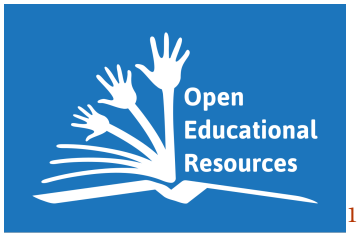

OPEN OPTIMIZATION
OPEN-SOURCE PROJECT FOR OPTIMIZATION COURSES

This work is done in align with the mission of UNESCO Open Educational Resources https://en.unesco.org/themes/building-knowledge-societies/oer



[1]

[1] https://en.wikipedia.org/wiki/Open_educational_resources#/media/File:Global_Open_Educational_Resources_Logo.svg

What is an Open Textbook?

Open textbooks are complete textbooks that have been funded, published, and licensed to be freely used, adapted, and distributed. As a particular type of Open Educational Resource (OER), this open textbook is intended to provide authoritative, accurate, and comprehensive subject content at no cost, to anyone including those who utilize technology to read and those who cannot afford traditional textbooks. This book is licensed with a Creative Commons Share-Alike 4.0 license (see p. iv.), which allows it to be adapted, remixed, and shared under the same license with attribution. Instructors and other readers may be interested in localizing, rearranging, or adapting content, or in transforming the content into other formats with the goal of better addressing student learning needs, and/or making use of various teaching methods.

Open textbooks in a variety of disciplines are available via the Open Textbook Library: https://open.umn.edu/opentextbooks.

Feedback Requested

The editor and author of this book seek content-related suggestions from faculty, students, and others using the book. Methods for providing feedback are presented in the User Feedback Guide at: http://bit.ly/userfeedbackguide

- Submit suggestions (anonymous) http://bit.ly/electromagnetics-suggestion
- Annotate using Hypothes.is http://web.hypothes.is (instructions are in the feedback guide)
- Send suggestions via email: publishing@vt.edu

Additional Resources

These following resources for Electromagnetics Vol 1 are available at: http://hdl.handle.net/10919/84164

- Downloadable PDF of the text
- Source files (LaTeX tarball)
- Errata for Volume 1
- Problem sets & solution manual
- Information for book's community portal and listserv

If you are a professor reviewing, adopting, or adapting this textbook please help us understand a little more about your use by filling out this form: http://bit-ly/vtpublishing-updates

2

[2]Take from open-source electromagnetics book

Mathematical Programming and Operations Research

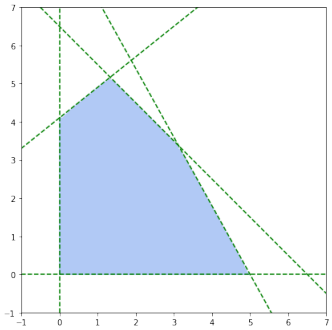
Modeling, Algorithms, and Complexity
Examples in Python and Julia
(Work in progress)

Edited by: Robert Hildebrand

Contributors:

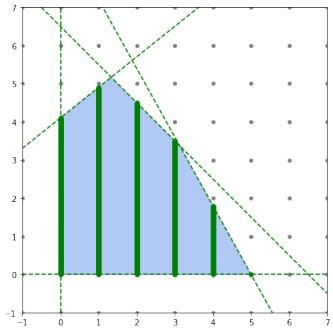
- Laurent Porrier

December 18, 2019



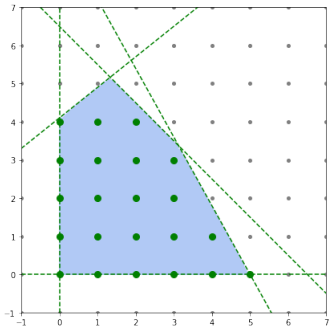
LP

$Ax \leq b$



MIP

$Ax \leq b$
$x_1 \in \mathbb{Z}$



IP

$Ax \leq b$
$x_1, x_2 \in \mathbb{Z}$

Front Matter

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 textbooks published through ???. All contributors to this repository will be acknowledged in any publication resulting from this material. Please see ???? as an example.

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)
    - Direct access 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

Contents

Introduction

s

0.1 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. $1, 2, 3, -1, -2, -3, ...$)
- $\mathbb{Q}$ - 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, π, 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. $[3, \pi, -e, \sqrt{2}]$)
- $\sum_{i=1}^{4} i = 1 + 2 + 3 + 4$

- $\sum_{i=1}^{4} i^2 = 1^2 + 2^2 + 3^2 + 4^4$
- $\sum_{i=1}^{4} x_i = x_1 + x_2 + x_3 + x_4$
- $\square$ - 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:

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

"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."

