i = z_1 + 2z_2 + \dots + uz_u \\ 1 &\geq \sum_{i=1}^u z_i = z_1 + z_2 + \dots + z_u \\ z_i &\in \{0, 1\} \quad \text{for } i = 1, \dots, u \end{aligned} \quad (3.10.10)$$

We call this the *full reformulation* because there is a binary variable  $z_i$  associated with every value  $i$  that  $x$  could take. That is, if  $z_3 = 1$ , then the second constraint forces  $z_i = 0$  for all  $i \neq 3$  (that is,  $z_3$  is the only non-zero binary variable), and hence by the first constraint,  $x = 3$ .

**Full reformulation:**

$O(\log u)$  many binary variables

For a non-negative integer variable  $x$  with upper bound  $u$ , modeled as

$$0 \leq x \leq u, \quad x \in \mathbb{Z}, \quad (3.10.11)$$

this can be reformulated with  $u$  binary variables  $z_1, \dots, z_{\log(\lfloor u \rfloor)+1}$  as

$$x = \sum_{i=0}^{\log(\lfloor u \rfloor)+1} 2^i z_i = z_0 + 2z_1 + 4z_2 + 8z_3 + \dots + 2^{\log(\lfloor u \rfloor)+1} z_{\log(\lfloor u \rfloor)+1} \quad (3.10.12)$$

$$z_i \in \{0, 1\} \quad \text{for } i = 1, \dots, \log(\lfloor u \rfloor) + 1$$

We call this the *log reformulation* because this requires only logarithmically many binary variables in terms of the upper bound  $u$ . This reformulation is particularly better than the full reformulation when the upper bound  $u$  is a “larger” number, although we will leave it ambiguous as to how larger a number need to be in order to be described as a “larger” number.

### 3.10.3 SOS1 Constraints

**Definition 11.** A *Special Ordered Sets of type 1 (SOS1)* constraint on a vector indicates that at most one element of the vector can non-zero.

We next give an example of how to use binary variables to model this and then show how much simpler it can be coded using the SOS1 constraint.

#### Example: SOS1 Constraints

[Code: ??]

Solve the following optimization problem:

$$\begin{aligned} &\text{maximize} && 3x_1 + 4x_2 + x_3 + 5x_4 \\ &\text{subject to} && 0 \leq x_i \leq 5 \\ &&& \text{at most one of the } x_i \text{ can be nonzero} \end{aligned}$$

### 3.10.4 SOS2 Constraints

**Definition 12.** A *Special Ordered Sets of type 2 (SOS2)* constraint on a vector indicates that at most two elements of the vector can non-zero AND the non-zero elements must appear consecutively.

We next give an example of how to use binary variables to model this and then show how much simpler it can be coded using the SOS2 constraint.

#### Example: SOS2

[Code: ??]

Solve the following optimization problem:

$$\begin{aligned} &\text{maximize} && 3x_1 + 4x_2 + x_3 + 5x_4 \\ &\text{subject to} && 0 \leq x_i \leq 5 \\ &&& \text{at most two of the } x_i \text{ can be nonzero} \\ &&& \text{and the nonzero } x_i \text{ must be consecutive} \end{aligned}$$

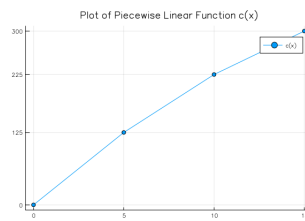
### 3.10.5 Piecewise linear functions with SOS2 constraint

#### Example: Piecewise Linear Function

[Code: ??]

Consider the piecewise linear function  $c(x)$  given by

$$c(x) = \begin{cases} 25x & \text{if } 0 \leq x \leq 5 \\ 20x + 25 & \text{if } 5 \leq x \leq 10 \\ 15x + 75 & \text{if } 10 \leq x \leq 15 \end{cases}$$



We will use integer programming to describe this function. We will fix  $x = a$  and then the integer program will set the value  $y$  to  $c(a)$ .

$$\begin{aligned} \min \quad & 0 \\ \text{Subject to} \quad & x - 5z_2 - 10z_3 - 15z_4 = 0 \\ & y - 125z_2 - 225z_3 - 300z_4 = 0 \\ & z_1 + z_2 + z_3 + z_4 = 1 \\ & \text{SOS2} : \{z_1, z_2, z_3, z_4\} \\ & 0 \leq z_i \leq 1 \quad \forall i \in \{1, 2, 3, 4\} \\ & x = a \end{aligned}$$

#### Example: Piecewise Linear Function Application

[Code: ??]

Consider the following optimization problem where the objective function includes the term  $c(x)$ , where  $c(x)$  is the piecewise linear function described in [Example 13](#):

$$\begin{aligned} \max \quad & z = 12x_{11} + 12x_{21} + 14x_{12} + 14x_{22} - c(x) \\ \text{s.t.} \quad & x_{11} + x_{12} \leq x + 5 \\ & x_{21} + x_{22} \leq 10 \\ & 0.5x_{11} - 0.5x_{21} \geq 0 \\ & 0.4x_{12} - 0.6x_{22} \geq 0 \\ & x_{ij} \geq 0 \\ & 0 \leq x \leq 15 \end{aligned}$$

Given the piecewise linear, we can model the whole problem explicitly as a mixed-integer linear program.

$$\begin{aligned}
 \max \quad & 12X_{1,1} + 12X_{2,1} + 14X_{1,2} + 14X_{2,2} - y \\
 \text{Subject to} \quad & x - 5z_2 - 10z_3 - 15z_4 = 0 \\
 & y - 125z_2 - 225z_3 - 300z_4 = 0 \\
 & z_1 + z_2 + z_3 + z_4 = 1 \\
 & X_{1,1} + X_{1,2} - x \leq 5 \\
 & X_{2,1} + X_{2,2} \leq 10 \\
 & 0.5X_{1,1} - 0.5X_{2,1} \geq 0 \\
 & 0.4X_{1,2} - 0.6X_{2,2} \geq 0 \\
 & \text{SOS2} : \{z_1, z_2, z_3, z_4\} \\
 & X_{i,j} \geq 0 \quad \forall i \in \{1, 2\}, j \in \{1, 2\} \\
 & 0 \leq z_i \leq 1 \quad \forall i \in \{1, 2, 3, 4\} \\
 & 0 \leq x \leq 15 \\
 & y
 \end{aligned}$$

### 3.10.6 Maximizing a minimum

When the constraints could be general, we will write  $x \in X$  to define general constraints. For instance, we could have  $X = \{x \in \mathbb{R}^n : Ax \leq b\}$  or  $X = \{x \in \mathbb{R}^n : Ax \leq b, x \in \mathbb{Z}^n\}$  or many other possibilities.

Consider the problem

$$\begin{aligned}
 \max \quad & \min\{x_1, \dots, x_n\} \\
 \text{such that} \quad & x \in X
 \end{aligned}$$

Having the minimum on the inside is inconvenient. To remove this, we just define a new variable  $y$  and enforce that  $y \leq x_i$  and then we maximize  $y$ . Since we are maximizing  $y$ , it will take the value of the smallest  $x_i$ . Thus, we can recast the problem as

$$\begin{aligned}
 \max \quad & y \\
 \text{such that} \quad & y \leq x_i \text{ for } i = 1, \dots, n \\
 & x \in X
 \end{aligned}$$

### 3.10.7 Relaxing (nonlinear) equality constraints

There are a number of scenarios where the constraints can be relaxed without sacrificing optimal solutions to your problem. In a similar vein of the maximizing a minimum, if because of the objective we know that certain constraints will be tight at optimal solutions,

we can relax the equality to an inequality. For example,

$$\begin{array}{ll} \max & x_1 + x_2 + \cdots + x_n \\ \text{such that} & x_i = y_i^2 + z_i^2 \text{ for } i = 1, \dots, n \end{array}$$

### 3.11 Notes from AIMMS modeling book.

- [AIMMS - Practical guidelines for solving difficult MILPs](#)
- [AIMMS - Linear Programming Tricks](#)
- [AIMMS - Formulating Optimization Models](#)
- [AIMMS - Practical guidelines for solving difficult linear programs](#)

#### 3.11.1 Further Topics

- [Precedence Constraints](#)



## Chapter 4

# Exponential Size Integer Programming Formulations

Although typically models need to be a reasonable size in order for us to code them and send them to a solver, there are some ways that we can allow having models of exponential size. The first example here is the cutting stock problem, where we will model with exponentially many variables. The second example is the traveling salesman problem, where we will model with exponentially many constraints. We will also look at some other models for the traveling salesman problem.

### 4.1 Cutting Stock

This is a classic problem that works excellent for a technique called *column generation*. We will discuss two versions of the model and then show how we can use column generation to solve the second version more efficiently. First, let's describe the problem.

#### **Cutting Stock:**

You run a company that sells of pipes of different lengths. These lengths are  $L_1, \dots, L_k$ . To produce these pipes, you have one machine that produces pipes of length  $L$ , and then cuts them into a collection of shorter pipes as needed.

[Image to add](#)

You have an order come in for  $d_i$  pipes of length  $i$  for  $i = 1, \dots, k$ . How can you fill the order while cutting up the fewest number of pipes?

**Example 15:** A plumber stocks standard lengths of pipe, all of length 19 m. An order arrives for:

- 12 lengths of 4m
- 15 lengths of 5m
- 22 lengths of 6m

How should these lengths be cut from standard stock pipes so as to minimize the number of standard pipes used?

An initial model for this problem could be constructed as follows:

- Let  $N$  be an upper bound on the number of pipes that we may need.
- Let  $z_j = 1$  if we use pipe  $j$  and  $z_j = 0$  if we do not use pipe  $j$ , for  $j = 1, \dots, N$ .
- Let  $x_{ij}$  be the number of cuts of length  $L_i$  in pipe  $j$  that we use.

Then we have the following model

$$\begin{aligned}
 \min \quad & \sum_{j=1}^N z_j \\
 \text{s.t.} \quad & \sum_{i=1}^k L_i x_{ij} \leq L z_j \quad \text{for } j = 1, \dots, N \\
 & \sum_{j=1}^N x_{ij} \geq d_i \quad \text{for } i = 1, \dots, k \\
 & z_j \in \{0, 1\} \quad \text{for } j = 1, \dots, N \\
 & x_{ij} \in \mathbb{Z}_+ \quad \text{for } i = 1, \dots, k, j = 1, \dots, N
 \end{aligned} \tag{4.1.1}$$

**Exercise 13.** In the example above, show that we can choose  $N = 16$ .

For our example above, using  $N = 16$ , we have

$$\begin{aligned}
 \min \quad & \sum_{j=1}^{16} z_j \\
 \text{s.t.} \quad & 4x_{1j} + 5x_{2j} + 6x_{3j} \leq 19z_j \\
 & \sum_{j=1}^{16} x_{1j} \geq 12 \\
 & \sum_{j=1}^{16} x_{2j} \geq 15 \\
 & \sum_{j=1}^{16} x_{3j} \geq 22 \\
 & z_j \in \{0, 1\} \quad \text{for } j = 1, \dots, 16 \\
 & x_{ij} \in \mathbb{Z}_+ \quad \text{for } i = 1, \dots, 3, j = 1, \dots, 16
 \end{aligned} \tag{4.1.2}$$

Additionally, we could break the symmetry in the problem. That is, suppose the solution uses 10 of the 16 pipes. The current formulation does not restrict which 10 pipes are



used. Thus, there are many possible solutions. To reduce this complexity, we can state that we only use the first 10 pipes. We can write a constraint that says *if we don't use pipe  $j$ , then we also will not use any subsequent pipes*. Hence, by not using pipe 11, we enforce that pipes 11, 12, 13, 14, 15, 16 are not used. This can be done by adding the constraints

$$z_1 \geq z_2 \geq z_3 \geq \cdots \geq z_N. \quad (4.1.3)$$

See ?? for code for this formulation.

Unfortunately, this formulation is slow and does not scale well with demand. In particular, the number of variables is  $N + kN$  and the number of constraints is  $N$  (plus integrality and non-negativity constraints on the variables). The solution times for this model are summarized in the following table:

### INPUT TABLE OF COMPUTATIONS

#### 4.1.1 Pattern formulation

We could instead list all patterns that are possible to cut each pipe. A pattern is an vector  $a \in \mathbb{Z}_+^k$  such that for each  $i$ ,  $a_i$  lengths of  $L_i$  can be cut from a pipe of length  $L$ . That is

$$\begin{aligned} \sum_{i=1}^k L_i a_i &\leq L \\ a_i &\in \mathbb{Z}_+ \text{ for all } i = 1, \dots, k \end{aligned} \quad (4.1.4)$$

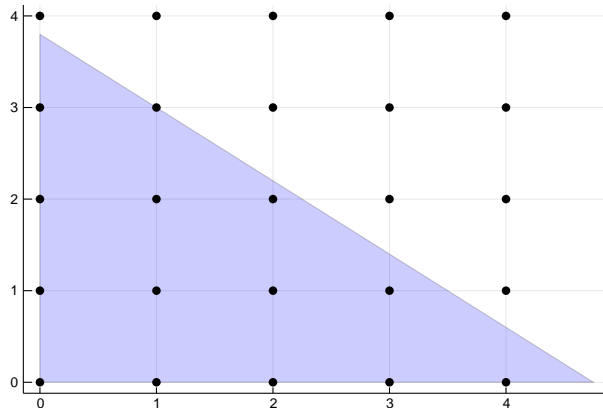
In our running example, we have

$$\begin{aligned} 4a_1 + 5a_2 + 6a_3 &\leq 19 \\ a_i &\in \mathbb{Z}_+ \text{ for all } i = 1, \dots, 3 \end{aligned} \quad (4.1.5)$$

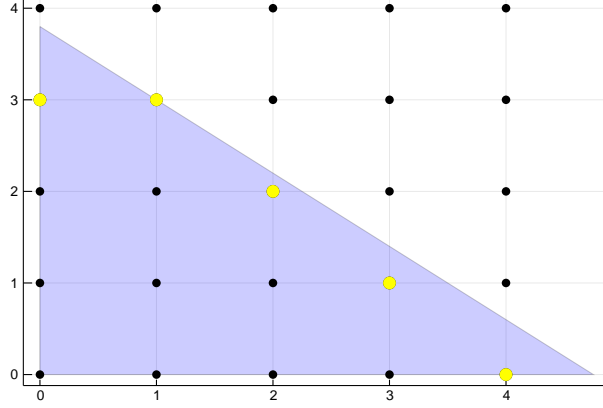
For visualization purposes, consider the patterns where  $a_3 = 0$ . That is, only patterns with cuts of length 4m or 5m. All patterns of this type are represented by an integer point in the polytope

$$P = \{(a_1, a_2) : 4a_1 + 5a_2 \leq 19, a_1 \geq 0, a_2 \geq 0\} \quad (4.1.6)$$

which we can see here:



where  $P$  is the blue triangle and each integer point represents a pattern. Feasible patterns lie inside the polytope  $P$ . Note that we only need patterns that are maximal with respect to number of each type we cut. Pictorially, we only need the patterns that are integer points represented as yellow dots in the picture below.



For example, the pattern  $[3, 0, 0]$  is not needed (only cut 3 of length 4m) since we could also use the pattern  $[4, 0, 0]$  (cut 4 of length 4m) or we could even use the pattern  $[3, 1, 0]$  (cut 3 of length 4m and 1 of length 5m).

#### 4.1.2 Column Generation

Consider the (??), but in this case we are instead minimizing. Thus we can write it as

$$\begin{aligned} \min \quad & (c_N - c_B B^{-1} N) x_N + c_B B^{-1} b \\ \text{s.t.} \quad & x_B + B^{-1} N x_N = B^{-1} b \\ & x \geq 0 \end{aligned} \tag{4.1.7}$$

In our LP we have  $c = \mathbb{1}$ , that is,  $c_i = 1$  for all  $i = 1, \dots, k$ . Hence, we can write it as

$$\begin{aligned} \min \quad & (\mathbb{1}_N - \mathbb{1}_B B^{-1} N) x_N + \mathbb{1}_B B^{-1} b \\ \text{s.t.} \quad & x_B + B^{-1} N x_N = B^{-1} b \\ & x \geq 0 \end{aligned} \tag{4.1.8}$$

Now, if there exists a non-basic variable that could enter the basis and improve the objective, then there is one with a reduced cost that is negative. For a particular non-basic variable, the coefficient on it is

$$\left(1 - \mathbb{1}_B B^{-1} N^i\right) x_i \tag{4.1.9}$$

where  $N^i$  is the  $i$ -th column of the matrix  $N$ . Thus, we want to look for a column  $a$  of  $N$  such that

$$1 - \mathbb{1}_B B^{-1} a < 0 \quad \Rightarrow \quad 1 < \mathbb{1}_B B^{-1} a \tag{4.1.10}$$

**Pricing Problem:**

(knapsack problem!)

Given a current basis  $B$  of the *master* linear program, there exists a new column to add to the basis that improves the LP objective if and only if the following problem has an objective value strictly larger than 1.

$$\begin{aligned}
 \max \quad & \mathbb{1}_B B^{-1}a \\
 \text{s.t.} \quad & \sum_{i=1}^k L_i a_i \leq L \\
 & a_i \in \mathbb{Z}_+ \text{ for } i = 1, \dots, k
 \end{aligned} \tag{4.1.11}$$

**Example 16:** Pricing Problem(knapsack problem!) After choosing the initial columns ....

we can find the objective function....

$$\begin{aligned}
 \max \quad & \dots\dots a \\
 \text{s.t.} \quad & 4a_1 + 5a_2 + 6a_3 \leq 19 \\
 & a_i \in \mathbb{Z}_+ \text{ for } i = 1, \dots, k
 \end{aligned} \tag{4.1.12}$$

**4.1.3 Cutting Stock - Multiple widths**

Gurobi has an excellent demonstration application to look at: [Gurobi - Cutting Stock Demo](#) [Gurobi - Multiple Master Rolls](#)

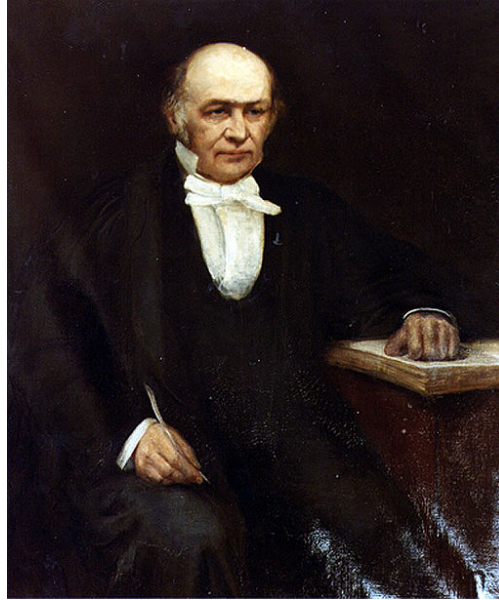
Here are some solutions:

- <https://github.com/fzsun/cutstock-gurobi>.
- <http://www.dcc.fc.up.pt/~jpp/mpa/cutstock.py>

Here is an AIMMS description of the problem: [AIMMS Cutting Stock](#)

**4.2 Spanning Trees**

See [5] for a list of 11 models for the minimum spanning tree and a comparison using CPLEX.



© See page for author [Public domain], via Wikimedia Commons<sup>1</sup>

**Figure 4.1:** Painting of Sir William Rowan Hamilton

### 4.3 Traveling Salesman Problem

See [math.uwaterloo.ca](http://math.uwaterloo.ca) for excellent material on the TSP.

See also this chapter [A Practical Guide to Discrete Optimization](#).

Also, watch this excellent talk by Bill Cook "Postcards from the Edge of Possibility": [Youtube!](#)

We consider a directed graph, graph  $G = (N, A)$  of nodes  $N$  and arcs  $A$ . Arcs are directed edges. Hence the arc  $(i, j)$  is the directed path  $i \rightarrow j$ .

A *tour*, or Hamiltonian cycle (see [Figure 4.1](#)), is a cycle that visits all the nodes in  $N$  exactly once and returns back to the starting node.

Given costs  $c_{ij}$  for each arc  $(i, j) \in A$ , the goal is to find a minimum cost tour.

#### **Traveling Salesman Problem:**

*NP-Hard*

Given a directed graph  $G = (N, A)$  and costs  $c_{ij}$  for all  $(i, j) \in A$ , find a tour of minimum cost.

#### **ADD TSP FIGURE**

In the figure, the nodes  $N$  are the cities and the arcs  $A$  are the directed paths  $\text{city}_i \rightarrow \text{city}_j$ .

<sup>1</sup> Painting of Sir William Rowan Hamilton, from [https://commons.wikimedia.org/wiki/File:William\\_Rowan\\_Hamilton\\_painting.jpg](https://commons.wikimedia.org/wiki/File:William_Rowan_Hamilton_painting.jpg). See page for author [Public domain], via Wikimedia Commons, mid 19th century.

**Models** When constructing an integer programming model for TSP, we define variables  $x_{ij}$  for all  $(i, j) \in A$  as

$x_{ij} = 1$  if the arc  $(i, j)$  is used and  $x_{ij} = 0$  otherwise.

We want the model to satisfy the fact that each node should have exactly one incoming arc and one leaving arc. Furthermore, we want to prevent self loops. Thus, we need the constraints:

$$\sum_{j \in N} x_{ij} = 1 \quad \text{for all } i \in N \quad [\text{outgoing arc}] \quad (4.3.1)$$

$$\sum_{i \in N} x_{ij} = 1 \quad \text{for all } j \in N \quad [\text{incoming arc}] \quad (4.3.2)$$

$$x_{ii} = 0 \quad \text{for all } i \in N \quad [\text{no self loops}] \quad (4.3.3)$$

Unfortunately, these constraints are not enough to completely describe the problem. The issue is that *subtours* may arise. For instance

#### ADD SUBTOURS FIGURE

#### MTZ Model

- This model adds variables  $u_i \in \mathbb{Z}$  with  $1 \leq u_i \leq n$  that decide the order in which nodes are visited.
- We set  $u_1 = 1$  to set a starting place.
- Crucially, this model relies on the following fact

*Let  $x$  be a solution to (4.3.1)-(4.3.3) with  $x_{ij} \in \{0, 1\}$ . If there exists a subtour in this solution that contains the node 1, then there also exists a subtour that does not contain the node 1.*

The following model adds constraints

$$\text{If } x_{ij} = 1, \text{ then } u_i + 1 \leq u_j. \quad (4.3.4)$$

This if-then statement can be modeled with a big-M, choosing  $M = n$  is a sufficient upper bound. Thus, it can be written as

$$u_i + 1 \leq u_j + n(1 - x_{ij}) \quad (4.3.5)$$

Setting these constraints to be active enforces the order  $u_i < u_j$ .

Consider a subtour now  $2 \rightarrow 5 \rightarrow 3 \rightarrow 2$ . Thus,  $x_{25} = x_{53} = x_{32} = 1$ . Then using the constraints from (4.3.5), we have that

$$u_2 < u_5 < u_3 < u_2, \quad (4.3.6)$$

but this is infeasible since we cannot have  $u_2 < u_2$ .

As stated above, if there is a subtour containing the node 1, then there is also a subtour not containing the node 1. Thus, we can enforce these constraints to only prevent subtours that don't contain the node 1. Thus, the full tour that contains the node 1 will still be feasible.

This is summarized in the following model:

**Traveling Salesman Problem - MTZ Model:**

$$\min \sum_{i,j \in N} c_{ij} x_{ij} \quad (4.3.7)$$

$$\sum_{j \in N} x_{ij} = 1 \quad \text{for all } i \in N \quad [\text{outgoing arc}] \quad (4.3.8)$$

$$\sum_{i \in N} x_{ij} = 1 \quad \text{for all } j \in N \quad [\text{incoming arc}] \quad (4.3.9)$$

$$x_{ii} = 0 \quad \text{for all } i \in N \quad [\text{no self loops}] \quad (4.3.10)$$

$$u_i + 1 \leq u_j + n(1 - x_{ij}) \quad \text{for all } i, j \in N, i, j \neq 1 \quad [\text{prevents subtours}] \quad (4.3.11)$$

$$u_1 = 1 \quad (4.3.12)$$

$$2 \leq u_i \leq n \quad \text{for all } i \in N, i \neq 1 \quad (4.3.13)$$

$$u_i \in \mathbb{Z} \quad \text{for all } i \in N \quad (4.3.14)$$

$$x_{ij} \in \{0, 1\} \quad \text{for all } i, j \in N \quad (4.3.15)$$

**Example 17:**

Distance Matrix:

A \ B	1	2	3	4
1	0	1	2	3
2	1	0	1	2
3	2	1	0	4
4	3	2	4	0

$$\begin{aligned}
& \min \quad x_{1,2} + 2x_{1,3} + 3x_{1,4} + x_{2,1} + x_{2,3} + 2x_{2,4} + \\
& 2x_{3,1} + x_{3,2} + 4x_{3,4} + 3x_{4,1} + 2x_{4,2} + 4x_{4,3} \\
& \text{Subject to} \\
& \quad x_{1,1} + x_{1,2} + x_{1,3} + x_{1,4} = 1 \quad \text{outgoing from node 1} \\
& \quad x_{2,1} + x_{2,2} + x_{2,3} + x_{2,4} = 1 \quad \text{outgoing from node 2} \\
& \quad x_{3,1} + x_{3,2} + x_{3,3} + x_{3,4} = 1 \quad \text{outgoing from node 3} \\
& \quad x_{4,1} + x_{4,2} + x_{4,3} + x_{4,4} = 1 \quad \text{outgoing from node 4} \\
& \quad x_{1,1} + x_{2,1} + x_{3,1} + x_{4,1} = 1 \quad \text{incoming to node 1} \\
& \quad x_{1,2} + x_{2,2} + x_{3,2} + x_{4,2} = 1 \quad \text{incoming to node 2} \\
& \quad x_{1,3} + x_{2,3} + x_{3,3} + x_{4,3} = 1 \quad \text{incoming to node 3} \\
& \quad x_{1,4} + x_{2,4} + x_{3,4} + x_{4,4} = 1 \quad \text{incoming to node 4} \\
& \quad x_{1,1} = 0 \quad \text{No self loop with node 1} \\
& \quad x_{2,2} = 0 \quad \text{No self loop with node 2} \\
& \quad x_{3,3} = 0 \quad \text{No self loop with node 3} \\
& \quad x_{4,4} = 0 \quad \text{No self loop with node 4} \\
& \quad u_1 = 1 \quad \text{Start at node 1} \\
& \quad 2 \leq u_i \leq 4, \quad \forall i \in \{2, 3, 4\} \\
& \quad u_2 + 1 \leq u_3 + 4(1 - x_{2,3}) \\
& \quad u_2 + 1 \leq u_4 + 4(1 - x_{2,4}) \leq 3 \\
& \quad u_3 + 1 \leq u_2 + 4(1 - x_{3,2}) \leq 3 \\
& \quad u_3 + 1 \leq u_4 + 4(1 - x_{3,4}) \leq 3 \\
& \quad u_4 + 1 \leq u_2 + 4(1 - x_{4,2}) \leq 3 \\
& \quad u_4 + 1 \leq u_3 + 4(1 - x_{4,3}) \leq 3 \\
& \quad x_{i,j} \in \{0, 1\} \quad \forall i \in \{1, 2, 3, 4\}, j \in \{1, 2, 3, 4\} \\
& \quad u_i \in \mathbb{Z}, \quad \forall i \in \{1, 2, 3, 4\}
\end{aligned}$$

**Example 18:** 5 nodes

$$\begin{aligned}
\min \quad & x_{1,2} + 2x_{1,3} + 3x_{1,4} + 4x_{1,5} + x_{2,1} + x_{2,3} + 2x_{2,4} + 2x_{2,5} + 2x_{3,1} + \\
& x_{3,2} + 4x_{3,4} + x_{3,5} + 3x_{4,1} + 2x_{4,2} + 4x_{4,3} + 2x_{4,5} + \\
& 4x_{5,1} + 2x_{5,2} + x_{5,3} + 2x_{5,4} \\
\text{Subject to} \quad & x_{1,1} + x_{1,2} + x_{1,3} + x_{1,4} + x_{1,5} = 1 \\
& x_{2,1} + x_{2,2} + x_{2,3} + x_{2,4} + x_{2,5} = 1 \\
& x_{3,1} + x_{3,2} + x_{3,3} + x_{3,4} + x_{3,5} = 1 \\
& x_{4,1} + x_{4,2} + x_{4,3} + x_{4,4} + x_{4,5} = 1 \\
& x_{5,1} + x_{5,2} + x_{5,3} + x_{5,4} + x_{5,5} = 1 \\
\\
& x_{1,1} + x_{2,1} + x_{3,1} + x_{4,1} + x_{5,1} = 1 \\
& x_{1,2} + x_{2,2} + x_{3,2} + x_{4,2} + x_{5,2} = 1 \\
& x_{1,3} + x_{2,3} + x_{3,3} + x_{4,3} + x_{5,3} = 1 \\
& x_{1,4} + x_{2,4} + x_{3,4} + x_{4,4} + x_{5,4} = 1 \\
& x_{1,5} + x_{2,5} + x_{3,5} + x_{4,5} + x_{5,5} = 1 \\
\\
& x_{1,1} = 0 \\
& x_{2,2} = 0 \\
& x_{3,3} = 0 \\
& x_{4,4} = 0 \\
& x_{5,5} = 0 \\
\\
& u_1 = 1 \\
& 2 \leq u_i \leq 5 \quad \forall i \in \{1, 2, 3, 4, 5\} \\
& u_2 + 1 \leq u_3 + 5(1 - x_{2,3}) \\
& u_2 + 1 \leq u_4 + 5(1 - x_{2,4}) \\
& u_2 + 1 \leq u_5 + 5(1 - x_{2,5}) \\
& u_3 + 1 \leq u_2 + 5(1 - x_{3,2}) \\
& u_3 + 1 \leq u_4 + 5(1 - x_{3,4}) \\
& u_4 + 1 \leq u_2 + 5(1 - x_{4,2}) \\
& u_4 + 1 \leq u_3 + 5(1 - x_{4,3}) \\
& u_3 + 1 \leq u_5 + 5(1 - x_{3,5}) \\
& u_4 + 1 \leq u_5 + 5(1 - x_{4,5}) \\
& u_5 + 1 \leq u_2 + 5(1 - x_{5,2}) \\
& u_5 + 1 \leq u_3 + 5(1 - x_{5,3}) \\
& u_5 + 1 \leq u_4 + 5(1 - x_{5,4}) \\
& x_{i,j} \in \{0, 1\} \quad \forall i \in \{1, 2, 3, 4, 5\}, j \in \{1, 2, 3, 4, 5\} \\
& u_i \in \mathbb{Z}, \quad \forall i \in \{1, 2, 3, 4, 5\}
\end{aligned}$$



**Pros of this model**

- Small description
- Easy to implement

**Cons of this model**

- Linear relaxation is not very tight. Thus, the solver may be slow when given this model.

**4.3.1 Dantzig-Fulkerson-Johnson Model**

This model does not add new variables. Instead, it adds constraints that conflict with the subtours. For instance, consider a subtour

$$2 \rightarrow 5 \rightarrow 3 \rightarrow 2. \quad (4.3.16)$$

We can prevent this subtour by adding the constraint

$$x_{25} + x_{53} + x_{32} \leq 2 \quad (4.3.17)$$

meaning that at most 2 of those arcs are allowed to happen at the same time. In general, for any subtour  $S$ , we can have the *subtour elimination constraint*

$$\sum_{(i,j) \in S} x_{ij} \leq |S| - 1 \quad \text{Subtour Elimination Constraint.} \quad (4.3.18)$$

In the previous example with  $S = \{(2,5), (5,3), (3,2)\}$  we have  $|S| = 3$ , where  $|S|$  denotes the size of the set  $S$ .

This model suggests that we just add all of these subtour elimination constraints.

**Traveling Salesman Problem - DFJ Model:**

$$\min \sum_{i,j \in N} c_{ij} x_{ij} \quad (4.3.19)$$

$$\sum_{j \in N} x_{ij} = 1 \quad \text{for all } i \in N \quad [\text{outgoing arc}] \quad (4.3.20)$$

$$\sum_{i \in N} x_{ij} = 1 \quad \text{for all } j \in N \quad [\text{incoming arc}] \quad (4.3.21)$$

$$x_{ii} = 0 \quad \text{for all } i \in N \quad [\text{no self loops}] \quad (4.3.22)$$

$$\sum_{(i,j) \in S} x_{ij} \leq |S| - 1 \quad \text{for all subtours } S \quad [\text{prevents subtours}] \quad (4.3.23)$$

$$x_{ij} \in \{0,1\} \quad \text{for all } i,j \in N \quad (4.3.24)$$

**Example 19:** DFJ Model for  $n = 4$  nodes

Distance Matrix:

A \ B	1	2	3	4
1	0	1	2	3
2	1	0	1	2
3	2	1	0	4
4	3	2	4	0

$$\begin{aligned} 1 \text{ min } & x_{1,2} + 2x_{1,3} + 3x_{1,4} + x_{2,1} + x_{2,3} + 2x_{2,4} \\ & + 2x_{3,1} + x_{3,2} + 4x_{3,4} + 3x_{4,1} + 2x_{4,2} + 4x_{4,3} \end{aligned}$$

Subject to

$$\begin{aligned} x_{1,1} + x_{1,2} + x_{1,3} + x_{1,4} &= 1 && \text{outgoing from node 1} \\ x_{2,1} + x_{2,2} + x_{2,3} + x_{2,4} &= 1 && \text{outgoing from node 2} \\ x_{3,1} + x_{3,2} + x_{3,3} + x_{3,4} &= 1 && \text{outgoing from node 3} \\ x_{4,1} + x_{4,2} + x_{4,3} + x_{4,4} &= 1 && \text{outgoing from node 4} \end{aligned}$$

$$\begin{aligned} x_{1,1} + x_{2,1} + x_{3,1} + x_{4,1} &= 1 && \text{incoming to node 1} \\ x_{1,2} + x_{2,2} + x_{3,2} + x_{4,2} &= 1 && \text{incoming to node 2} \\ x_{1,3} + x_{2,3} + x_{3,3} + x_{4,3} &= 1 && \text{incoming to node 3} \\ x_{1,4} + x_{2,4} + x_{3,4} + x_{4,4} &= 1 && \text{incoming to node 4} \end{aligned}$$

$$\begin{aligned} x_{1,1} &= 0 && \text{No self loop with node 1} \\ x_{2,2} &= 0 && \text{No self loop with node 2} \\ x_{3,3} &= 0 && \text{No self loop with node 3} \\ x_{4,4} &= 0 && \text{No self loop with node 4} \end{aligned}$$

$$\begin{aligned} x_{1,2} + x_{2,1} &\leq 1 && S = [(1,2), (2,1)] \\ x_{1,3} + x_{3,1} &\leq 1 && S = [(1,3), (3,1)] \\ x_{1,4} + x_{4,1} &\leq 1 && S = [(1,4), (4,1)] \\ x_{2,3} + x_{3,2} &\leq 1 && S = [(2,3), (3,2)] \\ x_{2,4} + x_{4,2} &\leq 1 && S = [(2,4), (4,2)] \\ x_{3,4} + x_{4,3} &\leq 1 && S = [(3,4), (4,3)] \\ x_{2,1} + x_{1,3} + x_{3,2} &\leq 2 && S = [(2,1), (1,3), (3,2)] \\ x_{1,2} + x_{2,3} + x_{3,1} &\leq 2 && S = [(1,2), (2,3), (3,1)] \\ x_{3,1} + x_{1,4} + x_{4,3} &\leq 2 && S = [(3,1), (1,4), (4,3)] \\ x_{1,3} + x_{3,4} + x_{4,1} &\leq 2 && S = [(1,3), (3,4), (4,1)] \\ x_{2,1} + x_{1,4} + x_{4,2} &\leq 2 && S = [(2,1), (1,4), (4,2)] \\ x_{1,2} + x_{2,4} + x_{4,1} &\leq 2 && S = [(1,2), (2,4), (4,1)] \\ x_{3,2} + x_{2,4} + x_{4,3} &\leq 2 && S = [(3,2), (2,4), (4,3)] \\ x_{2,3} + x_{3,4} + x_{4,2} &\leq 2 && S = [(2,3), (3,4), (4,2)] \end{aligned}$$

$$x_{i,j} \in \{0,1\} \quad \forall i \in \{1,2,3,4\}, j \in \{1,2,3,4\}$$

**Example 20:**

Consider a graph on 5 nodes.

Here are all the subtours of length at least 3 and also including the full length tours.

Hence, there are many subtours to consider.

**Pros of this model**

- Very tight linear relaxation

**Cons of this model**

- Exponentially many subtours  $S$  possible, hence this model is too large to write down.

**Solution: Add subtour elimination constraints as needed. We will discuss this in a future section on *cutting planes* .**

**4.3.2 Traveling Salesman Problem - Branching Solution**

We will see in the next section

1. That the constraint (4.3.1)-(4.3.3) always produce integer solutions as solutions to the linear relaxation.
2. A way to use branch and bound (the topic of the next section) in order to avoid subtours.

**4.4 Literature and other notes**

- Gilmore-Gomory Cutting Stock [13]
- [A Column Generation Algorithm for Vehicle Scheduling and Routing Problems](#)
- [The Integrated Last-Mile Transportation Problem](#)
- [http://www.optimization-online.org/DB\\_FILE/2017/11/6331.pdf](http://www.optimization-online.org/DB_FILE/2017/11/6331.pdf) A BRANCH-AND-PRICE ALGORITHM FOR CAPACITATED HYPERGRAPH VERTEX SEPARATION

**4.4.1 Google maps data**

[Blog - Python | Calculate distance and duration between two places using google distance matrix API](#)

## Chapter 5

# Algorithms to Solve Integer Programs

### 5.1 LP to solve IP

Recall that the linear relaxation of an integer program is the linear programming problem after removing the integrality constraints

Integer Program:

$$\begin{aligned} \max \quad & z_{IP} = c^\top x \\ & Ax \leq b \\ & x \in \mathbb{Z}^n \end{aligned}$$

Linear Relaxation:

$$\begin{aligned} \max \quad & z_{LP} = c^\top x \\ & Ax \leq b \\ & x \in \mathbb{R}^n \end{aligned}$$

**Theorem 14** (LP Bounds). *It always holds that*

$$z_{IP}^* \leq z_{LP}^*. \quad (5.1.1)$$

Furthermore, if  $x_{LP}^*$  is integral (feasible for the integer program), then

$$x_{LP}^* = x_{IP}^* \quad \text{and} \quad z_{LP}^* = z_{IP}^*. \quad (5.1.2)$$

#### Example 21:

Consider the problem

$$\begin{aligned} \max z = & 3x_1 + 2x_2 \\ & 2x_1 + x_2 \leq 6 \\ & x_1, x_2 \geq 0; x_1, x_2 \text{ integer} \end{aligned}$$

#### 5.1.1 Rounding LP Solution can be bad!

Video! - Michel Belaire (EPFL) looking at rounding the LP solution to an IP solution

Consider the two variable knapsack problem

$$\max 3x_1 + 100x_2 \quad (5.1.3)$$

$$x_1 + 100x_2 \leq 100 \quad (5.1.4)$$

$$x_i \in \{0, 1\} \text{ for } i = 1, 2. \quad (5.1.5)$$

Then  $x_{LP}^* = (1, 0.99)$  and  $z_{LP}^* = 1 \cdot 3 + 0.99 \cdot 100 = 3 + 99 = 102$ .

But  $x_{IP}^* = (0, 1)$  with  $z_{IP}^* = 0 \cdot 3 + 1 \cdot 100 = 100$ .

Suppose that we rounded the LP solution.

$x_{LP-Rounded-Down}^* = (1, 0)$ . Then  $z_{LP-Rounded-Down}^* = 1 \cdot 3 = 3$ . Which is a terrible solution!

How can we avoid this issue?

Cool trick! Using two different strategies gives you at least a  $1/2$  approximation to the optimal solution.

### 5.1.2 Rounding LP solution can be infeasible!

Now only could it produce a poor solution, it is not always clear how to round to a feasible solution.

### 5.1.3 Fractional Knapsack

The fractional knapsack problem has an exact greedy algorithm.

- [Youtube!](#)
- [Blog](#)

## 5.2 Branch and Bound

Video! - Michel Belaire (EPFL) Teaching Branch and Bound Theory [Video! - Michel Belaire \(EPFL\) Teaching Branch and Bound with Example](#)

See [Module by Miguel Casquilho](#) for some nice notes on branch and bound.

### 5.2.1 Algorithm

---

**Algorithm 2** Branch and Bound - Maximization
 

---

**Input:** Integer Linear Problem with max objective

**Output:** Exact Optimal Solution  $x^*$ 

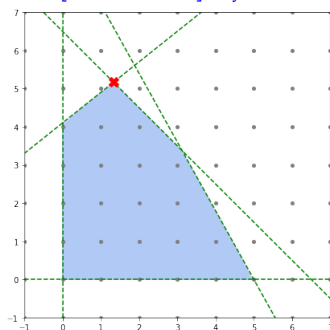
- 1: Set  $LB = -\infty$ .
  - 2: Solve LP relaxation.
    - a: If  $x^*$  is integer, stop!
    - b: Otherwise, choose fractional entry  $x_i^*$  and branch onto subproblems:
      - (i)  $x_i \leq \lfloor x_i^* \rfloor$  and (ii)  $x_i \geq \lceil x_i^* \rceil$ .
  - 3: Solve LP relaxation of any subproblem.
    - a: If LP relaxation is infeasible, prune this node as "Infeasible"
    - b: If  $z^* < LB$ , prune this node as "Suboptimal"
    - c:  $x^*$  is integer, prune this nodes as "Integer" and update  $LB = \max(LB, z^*)$ .
    - d: Otherwise, choose fractional entry  $x_i^*$  and branch onto subproblems:
      - (i)  $x_i \leq \lfloor x_i^* \rfloor$  and (ii)  $x_i \geq \lceil x_i^* \rceil$ . Return to step 2 until all subproblems are pruned.
  - 4: Return best integer solution found.
- 

Here is an example of branching on general integer variables.

**Example 22:** Consider the two variable example with

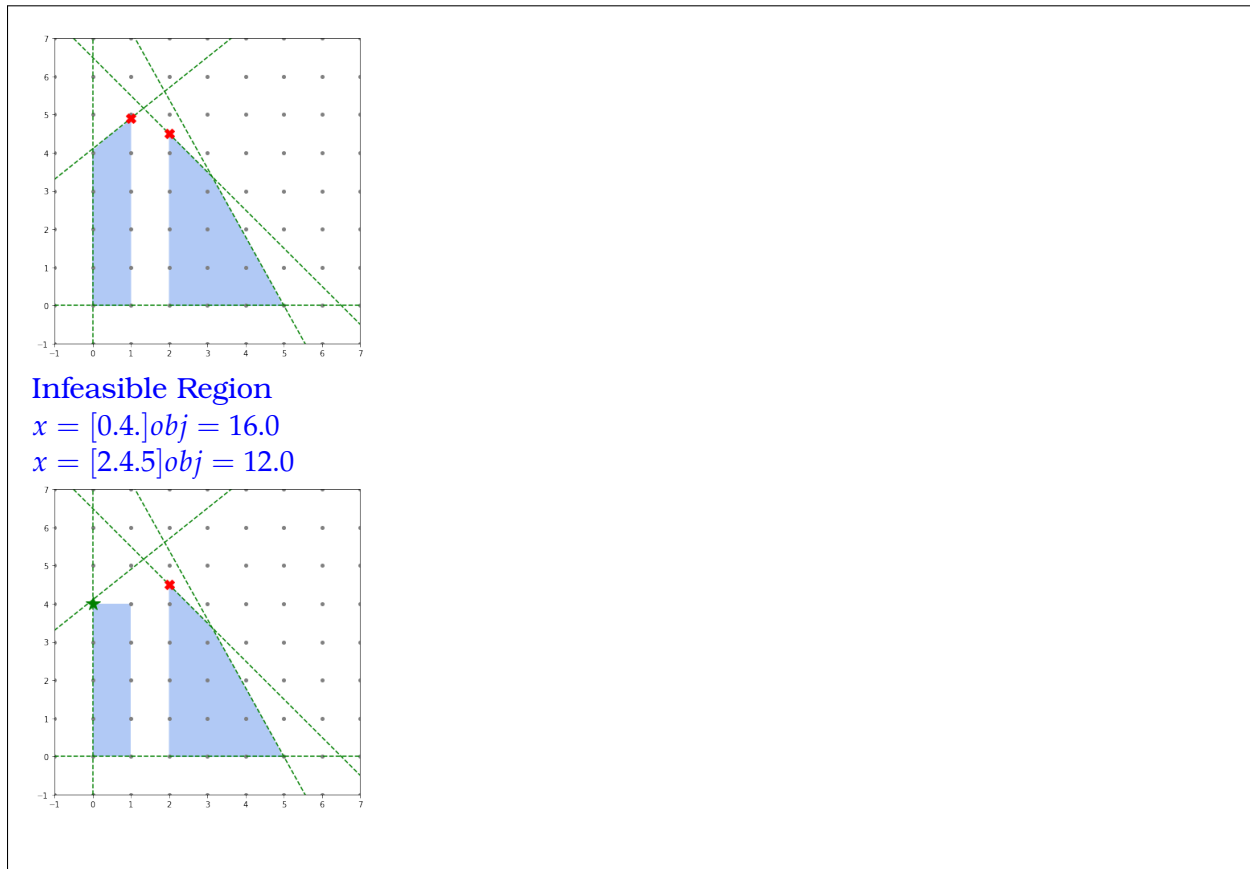
$$\begin{aligned}
 &\max -3x_1 + 4x_2 \\
 &2x_1 + 2x_2 \leq 13 \\
 &-8x_1 + 10x_2 \leq 41 \\
 &9x_1 + 5x_2 \leq 45 \\
 &0 \leq x_1 \leq 10, \text{ integer} \\
 &0 \leq x_2 \leq 10, \text{ integer}
 \end{aligned}$$

$$x = [1.33, 5.167] \text{obj} = 16.664$$



$$x = [1, 4.9] \text{obj} = 16.5998$$

$$x = [2, 4.5] \text{obj} = 12.0$$



### 5.2.2 Knapsack Problem and 0/1 branching

Consider the problem

$$\begin{aligned}
 \max \quad & 16x_1 + 22x_2 + 12x_3 + 8x_4 \\
 \text{s.t.} \quad & 5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14 \\
 & 0 \leq x_i \leq 1 \quad i = 1, 2, 3, 4 \\
 & x_i \in \{0, 1\} \quad i = 1, 2, 3, 4
 \end{aligned}$$

**Question: What is the optimal solution if we remove the binary constraints?**

$$\begin{aligned}
 \max \quad & c_1x_1 + c_2x_2 + c_3x_3 + c_4x_4 \\
 \text{s.t.} \quad & a_1x_1 + a_2x_2 + a_3x_3 + a_4x_4 \leq b \\
 & 0 \leq x_i \leq 1 \quad i = 1, 2, 3, 4
 \end{aligned}$$

**Question: How do I find the solution to this problem?**



$$\begin{aligned}
 \max \quad & c_1x_1 + c_2x_2 + c_3x_3 + c_4x_4 \\
 \text{s.t.} \quad & (a_1 - A)x_1 + (a_2 - A)x_2 + (a_3 - A)x_3 + (a_4 - A)x_4 \leq 0 \\
 & 0 \leq x_i \leq m_i \quad i = 1, 2, 3, 4
 \end{aligned}$$

**Question: How do I find the solution to this problem?**

Consider the problem

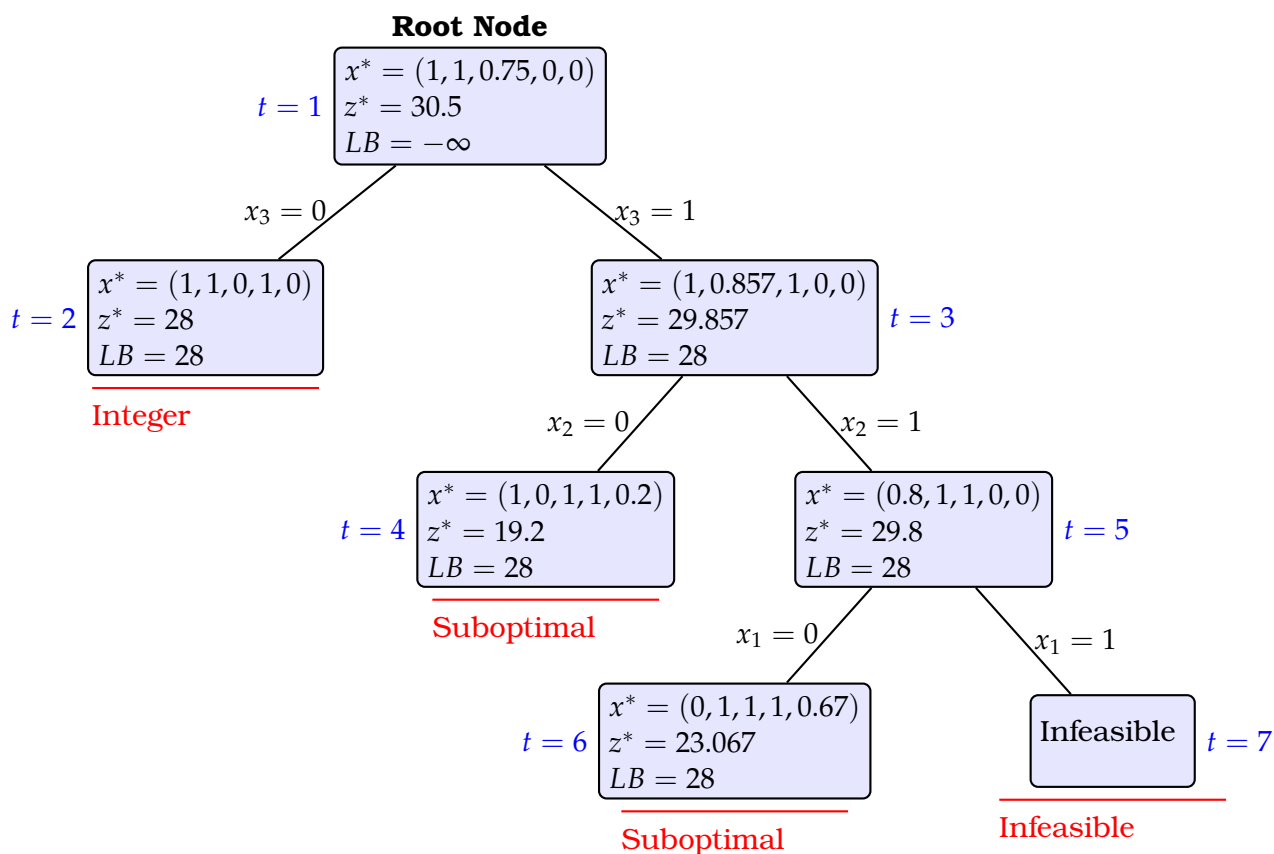
$$\begin{aligned}
 \max \quad & 16x_1 + 22x_2 + 12x_3 + 8x_4 \\
 \text{s.t.} \quad & 5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14 \\
 & 0 \leq x_i \leq 1 \quad i = 1, 2, 3, 4 \\
 & x_i \in \{0, 1\} \quad i = 1, 2, 3, 4
 \end{aligned}$$

We can solve this problem with branch and bound.

The optimal solution was found at  $t = 5$  at subproblem 6 to be  $x^* = (0, 1, 1, 1)$ ,  $z^* = 42$ .

**Example: Binary Knapsack** Solve the following problem with branch and bound.

$$\begin{aligned}
 \max \quad & z = 11x_1 + 15x_2 + 6x_3 + 2x_4 + x_5 \\
 \text{Subject to:} \quad & 5x_1 + 7x_2 + 4x_3 + 3x_4 + 15x_5 \leq 15 \\
 & x_i \text{ binary}, i = 1, \dots, 4
 \end{aligned}$$



### 5.2.3 Traveling Salesman Problem solution via Branching

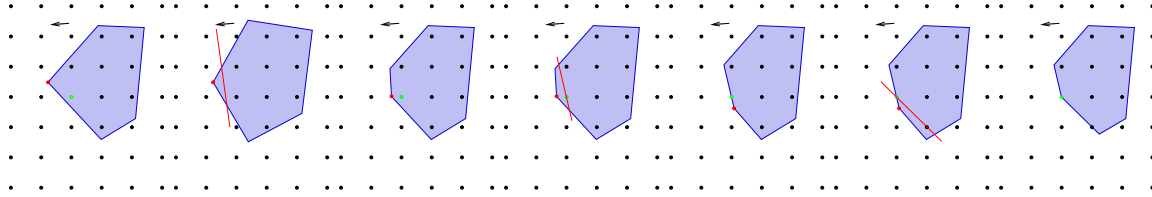
**To Do #10:** Describe solving TSP via a generalized branching method that removes subtours (instead of adding constraints).

## 5.3 Cutting Planes

Cutting planes are inequalities  $\pi^\top x \leq \pi_0$  that are valid for the feasible integer solutions that the cut off part of the LP relaxation. Cutting planes can create a tighter description of the feasible region that allows for the optimal solution to be obtained by simply solving a strengthened linear relaxation.

The cutting plane procedure, as demonstrated in Figure ??, The procedure is as follows:

1. Solve the current LP relaxation.
2. If solution is integral, then return that solution. STOP
3. Add a cutting plane (or many cutting planes) that cut off the LP-optimal solution.
4. Return to Step 1.



**Figure 5.1:** The cutting plane procedure.

In practice, this procedure is integrated in some with with branch and bound and also other primal heuristics.

### 5.3.1 Chvátal Cuts

Chvátal Cuts are a general technique to produce new inequalities that are valid for feasible integer points.

**Chvátal Cuts:**

Suppose

$$a_1x_1 + \cdots + a_nx_n \leq d \quad (5.3.1)$$

is a valid inequality for the polyhedron  $P = \{x \in \mathbb{R}^n : Ax \leq b, x \geq 0\}$ , then

$$\lfloor a_1 \rfloor x_1 + \cdots + \lfloor a_n \rfloor x_n \leq \lfloor d \rfloor \quad (5.3.2)$$

is valid for the integer points in  $P$ , that is, it is valid for the set  $P \cap \mathbb{Z}^n$ . Equation (??) is called a Chvátal Cut.

We will illustrate this idea with an example.

**Example 23:** Recall example 7. The model was  
**Model**

$$\begin{array}{ll} \min & p + n + d + q & \text{total number of coins used} \\ \text{s.t.} & p + 5n + 10d + 25q = 83 & \text{sums to 83¢} \\ & p, d, n, q \in \mathbb{Z}_+ & \text{each is a non-negative integer} \end{array}$$

From the equality constraint we can derive several inequalities.

1. Divide by 25 and round down both sides:

$$\frac{p + 5n + 10d + 25q}{25} = 83/25 \Rightarrow q \leq 3$$

2. Divide by 10 and round down both sides:

$$\frac{p + 5n + 10d + 25q}{10} = 83/10 \Rightarrow d + 2q \leq 8$$

3. Divide by 5 and round down both sides:

$$\frac{p + 5n + 10d + 25q}{10} = 83/5 \Rightarrow n + 2d + 5q \leq 16$$

4. Multiply by 0.12 and round down both sides:

$$0.12(p + 5n + 10d + 25q) = 0.12(83) \Rightarrow d + 3q \leq 9$$

These new inequalities are all valid for the integer solutions. Consider the new model:

**New Model**

$$\begin{array}{ll} \min & p + n + d + q & \text{total number of coins used} \\ \text{s.t.} & p + 5n + 10d + 25q = 83 & \text{sums to 83¢} \\ & q \leq 3 & \\ & d + 2q \leq 8 & \\ & n + 2d + 5q \leq 16 & \\ & d + 3q \leq 9 & \\ & p, d, n, q \in \mathbb{Z}_+ & \text{each is a non-negative integer} \end{array}$$

The solution to the LP relaxation is exactly  $q = 3, d = 0, n = 1, p = 3$ , which is an integral feasible solution, and hence it is an optimal solution.

### 5.3.2 Gomory Cuts

[Michel Bierlaire \(EPFL\) Teaching Gomory Cuts](#) Gomory cuts are a type of Chvátal cut that is derived from the simplex tableau. Specifically, suppose that

$$x_i + \sum_{i \in N} \tilde{a}_i x_i = \tilde{b}_i \quad (5.3.3)$$

is an equation in the optimal simplex tableau.

**Gomory Cut:**

The Gomory cut corresponding to the tableau row (??) is

$$\sum_{i \in N} (\tilde{a}_i - \lfloor \tilde{a}_i \rfloor) x_i \geq \tilde{b}_i - \lfloor \tilde{b}_i \rfloor \quad (5.3.4)$$

We will solve the following problem using only Gomory Cuts.

$$\begin{aligned} \min \quad & x_1 - 2x_2 \\ \text{s.t.} \quad & -4x_1 + 6x_2 \leq 9 \\ & x_1 + x_2 \leq 4 \\ & x \geq 0, \quad x_1, x_2 \in \mathbb{Z} \end{aligned}$$

**Step 1:** The first thing to do is to put this into standard form by appending slack variables.

$$\begin{aligned} \min \quad & x_1 - 2x_2 \\ \text{s.t.} \quad & -4x_1 + 6x_2 + s_1 = 9 \\ & x_1 + x_2 + s_2 = 4 \\ & x \geq 0, \quad x_1, x_2 \in \mathbb{Z} \end{aligned} \quad (5.3.5)$$

We can apply the simplex method to solve the LP relaxation.

	Basis	RHS	$x_1$	$x_2$	$s_1$	$s_2$
Initial Basis	z	0.0	1.0	-2.0	0.0	0.0
	$s_1$	9.0	-4.0	6.0	1.0	0.0
	$s_2$	4.0	1.0	1.0	0.0	1.0
	$\vdots$					
Optimal Basis	Basis	RHS	$x_1$	$x_2$	$s_1$	$s_2$
	z	-3.5	0.0	0.0	0.3	0.2
	$x_1$	1.5	1.0	0.0	-0.1	0.6
	$x_2$	2.5	0.0	1.0	0.1	0.4

This LP relaxation produces the fractional basic solution  $x_{LP} = (1.5, 2.5)$ .

**Example 24: (Gomory cut removes LP solution)**

We now identify an integer variable  $x_i$  that has a fractional basic solution. Since both variables have fractional values, we can choose either row to make a cut. Let's focus on the row corresponding to  $x_1$ .

The row from the tableau expresses the equation

$$x_1 - 0.1s_1 + -0.6s_2 = 1.5. \quad (5.3.6)$$

Applying the Gomory Cut (??), we have the inequality

$$0.9s_1 + 0.4s_2 \geq 0.5. \quad (5.3.7)$$

The current LP solution is  $(x_{LP}, s_{LP}) = (1.5, 2.5, 0, 0)$ . Trivially, since  $s_1, s_2 = 0$ , the inequality is violated.

#### Example 25: (Gomory Cut in Original Space)

The Gomory Cut (??) can be rewritten in the original variables using the equations from (??). That is, we can use the equations

$$\begin{aligned} s_1 &= 9 + 4x_1 - 6x_2 \\ s_2 &= 4 - x_1 - x_2, \end{aligned} \quad (5.3.8)$$

which transforms the Gomory cut into the original variables to create the inequality

$$0.9(9 + 4x_1 - 6x_2) + 0.4(4 - x_1 - x_2) \geq 0.5.$$

or equivalently

$$-3.2x_1 + 5.8x_2 \leq 9.2. \quad (5.3.9)$$

As you can see, this inequality does cut off the current LP relaxation.

**Example 26: (Gomory cuts plus new tableau)** Now we add the slack variable  $s_3 \geq 0$  to make the equation

$$0.9s_1 + 0.4s_2 - s_3 = 0.5. \quad (5.3.10)$$

Next, we need to solve the linear programming relaxation (where we assume the variables are continuous).

## 5.4 Branching Rules

There is a few clever ideas out there on how to choose which variables to branch on. We will not go into this here. But for the interested reader, look into

- Strong Branching
- Pseudo-cost Branching

## 5.5 Lagrangian Relaxation for Branch and Bound

At each node in the branch and bound tree, we want to bound the objective value. One way to get a good bound can be using the Lagrangian.

See [\[12\]](#) [\(link\)](#) for a description of this.

## 5.6 Benders Decomposition

[Benders Decomposition - Julia Opt](#)

## 5.7 Literature

Model	LP Solution
$\begin{array}{ll} \max & x_1 + x_2 \\ \text{subject to} & -2x_1 + x_2 \leq 0.5 \\ & x_1 + 2x_2 \leq 10.5 \\ & x_1 - x_2 \leq 0.5 \\ & -2x_1 - x_2 \leq -2 \end{array}$	
$\begin{array}{ll} \max & x_1 + x_2 \\ \text{subject to} & -2x_1 + x_2 \leq 0.5 \\ & x_1 + 2x_2 \leq 10.5 \\ & x_1 - x_2 \leq 0.5 \\ & -2x_1 - x_2 \leq -2 \\ & x_1 \leq 3 \end{array}$	
$\begin{array}{ll} \max & x_1 + x_2 \\ \text{subject to} & -2x_1 + x_2 \leq 0.5 \\ & x_1 + 2x_2 \leq 10.5 \\ & x_1 - x_2 \leq 0.5 \\ & -2x_1 - x_2 \leq -2 \\ & x_1 \leq 3 \\ & x_1 + x_2 \leq 6 \end{array}$	





## Chapter 6

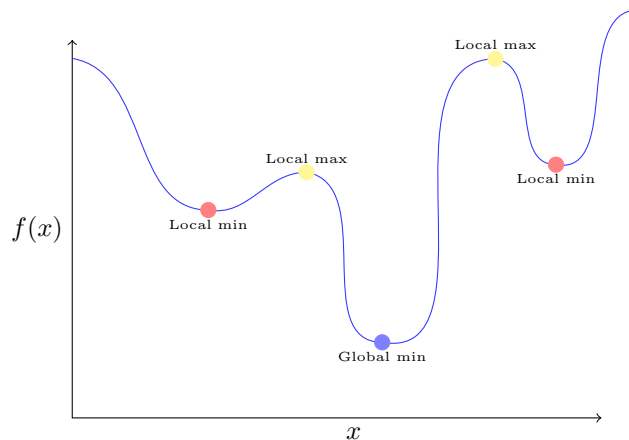
# Non-linear Programming (NLP)

$\min_{x \in \mathbb{R}^n} f(x)$	$\min_{x \in \mathbb{R}^n} f(x)$ $f_i(x) \leq 0 \quad \text{for } i = 1, \dots, m$
Unconstrained Minimization	Constrained Minimization

- **objective function**  $f: \mathbb{R}^n \rightarrow \mathbb{R}$
- may **maximize**  $f$  by minimizing the function  $g(x) := -f(x)$

**Definition 15** (Global and local optima). *The vector  $x^*$  is a*

- global minimizer *if  $f(x^*) \leq f(x)$  for all  $x \in \mathbb{R}^n$ .*
- local minimizer *if  $f(x^*) \leq f(x)$  for all  $x$  satisfying  $\|x - x^*\| \leq \epsilon$  for some  $\epsilon > 0$ .*
- strict local minimizer *if  $f(x^*) < f(x)$  for all  $x \neq x^*$  satisfying  $\|x - x^*\| \leq \epsilon$  for some  $\epsilon > 0$ .*



**Theorem 16.** *Let  $S$  be a nonempty set that is closed and bounded. Suppose that  $f: S \rightarrow \mathbb{R}$  is continuous. Then the problem  $\min\{f(x) : x \in S\}$  attains its minimum.*

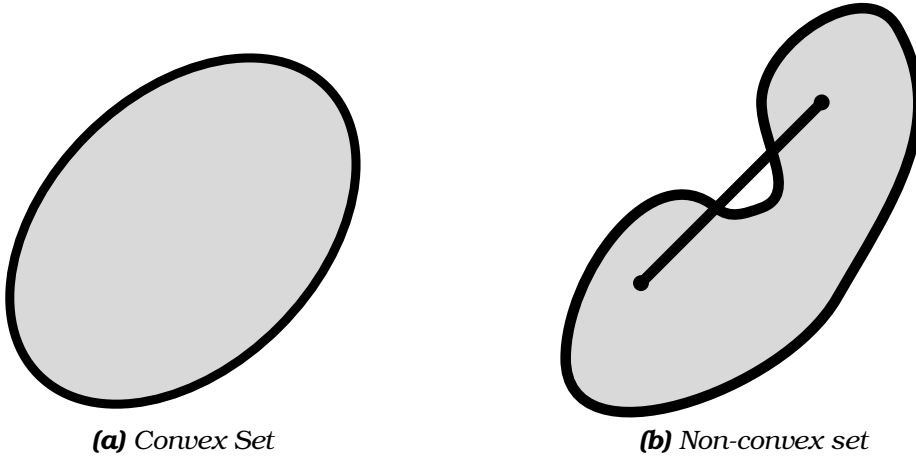
**Definition 17.** A critical point is a point  $\bar{x}$  where  $\nabla f(\bar{x}) = 0$ .

**Theorem 18.** *Suppose that  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is differentiable. If  $\min\{f(x) : x \in \mathbb{R}^n\}$  has an optimizer  $x^*$ , then  $x^*$  is a critical point of  $f$  (i.e.,  $\nabla f(x^*) = 0$ ).*

## 6.1 Convex Sets

**Definition 19** (Convex Combination). Given two points  $x, y$ , a convex combination is any point  $z$  that lies on the line between  $x$  and  $y$ . Algebraically, a convex combination is any point  $z$  that can be represented as  $z = \lambda x + (1 - \lambda)y$  for some multiplier  $\lambda \in [0, 1]$ .

**Definition 20** (Convex Set). A set  $C$  is convex if it contains all convex combinations of points in  $C$ . That is, for any  $x, y \in C$ , it holds that  $\lambda x + (1 - \lambda)y \in C$  for all  $\lambda \in [0, 1]$ .



**Figure 6.1:** Examples of convex and non-convex sets.

**Definition 21** (Convex Sets). A set  $S$  is convex if for any two points in  $S$ , the entire line segment between them is also contained in  $S$ . That is, for any  $x, y \in S$

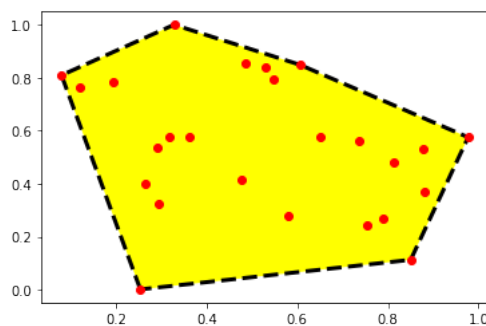
$$\lambda x + (1 - \lambda)y \in S \quad \text{for all } \lambda \in [0, 1].$$

Examples Convex Sets

1. Hyperplane  $H = \{x \in \mathbb{R}^n : a^\top x = b\}$
2. Halfspace  $H = \{x \in \mathbb{R}^n : a^\top x \leq b\}$
3. Polyhedron  $P = \{x \in \mathbb{R}^n : Ax \leq b\}$
4. Second Order Cone  $S = \{(x, t) \in \mathbb{R}^n \times \mathbb{R} : \sum_{i=1}^n x_i^2 \leq t^2\}$

**Definition 22** (Convex Hull). Let  $S \subseteq \mathbb{R}^n$ . The convex hull  $\text{conv}(S)$  is the smallest convex set containing  $S$ .

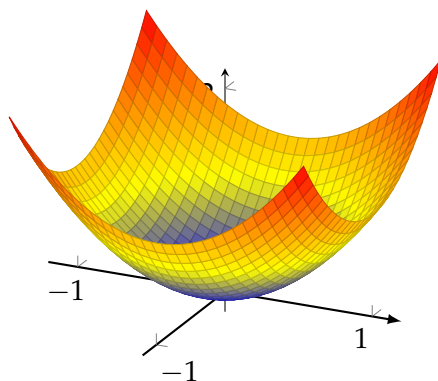
t



**Theorem 23** (Caratheodory's Theorem). *Let  $x \in \text{conv}(S)$  and  $S \subseteq \mathbb{R}^n$ . Then there exist  $x^1, \dots, x^k \in S$  such that  $x \in \text{conv}(\{x^1, \dots, x^k\})$  and  $k \leq n + 1$ .*

## 6.2 Convex Functions

Convex function are "nice" functions that "open up". They represent an extremely important class of functions in optimization and typically can be optimized over efficiently.



**Figure 6.2:** Convex Function  $f(x, y) = x^2 + y^2$ .

**Definition 24** (Convex Functions). *A function  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is convex if for all  $x, y \in \mathbb{R}^n$  and  $\lambda \in [0, 1]$  we have*

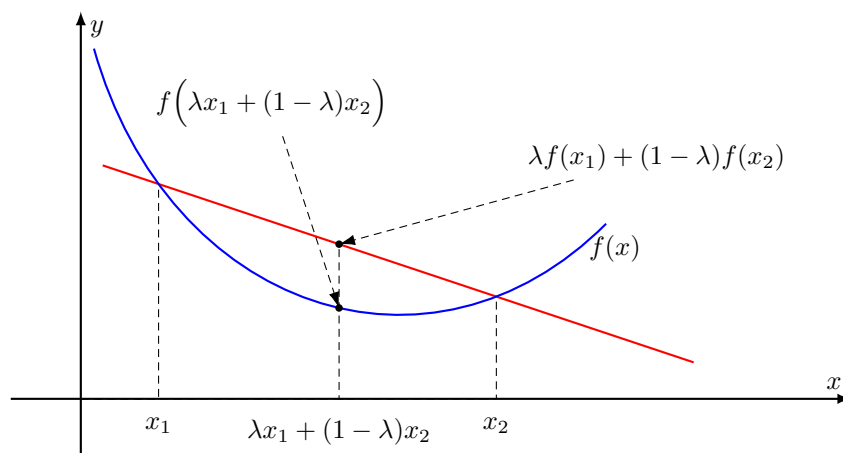
$$\lambda f(x) + (1 - \lambda)f(y) \geq f(\lambda x + (1 - \lambda)y). \quad (6.2.1)$$

An equivalent definition of convex function are through the epigraph.

**Definition 25** (Epigraph). *The epigraph of  $f$  is the set  $\{(x, y) : y \geq f(x)\}$ . This is the set of all points "above" the function.*

<sup>1</sup>[tikz/convexity-definition.pdf](#), from [tikz/convexity-definition.pdf](#). [tikz/convexity-definition.pdf](#), [tikz/convexity-definition.pdf](#).

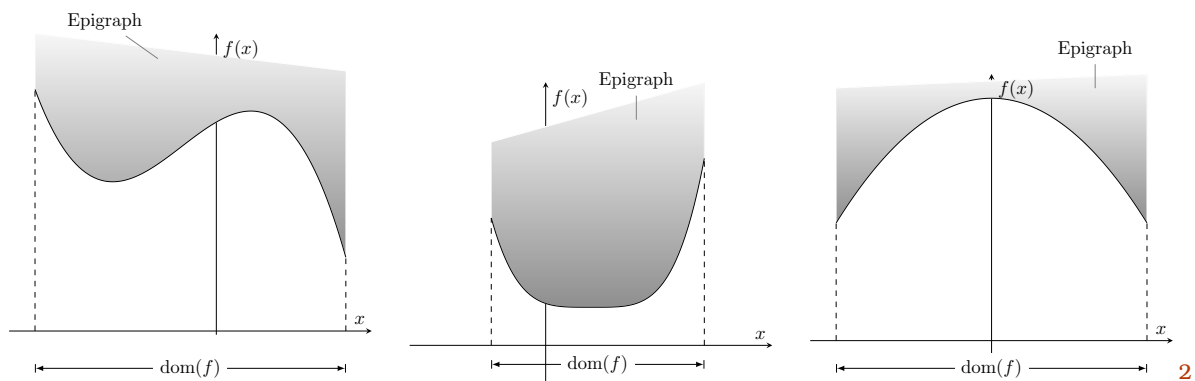
<sup>1</sup><https://tex.stackexchange.com/questions/394923/how-one-can-draw-a-convex-function>



© tikz/convexity-definition.pdf<sup>1</sup>

**Figure 6.3:** Illustration explaining the definition of a convex function.

**Theorem 26.**  $f(x)$  is a convex function if and only if the epigraph of  $f$  is a convex set.



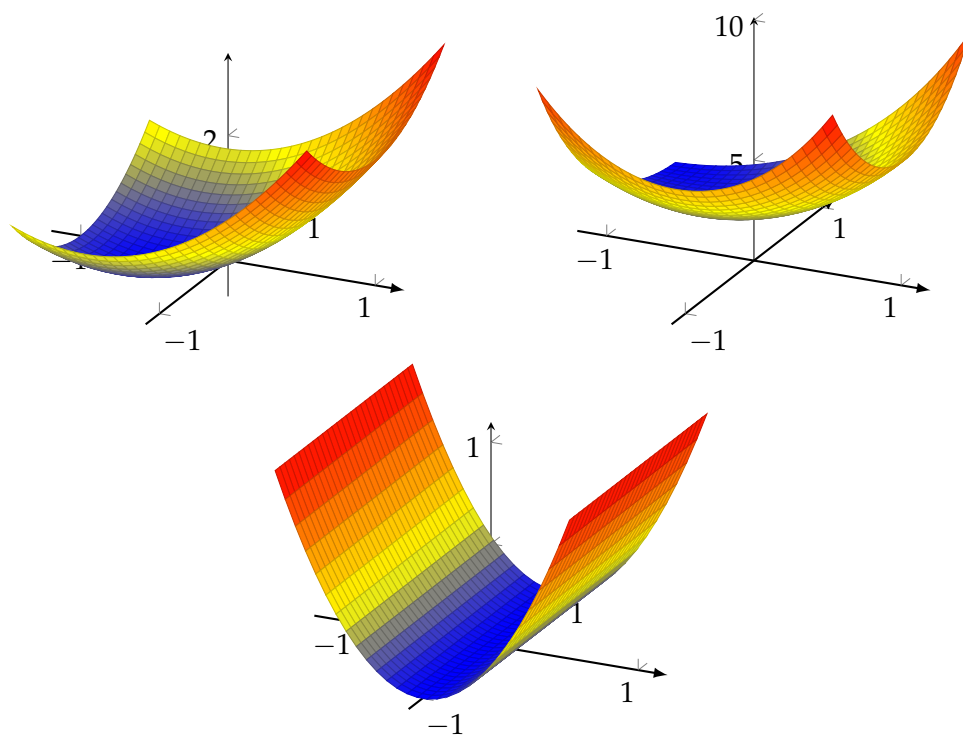
### Example 27: Examples of Convex functions

Some examples are

- $f(x) = ax + b$
- $f(x) = x^2$
- $f(x) = x^4$

<sup>2</sup><https://tex.stackexchange.com/questions/261501/function-epigraph-possibly-using-fillbetween>

- $f(x) = |x|$
- $f(x) = e^x$
- $f(x) = -\sqrt{x}$  on the domain  $[0, \infty)$ .
- $f(x) = x^3$  on the domain  $[0, \infty)$ .
- $f(x, y) = \sqrt{x^2 + y^2}$
- $f(x, y) = x^2 + y^2 + x$
- $f(x, y) = e^{x+y}$
- $f(x, y) = e^x + e^y + x^2 + (3x + 4y)^6$

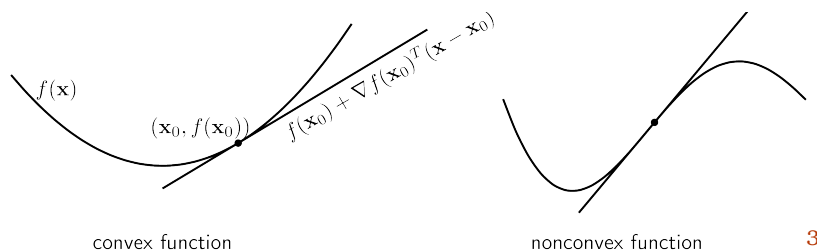


**Figure 6.4:** Convex Functions  $f(x, y) = x^2 + y^2 + x$ ,  $f(x, y) = e^{x+y} + e^{x-y} + e^{-x-y}$ , and  $f(x, y) = x^2$ .

### 6.2.1 Proving Convexity - Characterizations

**Theorem 27** (Convexity: First order characterization - linear underestimates). Suppose that  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is differentiable. Then  $f$  is convex if and only if for all  $\bar{x} \in \mathbb{R}^n$ , then linear tangent is an underestimator to the function, that is,

$$f(\bar{x}) + (x - \bar{x})^\top \nabla f(\bar{x}) \leq f(x).$$



**Theorem 28** (Convexity: Second order characterization - positive curvature). *We give statements for uni-variate functions and multi-variate functions.*

- Suppose  $f: \mathbb{R} \rightarrow \mathbb{R}$  is twice differentiable. Then  $f$  is convex if and only if  $f''(x) \geq 0$  for all  $x \in \mathbb{R}$ .
- Suppose  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is twice differentiable. Then  $f$  is convex if and only if  $\nabla^2 f(x) \succeq 0$  for all  $x \in \mathbb{R}^n$ .

### 6.2.2 Proving Convexity - Composition Tricks

#### Positive Scaling of Convex Function is Convex:

If  $f$  is convex and  $\alpha > 0$ , then  $\alpha f$  is convex.

**Example:**  $f(x) = e^x$  is convex. Therefore,  $25e^x$  is also convex.

#### Sum of Convex Functions is Convex:

If  $f$  and  $g$  are both convex, then  $f + g$  is also convex.

**Example:**  $f(x) = e^x, g(x) = x^4$  are convex. Therefore,  $e^x + x^4$  is also convex.

#### Composition with affine function:

If  $f(x)$  is convex, then  $f(a^\top x + b)$  is also convex.

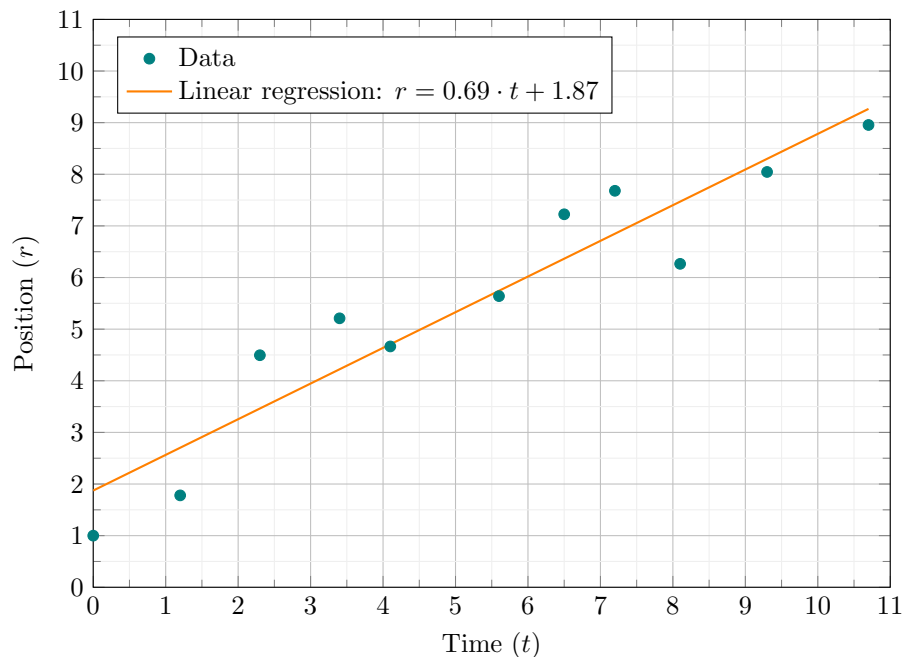
**Example:**  $f(x) = x^4$  are convex. Therefore,  $(3x + 5y + 10z)^4$  is also convex.

#### Pointwise maximum:

If  $f_i$  are convex for  $i = 1, \dots, t$ , then  $f(x) = \max_{i=1, \dots, t} f_i(x)$  is convex.

**Example:**  $f_1(x) = e^{-x}, f_2(x) = e^x$  are convex. Therefore,  $f(x) = \max(e^x, e^{-x})$  is also convex.

<sup>3</sup><https://machinelearningcoban.com/2017/03/12/convexity/>



© © Latex Draw CC BY-SA 4.0.<sup>4</sup>

**Figure 6.5:** Line derived through linear regression.

#### Other compositions:

Suppose

$$f(x) = h(g(x)).$$

1. If  $g$  is convex,  $h$  is convex and **non-decreasing**, then  $f$  is convex.
2. If  $g$  is concave,  $h$  is convex and **non-increasing**, then  $f$  is convex.

**Example 1:**  $g(x) = x^4$  is convex,  $h(x) = e^x$  is convex and non-decreasing. Therefore,  $f(x) = e^{x^4}$  is also convex.

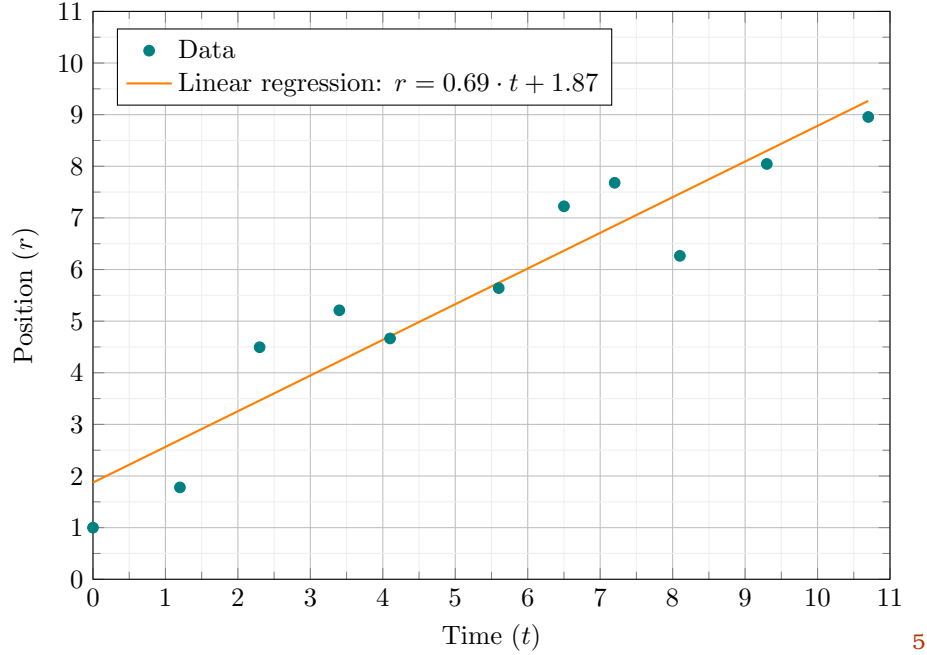
**Example 2:**  $g(x) = \sqrt{x}$  is concave (on  $[0, \infty)$ ),  $h(x) = e^{-x}$  is convex and non-increasing. Therefore,  $f(x) = e^{-\sqrt{x}}$  is convex on  $x \in [0, \infty)$ .

## 6.3 Convex Optimization Examples

### 6.3.1 Unconstrained Optimization: Linear Regression

Given data points  $x^1, \dots, x^N \in \mathbb{R}^d$  and out values  $y^i \in \mathbb{R}$ , we want to find a linear function  $y = \beta \cdot x$  that best approximates  $x^i \cdot \beta \approx y^i$ . For example, the data could  $x = (\text{time})$  and the output could be  $y = \text{position}$ .

<sup>4</sup>Linear Regression, from [Source:https://latexdraw.com/linear-regression-in-latex-using-tikz/](https://latexdraw.com/linear-regression-in-latex-using-tikz/), %20Adaptation:https://github.com/open-optimization-or-book/tree/master/content/figures/figures-source/tikz/linear-regression.tex. © Latex Draw CC BY-SA 4.0., 2020.



As is standard, we choose the error (or "loss") from each data point as the squared error. Hence, we can model this as the optimization problem:

$$\min_{\beta \in \mathbb{R}^d} \sum_{i=1}^N (x^i \cdot \beta - y^i)^2 \quad (6.3.1)$$

This problem has a nice closed form solution. We will derive this solution, and then in a later section discuss why using this solution might be too slow to compute on large data sets. In particular, the solution comes as a system of linear equations. But when  $N$  is really large, we may not have time to solve this system, so an alternative is to use decent methods, discussed later in this chapter.

**Theorem 29** (Linear Regression Solution). *The solution to (??) is*

$$\beta = (X^\top X)^{-1} X^\top Y, \quad (6.3.2)$$

where

$$X = \begin{bmatrix} x^1 \\ \vdots \\ x^N \end{bmatrix} = \begin{bmatrix} x_1^1 & x_2^1 & \dots & x_d^1 \\ \vdots & \vdots & \dots & \vdots \\ x_1^N & x_2^N & \dots & x_d^N \end{bmatrix}$$

*Proof.* Solve for  $\nabla f(\beta) = 0$ .

To be completed....

□

<https://www.youtube.com/watch?v=E5RjzSK0fvY>

<sup>5</sup><https://latexdraw.com/linear-regression-in-latex-using-tikz/>



## 6.4 Machine Learning - SVM

*Support Vector Machine* (SVM) is a tool used in machine learning for classifying data points. For instance, if there are red and black data points, how can we find a good line that separates them? The input data that you are given is as follows:

### Input:

- $d$ -dimensional data points  $x^1, \dots, x^N$
- 1-dimensional labels  $z^1, \dots, z^N$  (typically we will use  $z_i$  is either 1 or  $-1$ )

The output to the problem should be a hyperplane  $w^\top x + b = 0$  that separates the two data types (either exact separation or approximate separation).

### Output:

- A  $d$ -dimensional vector  $w$
- A 1-dimensional value  $b$

Given this output, we can construct a classification function  $f(x)$  as

$$f(x) = \begin{cases} 1 & \text{if } w^\top x + b \geq 0, \\ -1 & \text{if } w^\top x + b < 0. \end{cases} \quad (6.4.1)$$

There are three versions to consider:

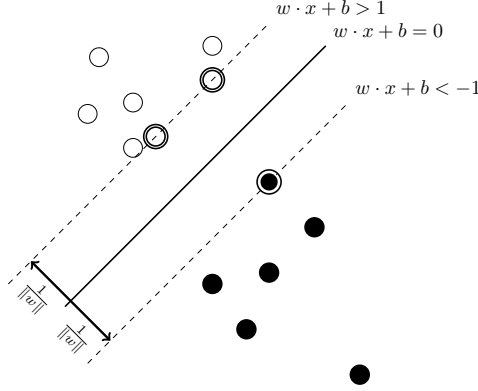
### Feasible separation

If we only want to a line that separates the data points, we can use the following optimization model.

$$\begin{aligned} \min \quad & 0 \\ \text{such that} \quad & z^i(w^\top x^i + b) \geq 1 && \text{for all } i = 1, \dots, N \\ & w \in \mathbb{R}^d \\ & b \in \mathbb{R} \end{aligned}$$

### SVM

We can modify the objective function to find a best separation between the points. This can be done in the following way



$$\begin{aligned}
 \min \quad & \|w\|_2^2 \\
 \text{such that} \quad & z^i(w^\top x^i + b) \geq 1 \quad \text{for all } i = 1, \dots, N \\
 & w \in \mathbb{R}^d \\
 & b \in \mathbb{R}
 \end{aligned}$$

Here,  $\|w\|_2^2 = \sum_{i=1}^d w_i^2 = w_1^2 + \dots + w_d^2$ .

### Approximate SVM

We can modify the objective function and the constraints to allow for approximate separation. This would be the case when you want to ignore outliers that don't fit well with the data set, or when exact SVM is not possible. This is done by changing the constraints to be

$$z^i(w^\top x^i + b) \geq 1 - \delta_i$$

where  $\delta_i \geq 0$  is the error in the constraint for datapoint  $i$ . In order to reduce these errors, we add a penalty term in the objective function that encourages these errors to be small. For this, we can pick some number  $C$  and write the objective as

$$\min \|w\|_2^2 + C \sum_{i=1}^N \delta_i.$$

This creates the following optimization problem

$$\begin{aligned}
 \min \quad & \|w\|_2^2 + C \sum_{i=1}^N \delta_i \\
 \text{such that} \quad & z^i(w^\top x^i + b) \geq 1 - \delta_i \quad \text{for all } i = 1, \dots, N \\
 & w \in \mathbb{R}^d \\
 & b \in \mathbb{R} \\
 & \delta_i \geq 0 \text{ for all } i = 1, \dots, N
 \end{aligned}$$

See information about the scikit-learn module for svm here: <https://scikit-learn.org/stable/modules/svm.html>.

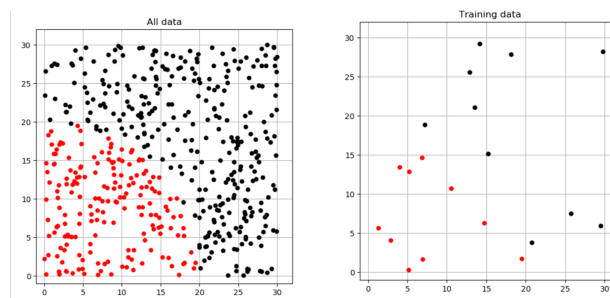
### 6.4.1 SVM with non-linear separators

[https://www.youtube.com/watch?time\\_continue=6&v=N1v0golbjSc](https://www.youtube.com/watch?time_continue=6&v=N1v0golbjSc)

Suppose for instance you are given data  $x^1, \dots, x^N \in \mathbb{R}^2$  (2-dimensional data) and given labels are dependent on the distance from the origin, that is, all data points  $x$  with  $x_1^2 + x_2^2 > r$  are given a label  $+1$  and all data points with  $x_1^2 + x_2^2 \leq r$  are given a label  $-1$ . That is, we want to learn the function

$$f(x) = \begin{cases} 1 & \text{if } x_1^2 + x_2^2 > r, \\ -1 & \text{if } x_1^2 + x_2^2 \leq r. \end{cases} \quad (6.4.2)$$

#### Example 28:



Here we have a classification problem where the data cannot be separated by a hyperplane. On the left, we have all of the data given to use. On the right, we have a subset of the data that we could try using for training and then test our learned function on the remaining data. As we saw in class, this amount of data was not sufficient to properly classify most of the data.

We cannot learn this classifier from the data directly using the hyperplane separation with SVM in the last section. But, if we modify the data set, then we can do this.

For each data point  $x$ , we transform it into a data point  $X$  by adding a third coordinate equal to  $x_1^2 + x_2^2$ . That is

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow X = \begin{pmatrix} x_1 \\ x_2 \\ x_1^2 + x_2^2 \end{pmatrix}. \quad (6.4.3)$$

In this way, we convert the data  $x^1, \dots, x^N$  into data  $X^1, \dots, X^N$  that lives in a higher-dimensional space. But with this new dataset, we can apply the hyperplane separation technique in the last section to properly classify the data.

This can be done with other nonlinear classification functions.

### 6.4.2 Support Vector Machines

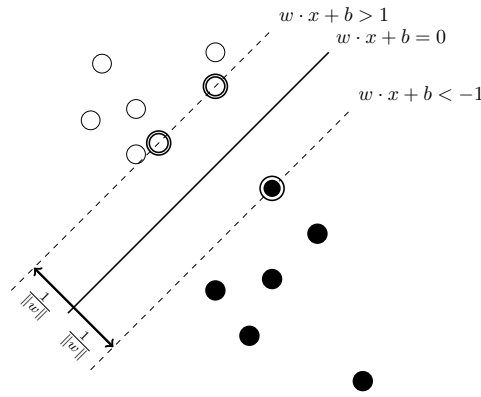
**Support Vector Machine - Exact Classification:**

Given labeled data  $(x^i, y_i)$  for  $i = 1, \dots, N$ , where  $x^i \in \mathbb{R}^d$  and  $y_i \in \{-1, 1\}$ , find a vector  $w \in \mathbb{R}^d$  and a number  $b \in \mathbb{R}$  such that

$$x^i \cdot w + b > 0 \quad \text{if } y^i = 1 \quad (6.4.4)$$

$$x^i \cdot w + b < 0 \quad \text{if } y^i = -1 \quad (6.4.5)$$

There may exist many solutions to this problem. Thus, we are interested in the "best" solution. Such a solution will maximize the separation between the two sets of points. To consider an equal margin on either side, we set the right hand sides to 1 and -1 and then compute the margin from the hyperplane. Notice that it is sufficient to use 1 and -1 on the right hand sides since any scaling can happen in  $w$  and  $b$ .



We will show that the margin under this model can be computed as  $\frac{2}{\|w\|}$  where  $\|w\| = \sqrt{w_1^2 + \dots + w_d^2}$ . Hence, maximizing the margin is equivalent to minimizing  $w_1^2 + \dots + w_d^2$ . We arrive at the model

$$\min \sum_{i=1}^d w_i^2 \quad (6.4.6)$$

$$x^i \cdot w + b \geq 1 \quad \text{if } y^i = 1 \quad (6.4.7)$$

$$x^i \cdot w + b \leq -1 \quad \text{if } y^i = -1 \quad (6.4.8)$$

Or even more compactly written as

$$\min \sum_{i=1}^d w_i^2 \quad (6.4.9)$$

$$y^i(x^i \cdot w + b) \geq 1 \quad \text{for } i = 1, \dots, N \quad (6.4.10)$$

## 6.5 Classification

### 6.5.1 Machine Learning

[https://www.youtube.com/watch?v=bwZ3Qiuuj3i8&list=PL9ooVrP1hQ0HUfd-g8GUpKI3hH0wM\\_9Dn&index=13](https://www.youtube.com/watch?v=bwZ3Qiuuj3i8&list=PL9ooVrP1hQ0HUfd-g8GUpKI3hH0wM_9Dn&index=13)

<https://towardsdatascience.com/solving-a-simple-classification-problem-with-python-fruits-lo>

### 6.5.2 Neural Networks

<https://www.youtube.com/watch?v=bVQUSndD1lU>

<https://www.youtube.com/watch?v=8bNikfRJZpo>

<https://www.youtube.com/watch?v=Dws9Zveu9ug>



## 6.6 Box Volume Optimization in Scipy.Minimize

<https://www.youtube.com/watch?v=iSnTtV6b0Gw>

## 6.7 Modeling

We will discuss a few models and mention important changes to the models that will make them solvable.

### Important tips

1. **Find a convex formulation.** It may be that the most obvious model for your problem is actually non-convex. Try to reformulate your model into one that is convex and hence easier for solvers to handle.
2. **Intelligent formulation.** Understanding the problem structure may help reduce the complexity of the problem. Try to deduce something about the solution to the problem that might make the problem easier to solve. This may work for special cases of the problem.
3. **Identify problem type and select solver.** Based on your formulation, identify which type of problem it is and which solver is best to use for that type of problem. For instance,  Gurobi can handle some convex quadratic problems, but not all. Ipopt is a more general solver, but may be slower due to the types of algorithms that it uses.
4. **Add bounds on the variables.** Many solvers perform much better if they are provided bounds to the variables. This is because it reduces the search region where the variables live. Adding good bounds could be the difference in the solver finding an optimal solution and not finding any solution at all.
5. **Warm start.** If you know good possible solutions to the problem (or even just a feasible solution), you can help the solver by telling it this solution. This will reduce the amount of work the solver needs to do. In  JUMP this can be done by using the

command `setvalue(x,[2 4 6])`, where here it sets the value of vector  $x$  to  $[2 \ 4 \ 6]$ . It may be necessary to specify values for all variables in the problem for it to start at.

6. **Rescaling variables.** It sometimes is useful to have all variables on the same rough scale. For instance, if minimizing  $x^2 + 100^2 y^2$ , it may be useful to define a new variable  $\bar{y} = 100y$  and instead minimize  $x^2 + \bar{y}^2$ .
7. **Provide derivatives.** Working out gradient and hessian information by hand can save the solver time. Particularly when these are sparse (many zeros). These can often be provided directly to the solver.

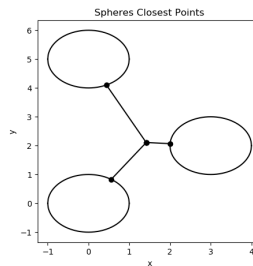
See <http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=4982C26EC5F25564BCC239FD3785E2D3?doi=10.1.1.210.3547&rep=rep1&type=pdf> for many other helpful tips on using Ipopt.

### 6.7.1 Minimum distance to circles

The problem we will consider here is: Given  $n$  circles, find a center point that minimizes the sum of the distances to all of the circles.

#### Minimize distance to circles:

Given circles described by center points  $(a_i, b_i)$  and radius  $r_i$  for  $i = 1, \dots, n$ , find a point  $c = (c_x, c_y)$  that minimizes the sum of the distances to the circles.



#### Minimize distance to circles - Model attempt #1:

Non-convex

Let  $(x_i, y_i)$  be a point in circle  $i$ . Let  $w_i$  be the distance from  $(x_i, y_i)$  to  $c$ . Then we can model the problem as follows:

$$\begin{aligned}
 \min \quad & \sum_{i=1}^3 w_i && \text{Sum of distances} \\
 \text{s.t.} \quad & \sqrt{(x_i - a_i)^2 + (y_i - b_i)^2} = r_i, \quad i = 1, \dots, n && (x_i, y_i) \text{ is in circle } i \\
 & \sqrt{(x_i - c_x)^2 + (y_i - c_y)^2} = w_i \quad i = 1, \dots, n && w_i \text{ is distance from } (x_i, y_i) \text{ to } c
 \end{aligned}
 \tag{6.7.1}$$

**This model has several issues:**

1. If the center  $c$  lies inside one of the circles, then the constraint  $\sqrt{(x_i - a_i)^2 + (y_i - b_i)^2} = r$  may not be valid. This is because the optimal choice for  $(x_i, y_i)$  in this case would be inside the circle, that is, satisfying  $\sqrt{(x_i - a_i)^2 + (y_i - b_i)^2} \leq r$ .
2. This model is **nonconvex**. In particular the equality constraints make the problem nonconvex.

Fortunately, we can relax the problem to make it convex and still model the correct solution. In particular, consider the constraint

$$\sqrt{(x_i - c_x)^2 + (y_i - c_y)^2} = w_i.$$

Since we are minimizing  $\sum w_i$ , it is equivalent to have the constraint

$$\sqrt{(x_i - c_x)^2 + (y_i - c_y)^2} \leq w_i.$$

This is equivalent because any optimal solution make  $w_i$  the smallest it can, and hence will meet that constraint at equality.

What is great about this change, it that it makes the constraint **convex!**. To see this we can write  $f(z) = \|z\|_2^2$ ,  $z = (x_i - c_x, y_i - c_y)$ . Since  $f(z)$  is convex and the transformation into variables  $x_i, c_x, y_i, c_y$  is linear, we have that  $f(x_i - c_x, y_i - c_y)$  is convex. Then since  $-w_i$  is linear, we have that

$$f(x_i - c_x, y_i - c_y) - w_i$$

is a convex function. Thus, the constraint

$$f(x_i - c_x, y_i - c_y) - w_i \leq 0$$

is a convex constraint.

This brings us to our second model.

#### Minimize distance to circles - Model attempt #2:

Convex

Let  $(x_i, y_i)$  be a point in circle  $i$ . Let  $w_i$  be the distance from  $(x_i, y_i)$  to  $c$ . Then we can model the problem as follows:

$$\begin{array}{ll} \min & \sum_{i=1}^3 w_i & \text{Sum of distances} \\ \text{s.t.} & \sqrt{(x_i - a_i)^2 + (y_i - b_i)^2} \leq r, \quad i = 1, \dots, n & (x_i, y_i) \text{ is in circle } i \\ & \sqrt{(x_i - c_x)^2 + (y_i - c_y)^2} \leq w_i \quad i = 1, \dots, n & w_i \text{ is distance from } (x_i, y_i) \text{ to } c \end{array} \quad (6.7.2)$$

Lastly, we would like to make this model better for a solver. For this we will

1. Add bounds on all the variables
2. Change format of non-linear inequalities

**Minimize distance to circles - Model attempt #3:**

Convex

Let  $(x_i, y_i)$  be a point in circle  $i$ . Let  $w_i$  be the distance from  $(x_i, y_i)$  to  $c$ . Then we can model the problem as follows:

$$\begin{aligned}
 \min \quad & \sum_{i=1}^3 w_i && \text{Sum of distances} \\
 \text{s.t.} \quad & (x_i - a_i)^2 + (y_i - b_i)^2 \leq r^2, \quad i = 1, \dots, n && (x_i, y_i) \text{ is in circle } i \\
 & (x_i - c_x)^2 + (y_i - c_y)^2 \leq w_i^2 \quad i = 1, \dots, n && w_i \text{ is distance from } (x_i, y_i) \text{ to } c \\
 & 0 \leq w_i \leq u_i && \\
 & a_i - r \leq x_i \leq a_i + r && \\
 & b_i - r \leq y_i \leq b_i + r && 
 \end{aligned} \tag{6.7.3}$$

**Example: Minimize distance to circles**

[Code: ??]

Here we minimize the distance of three circles of radius 1 centered at  $(0,0)$ ,  $(3,2)$ , and  $(0,5)$ . Note: The bounds on the variables here are not chosen optimally.

$$\begin{aligned}
 \min \quad & w_1 + w_2 + w_3 \\
 \text{Subject to} \quad & (x_1 - 0)^2 + (y_1 - 0)^2 \leq 1 \\
 & (x_2 - 3)^2 + (y_2 - 2)^2 \leq 1 \\
 & (x_3 - 0)^2 + (y_3 - 5)^2 \leq 1 \\
 & (x_1 - c_x)^2 + (y_1 - c_y)^2 \leq w_1^2 \\
 & (x_2 - c_x)^2 + (y_2 - c_y)^2 \leq w_2^2 \\
 & (x_3 - c_x)^2 + (y_3 - c_y)^2 \leq w_3^2 \\
 & -1 \leq x_i \leq 10 \quad \forall i \in \{1, 2, 3\} \\
 & -1 \leq y_i \leq 10 \quad \forall i \in \{1, 2, 3\} \\
 & 0 \leq w_i \leq 40 \quad \forall i \in \{1, 2, 3\} \\
 & -1 \leq c_x \leq 10 \\
 & -1 \leq c_y \leq 10
 \end{aligned}$$

**6.8 Machine Learning**

There are two main fields of machine learning:

- Supervised Machine Learning,
- Unsupervised Machine Learning.



Supervised machine learning is composed of *Regression* and *Classification*. This area is thought of as being given labeled data that you are then trying to understand the trends of this labeled data.

Unsupervised machine learning is where you are given unlabeled data and then need to decide how to label this data. For instance, how can you optimally partition the people in a room into 5 groups that share the most commonalities?

## 6.9 Machine Learning - Supervised Learning - Regression

See the video lecture information.

## 6.10 Machine learning - Supervised Learning - Classification

The problem of data *classification* begins with *data* and *labels*. The goal is *classification* of future data based on sample data that you have by constructing a function to understand future data.

**Goal:** *Classification - create a function  $f(x)$  that takes in a data point  $x$  and outputs the correct label.*

These functions can take many forms. In binary classification, the label set is  $\{+1, -1\}$ , and we want to correctly (as often as we can) determine the correct label for a future data point.

There are many ways to determine such a function  $f(x)$ . In the next section, we will learn about SVM that determines the function by computing a hyperplane that separates the data labeled  $+1$  from the data labeled  $-1$ .

Later, we will learn about *neural networks* that describe much more complicated functions.

Another method is to create a *decision tree*. These are typically more interpretable functions (neural networks are often a bit mysterious) and thus sometimes preferred in settings where the classification should be easily understood, such as a medical diagnosis. We will not discuss this method here since it fits less well with the theme of nonlinear programming.

### 6.10.1 Python SGD implementation and video

[https://github.com/llSourcell/Classifying\\_Data\\_Using\\_a\\_Support\\_Vector\\_Machine/blob/master/support\\_vector\\_machine\\_lesson.ipynb](https://github.com/llSourcell/Classifying_Data_Using_a_Support_Vector_Machine/blob/master/support_vector_machine_lesson.ipynb)



# Chapter 7

## NLP Algorithms

### 7.1 Algorithms Introduction

We will begin with unconstrained optimization and consider several different algorithms based on what is known about the objective function. In particular, we will consider the cases where we use

- Only function evaluations (also known as *derivative free optimization*),
- Function and gradient evaluations,
- Function, gradient, and hessian evaluations.

We will first look at these algorithms and their convergence rates in the 1-dimensional setting and then extend these results to higher dimensions.

### 7.2 1-Dimensional Algorithms

We suppose that we solve the problem

$$\min f(x) \tag{7.2.1}$$

$$x \in [a, b]. \tag{7.2.2}$$

That is, we minimize the univariate function  $f(x)$  on the interval  $[l, u]$ .

For example,

$$\min (x^2 - 2)^2 \tag{7.2.3}$$

$$0 \leq x \leq 10. \tag{7.2.4}$$

Note, the optimal solution lies at  $x^* = \sqrt{2}$ , which is an irrational number. Since we will consider algorithms using floating point precision, we will look to return a solution  $\bar{x}$  such that  $\|x^* - \bar{x}\| < \epsilon$  for some small  $\epsilon > 0$ , for instance,  $\epsilon = 10^{-6}$ .

### 7.2.1 Golden Search Method - Derivative Free Algorithm

<https://www.youtube.com/watch?v=hLm8xfwWYPw>

Suppose that  $f(x)$  is unimodal on the interval  $[a, b]$ , that is, it is a continuous function that has a single minimizer on the interval.

Without any extra information, our best guess for the optimizer is  $\bar{x} = \frac{a+b}{2}$  with a maximum error of  $\epsilon = \frac{b-a}{2}$ . Our goal is to reduce the size of the interval where we know  $x^*$  to be, and hence improve our best guess and the maximum error of our guess.

Now we want to choose points in the interior of the interval to help us decide where the minimizer is. Let  $x_1, x_2$  such that

$$a < x_2 < x_1 < b.$$

Next, we evaluate the function at these four points. Using this information, we would like to argue a smaller interval in which  $x^*$  is contained. In particular, since  $f$  is unimodal, it must hold that

1.  $x^* \in [a, x_2]$  if  $f(x_1) \leq f(x_2)$ ,
2.  $x^* \in [x_1, b]$  if  $f(x_2) < f(x_1)$ ,

After comparing these function values, we can reduce the size of the interval and hence reduce the region where we think  $x^*$  is.

We will now discuss how to choose  $x_1, x_2$  in a way that we can

1. Reuse function evaluations,
2. Have a constant multiplicative reduction in the size of the interval.

We consider the picture:

To determine the best  $d$ , we want to decrease by a constant factor. Hence, we decrease by a factor  $\gamma$ , which we will see is the golden ration (GR). To see this, we assume that  $(b - a) = 1$ , and ask that  $d = \gamma$ . Thus,  $x_1 - a = \gamma$  and  $b - x_2 = \gamma$ . If we are in case 1, then we cut off  $b - x_1 = 1 - \gamma$ . Now, if we iterate and do this again, we will have an initial length of  $\gamma$  and we want to cut off the interval  $x_2 - x_1$  with this being a proportion of  $(1 - \gamma)$  of the remaining length. Hence, the second time we will cut off  $(1 - \gamma)\gamma$ , which we set as the length between  $x_1$  and  $x_2$ .

Considering the geometry, we have

$$\text{length } a \text{ to } x_1 + \text{length } x_2 \text{ to } b = \text{total length} + \text{length } x_2 \text{ to } x_1$$

hence

$$\gamma + \gamma = 1 + (1 - \gamma)\gamma.$$

Simplifying, we have

$$\gamma^2 + \gamma - 1 = 0.$$

Applying the quadratic formula, we see

$$\gamma = \frac{-1 \pm \sqrt{5}}{2}.$$

Since we want  $\gamma > 0$ , we take

$$\gamma = \frac{-1 + \sqrt{5}}{2} \approx 0.618$$

This is exactly the Golden Ratio (or, depending on the definition, the golden ration minus 1).

**Example:**

We can conclude that the optimal solution is in  $[1.4, 3.8]$ , so we would guess the midpoint  $\bar{x} = 2.6$  as our approximate solution with a maximum error of  $\epsilon = 1.2$ .

**Convergence Analysis of Golden Search Method:**

After  $t$  steps of the Golden Search Method, the interval in question will be of length

$$(b - a)(GR)^t \approx (b - a)(0.618)^t$$

Hence, by guessing the midpoint, our worst error could be

$$\frac{1}{2}(b - a)(0.618)^t.$$

### 7.2.2 Bisection Method - 1st Order Method (using Derivative)

**Minimization Interpretation**

**Assumptions:**  $f$  is convex, differentiable

We can look for a minimizer of the function  $f(x)$  on the interval  $[a, b]$ .

**Root finding Interpretation**

Instead of minimizing, we can look for a root of  $f'(x)$ . That is, find  $x$  such that  $f'(x) = 0$ .

**Assumptions:**  $f'(a) < 0 < f'(b)$ , **OR**,  $f'(b) < 0 < f'(a)$ .  $f'$  is continuous

The goal is to find a root of the function  $f'(x)$  on the interval  $[a, b]$ . If  $f$  is convex, then we know that this root is indeed a global minimizer.

Note that if  $f$  is convex, it only makes sense to have the assumption  $f'(a) < 0 < f'(b)$ .

**Convergence Analysis of Bisection Method:**

After  $t$  steps of the Bisection Method, the interval in question will be of length

$$(b - a) \left( \frac{1}{2} \right)^t.$$

Hence, by guessing the midpoint, our worst error could be

$$\frac{1}{2}(b-a) \left(\frac{1}{2}\right)^t.$$

### 7.2.3 Gradient Descent - 1st Order Method (using Derivative)

**Input:**  $f(x)$ ,  $\nabla f(x)$ , initial guess  $x^0$ , learning rate  $\alpha$ , tolerance  $\epsilon$

**Output:** An approximate solution  $x$

1. Set  $t = 0$
2. While  $\|f(x^t)\|_2 > \epsilon$ :
  - (a) Set  $x^{t+1} \leftarrow x^t - \alpha \nabla f(x^t)$ .
  - (b) Set  $t \leftarrow t + 1$ .
3. Return  $x^t$ .

### 7.2.4 Newton's Method - 2nd Order Method (using Derivative and Hessian)

**Input:**  $f(x)$ ,  $\nabla f(x)$ ,  $\nabla^2 f(x)$ , initial guess  $x^0$ , learning rate  $\alpha$ , tolerance  $\epsilon$

**Output:** An approximate solution  $x$

1. Set  $t = 0$
2. While  $\|f(x^t)\|_2 > \epsilon$ :
  - (a) Set  $x^{t+1} \leftarrow x^t - \alpha [\nabla^2 f(x^t)]^{-1} \nabla f(x^t)$ .
  - (b) Set  $t \leftarrow t + 1$ .
3. Return  $x^t$ .

## 7.3 Multi-Variate Unconstrained Optimizaiton

We will now use the techniques for 1-Dimensional optimization and extend them to multi-variate case. We will begin with unconstrained versions (or at least, constrained to a large box) and then show how we can apply these techniques to constrained optimization.

### 7.3.1 Descent Methods - Unconstrained Optimization - Gradient, Newton

**Outline for Descent Method for Unconstrained Optimization:**

**Input:**

- A function  $f(x)$
- Initial solution  $x^0$
- Method for computing step direction  $d_t$
- Method for computing length  $t$  of step
- Number of iterations  $T$

**Output:**

- A point  $x_T$  (hopefully an approximate minimizer)

**Algorithm**

1. For  $t = 1, \dots, T$ ,

$$\text{set } x_{t+1} = x_t + \alpha_t d_t$$

**Choice of  $\alpha_t$** 

There are many different ways to choose the step length  $\alpha_t$ . Some choices have proofs that the algorithm will converge quickly. An easy choice is to have a constant step length  $\alpha_t = \alpha$ , but this may depend on the specific problem.

**Choice of  $d_t$  using  $\nabla f(x)$** 

Choice of descent methods using  $\nabla f(x)$  are known as *first order methods*. Here are some choices:

1. **Gradient Descent:**  $d_t = -\nabla f(x_t)$
2. **Nesterov Accelerated Descent:**  $d_t = \mu(x_t - x_{t-1}) - \gamma \nabla f(x_t + \mu(x_t - x_{t-1}))$

Here,  $\mu, \gamma$  are some numbers. The number  $\mu$  is called the momentum.

**7.3.2 Stochastic Gradient Descent - The mother of all algorithms.**

A popular method is called *stochastic gradient descent* (SGD). This has been described as "The mother of all algorithms". This is a method to **approximate the gradient** typically used in machine learning or stochastic programming settings.

**Stochastic Gradient Descent:**

Suppose we want to solve

$$\min_{x \in \mathbb{R}^n} F(x) = \sum_{i=1}^N f_i(x). \quad (7.3.1)$$

We could use *gradient descent* and have to compute the gradient  $\nabla F(x)$  at each iteration. But! We see that in the **cost to compute the gradient** is roughly  $O(nN)$ , that is, it is very dependent on the number of function  $N$ , and hence each iteration will take time dependent on  $N$ .

**Instead!** Let  $i$  be a uniformly random sample from  $\{1, \dots, N\}$ . Then we will use  $\nabla f_i(x)$  as an approximation of  $\nabla F(x)$ . Although we lose a bit by using a guess of the gradient, this approximation only takes  $O(n)$  time to compute. And in fact, in expectation, we are doing the same thing. That is,

$$N \cdot \mathbb{E}(\nabla f_i(x)) = N \sum_{i=1}^N \frac{1}{N} \nabla f_i(x) = \sum_{i=1}^N \nabla f_i(x) = \nabla \left( \sum_{i=1}^N f_i(x) \right) = \nabla F(x).$$

Hence, the SGD algorithm is:

1. Set  $t = 0$
2. While ...(some stopping criterion)
  - (a) Choose  $i$  uniformly at random in  $\{1, \dots, N\}$ .
  - (b) Set  $d_t = \nabla f_i(x_t)$
  - (c) Set  $x_{t+1} = x_t - \alpha d_t$

There can be many variations on how to decide which functions  $f_i$  to evaluate gradient information on. Above is just one example.

Linear regression is an excellent example of this.

**Example 30:** Linear Regression with SGD Given data points  $x^1, \dots, x^N \in \mathbb{R}^d$  and output  $y^1, \dots, y^N \in \mathbb{R}$ , find  $a \in \mathbb{R}^d$  and  $b \in \mathbb{R}$  such that  $a^\top x^i + b \approx y^i$ . This can be written as the optimization problem

$$\min_{a,b} \sum_{i=1}^N g_i(a,b) \tag{7.3.2}$$

where  $g_i(a,b) = (a^\top x^i + b)^2$ .

Notice that the objective function  $G(a,b) = \sum_{i=1}^N g_i(a,b)$  is a convex quadratic function. The gradient of the objective function is

$$\nabla G(a,b) = \sum_{i=1}^N \nabla g_i(a,b) = \sum_{i=1}^N 2x^i(a^\top x^i + b)$$

Hence, if we want to use gradient descent, we must compute this large sum (think of  $N \approx 10,000$ ).

Instead, we can **approximate the gradient!**. Let  $\tilde{\nabla} G(a,b)$  be our approximate gradient. We will compute this by randomly choosing a value  $r \in \{1, \dots, N\}$  (with uniform probability). Then set



$$\tilde{\nabla}G(a, b) = \nabla g_r(a, b).$$

It holds that the expected value is the same as the gradient, that is,

$$\mathbb{E}(\tilde{\nabla}G(a, b)) = \nabla G(a, b).$$

Hence, we can make probabilistic arguments that these two will have the same (or similar) convergence properties (in expectation).

### Choice of $\Delta_k$ using the hessian $\nabla^2 f(x)$

These choices are called *second order methods*

1. **Newton's Method:**  $\Delta_k = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k)$
2. **BFGS (Quasi-Newton):**  $\Delta_k = -(B_k)^{-1} \nabla f(x_k)$

Here

$$\begin{aligned} s_k &= x_{k+1} - x_k \\ y_k &= \nabla f(x_{k+1}) - \nabla f(x_k) \end{aligned}$$

and

$$B_{k+1} = B_k - \frac{(B_k s_k)(B_k s_k)^\top}{s_k^\top B_k s_k} + \frac{y_k y_k^\top}{y_k^\top s_k}.$$

This serves as an approximation of the hessian and can be efficiently computed. Furthermore, the inverse can be easily computed using certain updating rules. This makes for a fast way to approximate the hessian.

## 7.4 Constrained Convex Nonlinear Programming

Given a convex function  $f(x): \mathbb{R}^d \rightarrow \mathbb{R}$  and convex functions  $f_i(x): \mathbb{R}^d \rightarrow \mathbb{R}$  for  $i = 1, \dots, m$ , the *convex programming* problem is

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & f_i(x) \leq 0 \quad \text{for } i = 1, \dots, m \\ & x \in \mathbb{R}^d \end{aligned} \tag{7.4.1}$$

### 7.4.1 Barrier Method

**Constrained Convex Programming via Barrier Method:**

We convert (??) into the unconstrained minimization problem:

$$\begin{aligned} \min \quad & f(x) - \phi \sum_{i=1}^m \log(-f_i(x)) \\ & x \in \mathbb{R}^d \end{aligned} \quad (7.4.2)$$

Here  $\phi > 0$  is some number that we choose. As  $\phi \rightarrow 0$ , the optimal solution  $x(\phi)$  to (??) tends to the optimal solution of (??). That is  $x(\phi) \rightarrow x^*$  as  $\phi \rightarrow 0$ .

**Constrained Convex Programming via Barrier Method - Initial solution:**

Define a variable  $s \in \mathbb{R}$  and add that to the right hand side of the inequalities and then minimize it in the objective function.

$$\begin{aligned} \min \quad & s \\ \text{s.t.} \quad & f_i(x) \leq s \quad \text{for } i = 1, \dots, m \\ & x \in \mathbb{R}^d, s \in \mathbb{R} \end{aligned} \quad (7.4.3)$$

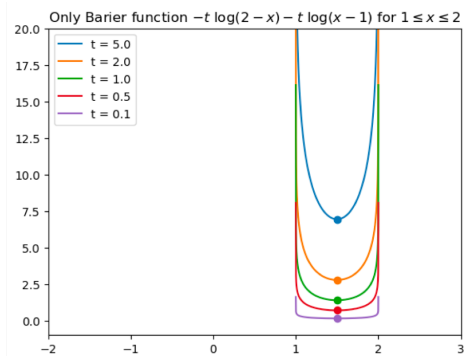
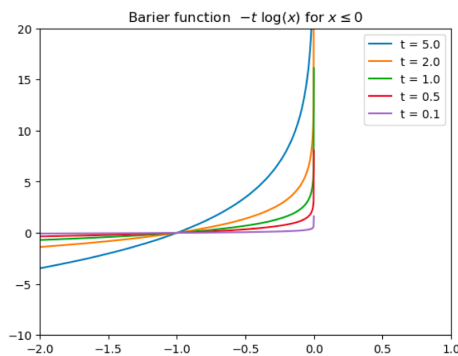
Note that this problem is feasible for all  $x$  values since  $s$  can always be made larger. If there exists a solution with  $s \leq 0$ , then we can use the corresponding  $x$  solution as an initial feasible solution. Otherwise, the problem is infeasible.

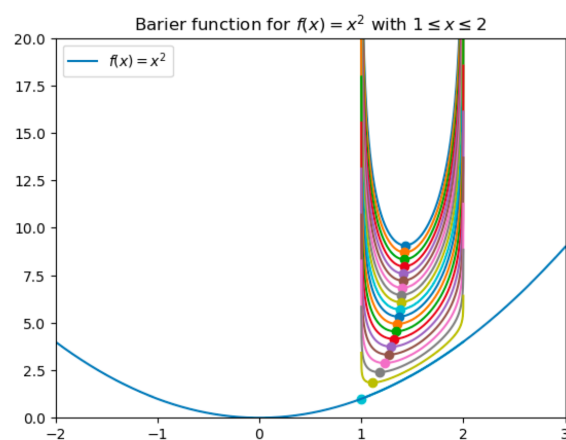
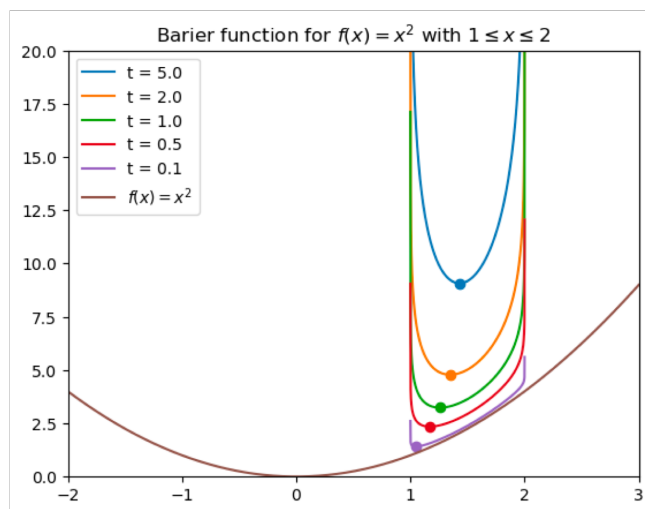
Now, convert this problem into the unconstrained minimization problem:

$$\begin{aligned} \min \quad & f(x) - \phi \sum_{i=1}^m \log(-(f_i(x) - s)) \\ & x \in \mathbb{R}^d, s \in \mathbb{R} \end{aligned} \quad (7.4.4)$$

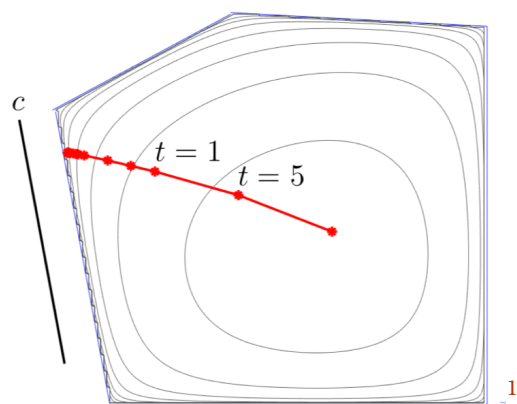
This problem has an easy time of finding an initial feasible solution. For instance, let  $x = 0$ , and then  $s = \max_i f_i(x) + 1$ .

**Images below: the value  $t$  is the value  $\phi$  discussed above**





Minimizing  $c^T x$  subject to  $Ax \leq b$ .



<sup>1</sup>Image taken from unknown source.



## Chapter 8

# Computational Issues with NLP

We mention a few computational issues to consider with nonlinear programs.

### 8.1 Irrational Solutions

Consider nonlinear problem (this is even convex)

$$\begin{array}{ll} \min & -x \\ \text{s.t.} & x^2 \leq 2. \end{array} \quad (8.1.1)$$

The optimal solution is  $x^* = \sqrt{2}$ , which cannot be easily represented. Hence, we would settle for an **approximate solution** such as  $\bar{x} = 1.41421$ , which is feasible since  $\bar{x}^2 \leq 2$ , and it is close to optimal.

### 8.2 Discrete Solutions

Consider nonlinear problem (not convex)

$$\begin{array}{ll} \min & -x \\ \text{s.t.} & x^2 = 2. \end{array} \quad (8.2.1)$$

Just as before, the optimal solution is  $x^* = \sqrt{2}$ , which cannot be easily represented. Furthermore, the only two feasible solutions are  $\sqrt{2}$  and  $-\sqrt{2}$ . Thus, there is no chance to write down a feasible rational approximation.

### 8.3 Convex NLP Harder than LP

Convex NLP is typically polynomially solvable. It is a generalization of linear programming.

**Convex Programming:**  
*Polynomial time (P)* (typically)

Given a convex function  $f(x): \mathbb{R}^d \rightarrow \mathbb{R}$  and convex functions  $f_i(x): \mathbb{R}^d \rightarrow \mathbb{R}$  for  $i = 1, \dots, m$ , the *convex programming* problem is

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & f_i(x) \leq 0 \quad \text{for } i = 1, \dots, m \\ & x \in \mathbb{R}^d \end{aligned} \tag{8.3.1}$$

**Example 31:** Convex programming is a generalization of linear programming. This can be seen by letting  $f(x) = c^\top x$  and  $f_i(x) = A_i x - b_i$ .

## 8.4 NLP is harder than IP

As seen above, quadratic constraints can be used to create a feasible region with discrete solutions. For example

$$x(1 - x) = 0$$

has exactly two solutions:  $x = 0, x = 1$ . Thus, quadratic constraints can be used to model binary constraints.

### Binary Integer programming (BIP) as a NLP:

*NP-Hard*

Given a matrix  $A \in \mathbb{R}^{m \times n}$ , vector  $b \in \mathbb{R}^m$  and vector  $c \in \mathbb{R}^n$ , the *binary integer programming* problem is

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \\ & \cancel{x \in \{0, 1\}^n} \\ & x_i(1 - x_i) = 0 \quad \text{for } i = 1, \dots, n \end{aligned} \tag{8.4.1}$$

## 8.5 Karush-Huhn-Tucker (KKT) Conditions

The KKT conditions use the augmented Lagrangian problem to describe sufficient conditions for optimality of a convex program.

**KKT Conditions for Optimality:**

Given a convex function  $f(x): \mathbb{R}^d \rightarrow \mathbb{R}$  and convex functions  $g_i(x): \mathbb{R}^d \rightarrow \mathbb{R}$  for  $i = 1, \dots, m$ , the *convex programming* problem is

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & g_i(x) \leq 0 \quad \text{for } i = 1, \dots, m \\ & x \in \mathbb{R}^d \end{aligned} \tag{8.5.1}$$

Given  $(\bar{x}, \bar{\lambda})$  with  $\bar{x} \in \mathbb{R}^d$  and  $\bar{\lambda} \in \mathbb{R}^m$ , if the KKT conditions hold, then  $\bar{x}$  is optimal for the convex programming problem.

The KKT conditions are

1. (Stationary).

$$-\nabla f(\bar{x}) = \sum_{i=1}^m \bar{\lambda}_i \nabla g_i(\bar{x}) \tag{8.5.2}$$

2. (Complimentary Slackness).

$$\bar{\lambda}_i g_i(\bar{x}) = 0 \text{ for } i = 1, \dots, m \tag{8.5.3}$$

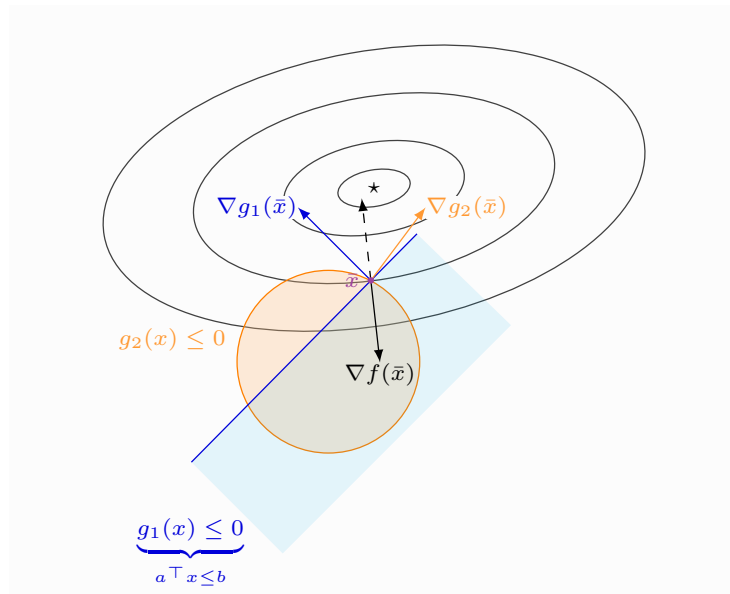
3. (Primal Feasibility).

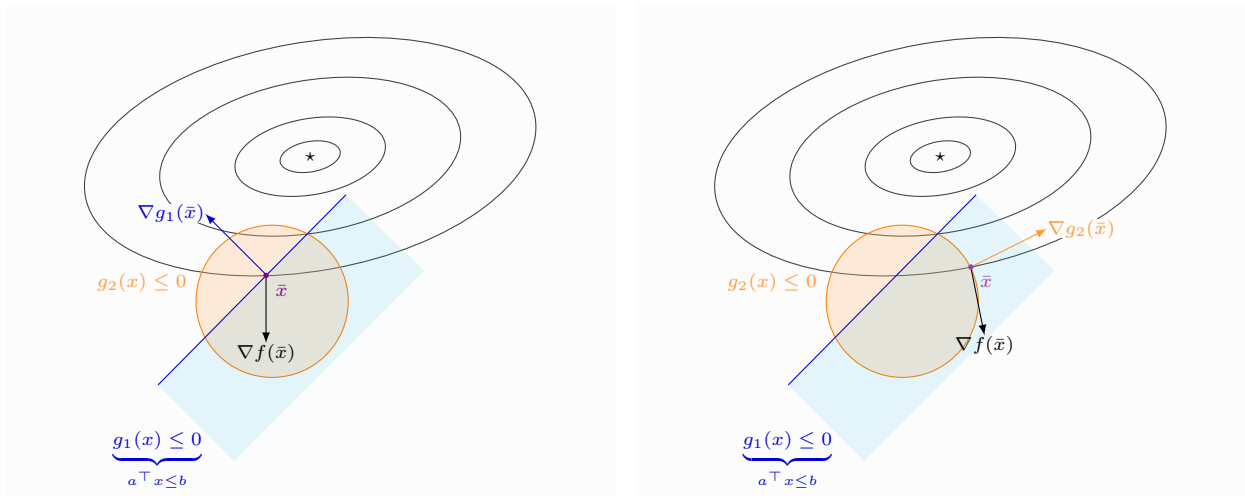
$$g_i(\bar{x}) \leq 0 \text{ for } i = 1, \dots, m \tag{8.5.4}$$

4. (Dual Feasibility).

$$\bar{\lambda}_i \geq 0 \text{ for } i = 1, \dots, m \tag{8.5.5}$$

If certain properties are true of the convex program, then every optimizer has these properties. In particular, this holds for Linear Programming.



© © Robert Hildebrand CC BY-SA 4.0.<sup>1</sup>© © Robert Hildebrand CC BY-SA 4.0.<sup>2</sup>

A non-optimal point, from <https://github.com/open-optimization/open-optimization-or-book/tree/master/content/figures/figures-source/tikz/kkt-non-optimal1.tex>. © Robert Hildebrand CC BY-SA 4.0., 2020.

© © Robert Hildebrand CC BY-SA 4.0.<sup>3</sup>

A non-optimal point, from <https://github.com/open-optimization/open-optimization-or-book/tree/master/content/figures/figures-source/tikz/kkt-non-optimal1.tex>. © Robert Hildebrand CC BY-SA 4.0., 2020.

## 8.6 Gradient Free Algorithms

### 8.6.1 Needler-Mead

[Wikipedia](#)

[Youtube](#)

<sup>1</sup>KKT Optimal Point, from <https://github.com/open-optimization/open-optimization-or-book/tree/master/content/figures/figures-source/tikz/kkt-optimal.tex>. © Robert Hildebrand CC BY-SA 4.0., 2020.



# Chapter 9

## Material to add...

### 9.0.1 Bisection Method and Newton's Method

See section 4 of the following nodes: <http://www.seas.ucla.edu/~vandenbe/133A/133A-notes.pdf>

### 9.1 Gradient Descent

Recap Gradient and Directional Derivatives: <https://www.youtube.com/watch?v=tIpKfDc295M>

<https://www.youtube.com/watch?v=-02ze7tf08>

[https://www.youtube.com/watch?v=N\\_ZRcLheNv0](https://www.youtube.com/watch?v=N_ZRcLheNv0)

<https://www.youtube.com/watch?v=4RBkIJPG6Yo>

Idea of Gradient descent:

<https://youtu.be/IHZwWFHwa-w?t=323>

Vectors:

[https://www.youtube.com/watch?v=fNk\\_zzaMoSs&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE\\_ab&index=2&t=0s](https://www.youtube.com/watch?v=fNk_zzaMoSs&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab&index=2&t=0s)



## **Part II**

# **Advanced Linear Programming**



## 9.2 Duality

Video! - [Michel Belaire \(EPFL\) teaching LP duality](#)



## **Part III**

# **Advanced Integer Programming**





# Chapter 10

## Integral polyhedra, TU matrices, TDI systems

### 10.1 Integral polyhedra

#### 10.1.1 Basics

**Definition 30** (Integral polyhedron). A polyhedron  $P$  is called integral if every minimal face of  $P$  contains an integral vector.

**Remark 31.** If  $P$  has vertices, then  $P$  is integral if and only if every vertex is an integral vector.

#### 10.1.2 Properties

**Theorem 32.** Let  $P \subseteq \mathbb{R}^n$  be a polyhedron. Then the following are equivalent:

1.  $P = \text{conv}(P \cap \mathbb{Z}^n)$
2.  $P$  is integral
3.  $\max\{c^T x : x \in P\}$  has an integral optimal solution for all  $c \in \mathbb{R}^n$  such that the optimal value is finite.
4.  $\max\{c^T x : x \in P\}$  has an integral optimal solution for all  $c \in \mathbb{Z}^n$  such that the optimal value is finite.
5.  $\max\{c^T x : x \in P\}$  is an integer for all  $c \in \mathbb{Z}^n$  such that the optimal value is finite.

### 10.2 Unimodular and totally unimodular matrices

#### 10.2.1 Unimodular matrices

**Definition 33** (Unimodular matrix). A matrix  $A \in \mathbb{R}^{m \times n}$  is called unimodular if: (1) All entries are integers. (2)  $A$  has full rank. (3) Every  $m \times m$  square submatrix of  $A$  has determinant  $-1, 0, 1$ .

**Theorem 34.** Let  $A \in \mathbb{Z}^{m \times n}$  be a full row rank matrix. Then the polyhedron  $P = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$  is integral for all  $b \in \mathbb{Z}^m$  if and only if  $A$  is unimodular.

### 10.2.2 Totally unimodular matrices

**Definition 35** (Totally unimodular matrix). *The matrix  $A \in \mathbb{R}^{m \times n}$  is called totally unimodular if every square submatrix of  $A$  has determinant  $-1, 0, 1$ .*

**Theorem 36.** *Let  $A \in \mathbb{Z}^{m \times n}$ . Then the polyhedron  $P = \{x \in \mathbb{R}^n : Ax \leq b, x \geq 0\}$  is integral for all  $b \in \mathbb{Z}^m$  if and only if  $A$  is totally unimodular.*

**Theorem 37.** *Let  $A \in \mathbb{Z}^{m \times n}$  be a totally unimodular matrix. Then the polyhedron  $P = \{x \in \mathbb{R}^n : Ax \leq b, x \geq 0\}$  is integral for all  $b \in \mathbb{Z}^m$ .*

### 10.2.3 How to detect unimodularity and totally unimodularity

**Theorem 38** (Basic properties). *Let  $A \in \mathbb{Z}^{n \times m}$ . Then the following are equivalent:*

1.  $A$  is totally unimodular
2.  $A^T$  is totally unimodular
3.  $[A \ I]$  is totally unimodular (where  $I \in \mathbb{R}^n$  denotes the identity matrix)
4.  $[A \ I]$  is unimodular

**Theorem 39.** *Let  $A \in \mathbb{Z}^{m \times n}$ . Then  $A$  is totally unimodular if and only if for all  $J \subseteq \{1, \dots, m\}$  there exists  $J_1, J_2$  such that*

1.  $J_1 \cap J_2 = \emptyset$  and  $J = J_1 \cup J_2$
2. For all  $i = 1, \dots, n$  we have

$$\left| \sum_{j \in J_1} a_{ji} - \sum_{j \in J_2} a_{ji} \right| \leq 1$$

**Remark 40.** *A analogous result can be written in terms of the columns instead of the rows of  $A$ .*

### 10.2.4 Examples of totally unimodular matrices

Classical examples of matrices that are totally unimodular are: network flow matrices, the node-incidence matrix for a bipartite graph, interval matrices.

## 10.3 Totally dual integral systems

### 10.3.1 Basics

**Definition 41** (Totally dual integral system). *Let  $A \in \mathbb{Q}^{m \times n}$  and  $b \in \mathbb{Q}^m$ . The system  $Ax \leq b$  is totally dual integral system (TDI) if for each integral vector  $c \in \mathbb{Z}^n$  such that*

$$\max\{c^T x : Ax \leq b\}$$

*is finite, then the dual*

$$\min\{b^T y : A^T y = c, y \geq 0\}$$

*has an integral optimal solution.*

### 10.3.2 Properties

**Theorem 42.** *Let  $A \in \mathbb{Q}^{m \times n}$  and  $b \in \mathbb{Z}^m$ . If  $Ax \leq b$  is TDI then  $P = \{x \in \mathbb{R}^n : Ax \leq b\}$  is an integral polyhedron.*

**Remark 43.** *The condition  $b \in \mathbb{Z}^m$  is crucial in the proof of the theorem above.*

### 10.3.3 Totally unimodularity and TDI systems

**Theorem 44.** *Let  $A \in \mathbb{Q}^{m \times n}$  be a totally unimodular matrix. Then the system  $Ax \leq b$  is TDI for all  $b \in \mathbb{R}^m$ .*

### 10.3.4 Examples of TDI systems

Classical examples of TDI systems are: the independent set formulation for matroids, matchings.



# Chapter 11

## Cutting Planes

### 11.1 Introduction

#### 11.1.1 Cutting planes

**Definition 45** (Cutting plane for IP). Let  $P \subseteq \mathbb{R}^n$  be a polyhedron. An inequality  $a^T x \leq b$  is called a cutting plane if

$$P \cap \mathbb{Z}^n \subseteq \{x \in \mathbb{R}^n : a^T x \leq b\}.$$

**Definition 46** (Cutting plane for MIP). Let  $P \subseteq \mathbb{R}^n$  be a polyhedron. An inequality  $a^T x \leq b$  is called a cutting plane if

$$P \cap (\mathbb{Z}^{n_1} \times \mathbb{R}^{n_2}) \subseteq \{x \in \mathbb{R}^n : a^T x \leq b\},$$

where we are assuming that in the MIP only the first  $n_1$  variables must be integers ( $n = n_1 + n_2$ ).

#### 11.1.2 Cutting plane algorithm

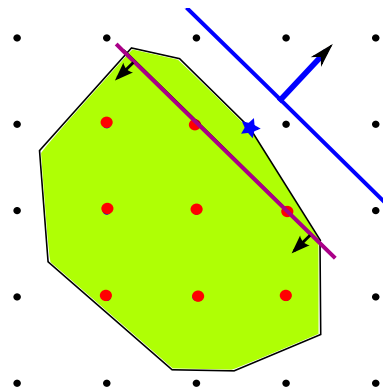
##### Generic cutting plane algorithm

1. **Solve** LP (continuous relaxation of MILP).
2. If solution of LP is **fractional**: add cutting plane and go to (1.)
3. If solution of LP is **integral**: **STOP**.

#### 11.1.3 How to compute cutting planes

Two approaches:

1. Computing cutting planes for general IPs.
  - From “Algebraic” properties: CG cuts, MIR inequalities, functional cuts, etc.



- From “Geometric” properties: lattice-free cuts, etc.
2. Computing cutting planes for specific IPs.
- Knapsack problem, Node packing, etc. (many many other examples...)

## 11.2 Computing cutting planes for general IPs

### 11.2.1 Chvátal-Gomory cuts (for pure integer programs)

**Definition 47** (Chvátal-Gomory cut for  $P$ ). Let  $P \subseteq \mathbb{R}^n$  be a polyhedron. Let  $a \in \mathbb{Z}^n$ ,  $b \in \mathbb{R}$  and let  $a^T x \leq b$  be a valid inequality for  $P$ . Then the inequality

$$a^T x \leq \lfloor b \rfloor$$

is called a Chvátal-Gomory cut.

**Remark 48.** Some examples of CG cuts are: blossom inequalities for the matching problem, clique inequalities for the independent set problem, Gomory’s fractional cut.

#### A nice property of CG cuts

**Definition 49** (Chvátal-Gomory closure of  $P$ ). Let  $P \subseteq \mathbb{R}^n$  be a polyhedron. Then the set

$$P' = P \cap \bigcap_{\substack{\alpha^T x \leq \beta \\ \text{is a CG cut for } P}} \{x \in \mathbb{R}^n : \alpha^T x \leq \beta\}$$

is called a Chvátal-Gomory closure.

**Theorem 50** (Finiteness of the CG cuts procedure). Let  $P_0$  be a rational polyhedron and for  $k \in \mathbb{Z}_+$  define  $P^{k+1} = (P^k)'$ . Then

1. For all  $k \in \mathbb{Z}_+$ ,  $P^k$  is again a rational polyhedron.
2. There exists  $t \in \mathbb{Z}_+$  such that  $P^t = P_t$ .

### 11.2.2 Cutting planes from the Simplex tableau

Assume  $P = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$  where  $A \in \mathbb{R}^{m \times n}$  is a full-row rank matrix. Let  $B, N$  denote the basic and nonbasic variables defining a vertex  $(\hat{x}_B, \hat{x}_N)$  of  $P$  (where  $\hat{x}_N = 0$ ). You can write the constraints defining  $P$  in terms of the basis  $B$ :

$$\begin{aligned} x_B &= \bar{b} - \bar{A}_N x_N \\ x_B, x_N &\geq 0, \end{aligned}$$

where  $\bar{b} = A_B^{-1}b$  and  $\bar{A}_N = A_B^{-1}A_N$ .

Denote  $\bar{b} = (\bar{b}_i)_{i \in B}$  and  $\bar{A}_N = (\bar{a}_{ij})_{i \in B, j \in N}$ . Assume that  $\bar{b}_i \notin \mathbb{Z}$ , so the vertex is fractional (that is,  $(\hat{x}_B, \hat{x}_N) \notin \mathbb{Z}^n$ ), and therefore, we would want to cut off that LP solution.

**Remark 51.** Recall that the vertex  $(\hat{x}_B, \hat{x}_N)$  is the only feasible point in  $P$  satisfying  $x_N = 0$ . We will use this fact in order to derive some cutting planes.

### A simple inequality

The following is a valid inequality that cuts off the fractional vertex:

$$\sum_{j \in N} x_j \geq 1.$$

### 11.2.3 A stronger inequality

Let  $N_f = \{j \in N : \bar{a}_{ij} \text{ is fractional}\}$ . Then the following is a valid inequality that cuts off the fractional vertex:

$$\sum_{j \in N_f} x_j \geq 1.$$

### Gomory's fractional cut

The following inequality can be derived as a CG cut:

$$\sum_{j \in N} (\bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor) x_j \geq (\bar{b}_i - \lfloor \bar{b}_i \rfloor).$$

It can be verified that this valid inequality cuts off the fractional vertex.

## 11.3 Cutting planes from lattice free sets

### 11.3.1 The general case

**Definition 52** (Lattice-free sets). *A set  $L \subseteq \mathbb{R}^n$  is a lattice-free set if it does not contain any integral vector in its (topological) interior, that is,  $\text{int}(L) \cap \mathbb{Z}^n = \emptyset$ .*

Let  $P$  be a polyhedron and let  $L$  be a lattice-free convex set. Then, we can derive cutting planes from  $L$  by using the following fact:

$$P \cap \mathbb{Z}^n \subseteq P \setminus \text{int}(L).$$

Such a cutting plane is called a cutting plane derived from a lattice-free set.

**Remark 53.** *It suffice to consider only the cutting planes defining facets of  $\text{conv}(P \setminus \text{int}(L))$  as all the cuts not defining these facets are redundant.*

**Definition 54** (Maximal lattice-free convex sets). *A maximal lattice-free is a lattice-free convex set that is not strictly contained in any other lattice-free convex set.*

Maximal lattice-free convex sets are important since they give stronger cuts, since  $L' \subseteq L$  implies  $P \setminus \text{int}(L) \subseteq P \setminus \text{int}(L')$ . The following is a nice property of such sets:

**Theorem 55.**  *$L$  is a maximal lattice-free convex set if and only if  $L$  is a polyhedron satisfying certain “simple characterization”.*

**Remark 56.** *The fact that all maximal lattice-free convex set are polyhedra is useful because if  $L$  a polyhedron, then cutting planes for the set  $P \setminus \text{int}(L)$  are likely ‘easy’ to obtain.*

### 11.3.2 Split cuts

A special case of a maximal lattice-free convex set is the case of split sets.

**Definition 57** (Split set, split cuts). *Let  $\pi \in \mathbb{Z}^n$  and let  $\pi_0 \in \mathbb{Z}$ . Then, a split set is a set of the form*

$$\{x \in \mathbb{R}^n : \pi_0 < \pi^T x < \pi_0 + 1\}.$$

*A split cut is a any cutting plane valid for  $P \setminus S$ , where  $S$  is some split set.*

**Remark 58.** *The set  $P \setminus S$  can be seen as a disjunction. Let  $S = \{x \in \mathbb{R}^n : \pi_0 < \pi^T x < \pi_0 + 1\}$ , then*

$$P \setminus S = \{x \in P : \pi^T x \leq \pi_0\} \cup \{x \in P : \pi_0 + 1 \leq \pi^T x\}.$$

*In general, disjunctions as the one given by a split set or more general ones are very useful to derived cutting planes for integer programs.*

## 11.4 Mixed-integer rounding cuts (MIR)

### 11.4.1 Basic MIR inequality

Let  $B = \{(u, v) \in \mathbb{Z} \times \mathbb{R} : u + v \geq b, v \geq 0\}$ . Then the inequality

$$v \geq (b - \lfloor b \rfloor)(\lceil b \rceil - u)$$

is valid for the set  $B$ .

### 11.4.2 MIR inequalities from one-row relaxations

Let  $P$  be a polyhedron. We want to find valid inequalities for  $P \cap (\mathbb{Z}^{|I|} \times \mathbb{R}^{|J|})$ .

#### One-row relaxation

A set of the form

$$Q = \left\{ (x, y) \in \mathbb{R}^{|I|} \times \mathbb{R}^{|J|} : \sum_{i \in I} a_i x_i + \sum_{j \in J} c_j y_j \geq b, x, y \geq 0 \right\}$$

is a one-row relaxation of  $P$  if  $P \subseteq Q$ .

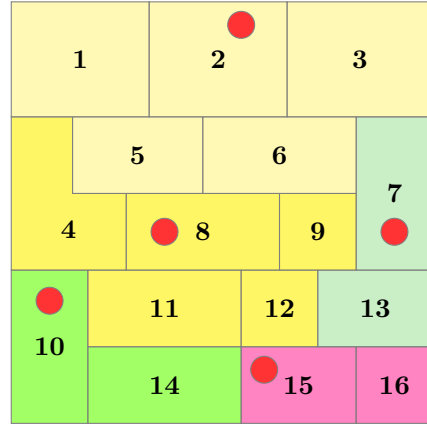
**Remark 59.** *One-row relaxations can be constructed by using any valid inequality for  $P$ . In particular one could obtain a valid inequality by combining rows of the matrix and vector defining  $P$ .*

#### Applying the basic MIR inequality

We first relax the inequality defining  $Q$ . Let  $I' \subseteq I$  and consider the following mixed-integer set:

$$B = \left\{ (x, y) \in \mathbb{Z}^{|I|} \times \mathbb{R}^{|J|} : \left( \sum_{i \in I \setminus I'} x_i + \sum_{i \in I'} \lfloor a_i \rfloor x_i \right) + \left( \sum_{i \in I'} (a_i - \lfloor a_i \rfloor) x_i + \sum_{j \in J} \max\{0, c_j\} y_j \right) \geq b \right\}.$$



© tikz/Illustration1.pdf<sup>1</sup>**Figure 11.1:** tikz/Illustration1.pdf

Since the first part of the l.h.s. of the inequality is integral and the second part is non-negative, we can apply the procedure described in Section ?? . We obtain the following valid inequality for  $B$ :

$$\left( \sum_{i \in I'} (a_i - \lfloor a_i \rfloor) x_i + \sum_{j \in J} \max\{0, c_j\} y_j \right) \geq (b - \lfloor b \rfloor) \left( \lceil b \rceil - \left( \sum_{i \in I \setminus I'} x_i + \sum_{i \in I} \lfloor a_i \rfloor x_i \right) \right).$$

**Remark 60.** The above inequality is valid for  $P \cap (\mathbb{Z}^{|I|} \times \mathbb{R}^{|J|})$ , for all  $I' \subseteq I$ . The set  $I' = \{i \in I : (a_i - \lfloor a_i \rfloor) < (b - \lfloor b \rfloor)\}$  gives the strongest inequality of this form.

## 11.5 Gomory Mixed-integer cut (GMI)

Consider the one-row relaxation  $Q = \{(x, y) \in \mathbb{Z}^{|I|} \times \mathbb{R}^{|J|} : \sum_{i \in I} a_i x_i + \sum_{j \in J} c_j y_j = b, x, y \geq 0\}$ . Denote  $f_0 = b - \lfloor b \rfloor$  and for  $i \in I$  denote  $f_i = a_i - \lfloor a_i \rfloor$ .

We will assume that  $0 < f_0 < 1$ . In this case, the Gomory mixed-integer cut (GMI) is given by

$$\sum_{\substack{i \in I \\ f_i \leq f_0}} \frac{f_i}{f_0} x_i + \sum_{\substack{i \in I \\ f_i > f_0}} \frac{1 - f_i}{1 - f_0} x_i + \sum_{\substack{j \in J \\ c_j > 0}} \frac{c_j}{f_0} y_j + \sum_{\substack{j \in J \\ c_j < 0}} \frac{c_j}{1 - f_0} y_j \geq 1.$$

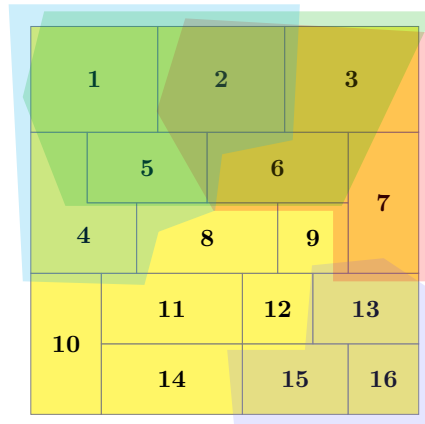
**Remark 61.**

- The validity of GMI cuts follows from the fact that they are also split cuts.
- In the pure integer programming case (that is,  $J = \emptyset$ ), GMI gives a cut that is stronger than the Gomory's fractional cut.

<sup>1</sup>tikz/Illustration1.pdf, from tikz/Illustration1.pdf. tikz/Illustration1.pdf, tikz/Illustration1.pdf.

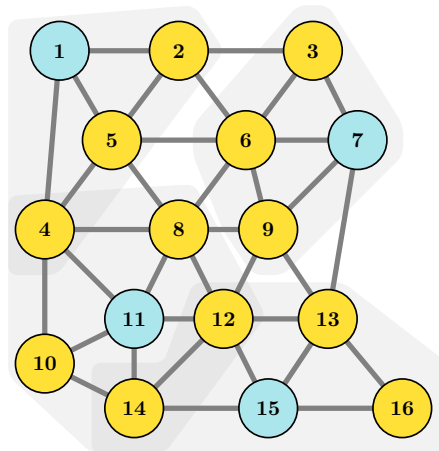
<sup>2</sup>tikz/Illustration2.pdf, from tikz/Illustration2.pdf. tikz/Illustration2.pdf, tikz/Illustration2.pdf.

<sup>3</sup>tikz/Illustration3.pdf, from tikz/Illustration3.pdf. tikz/Illustration3.pdf, tikz/Illustration3.pdf.



© tikz/Illustration2.pdf<sup>2</sup>

**Figure 11.2:** tikz/Illustration2.pdf



© tikz/Illustration3.pdf<sup>3</sup>

**Figure 11.3:** tikz/Illustration3.pdf

## **Part IV**

# **Advanced Nonlinear Programming**



# **Appendix A**

# **Complexity**

[Halting Problem Proof](#)



# Bibliography

- [5] T. F. Abdelmaguid. “An Efficient Mixed Integer Linear Programming Model for the Minimum Spanning Tree Problem”. In: *Mathematics* 6.10 (2018). issn: 2227-7390. doi: [10.3390/math6100183](https://doi.org/10.3390/math6100183) (cit. on p. 53).
- [6] D. Avis. “A survey of heuristics for the weighted matching problem”. In: *Networks* 13.4 (1983), pp. 475–493. doi: [10.1002/net.3230130404](https://doi.org/10.1002/net.3230130404) (cit. on p. 29).
- [9] D. E. Drake and S. Hougardy. “A simple approximation algorithm for the weighted matching problem”. In: *Information Processing Letters* 85.4 (2003), pp. 211–213. issn: 0020-0190. doi: [https://doi.org/10.1016/S0020-0190\(02\)00393-9](https://doi.org/10.1016/S0020-0190(02)00393-9) (cit. on p. 29).
- [12] M. L. Fisher. “The Lagrangian Relaxation Method for Solving Integer Programming Problems”. In: *Management Science* 50.12\_supplement (2004), pp. 1861–1871. doi: [10.1287/mnsc.1040.0263](https://doi.org/10.1287/mnsc.1040.0263).
- [13] P. C. Gilmore and R. E. Gomory. “A Linear Programming Approach to the Cutting-Stock Problem”. In: *Operations Research* 9.6 (1961), pp. 849–859. doi: [10.1287/opre.9.6.849](https://doi.org/10.1287/opre.9.6.849).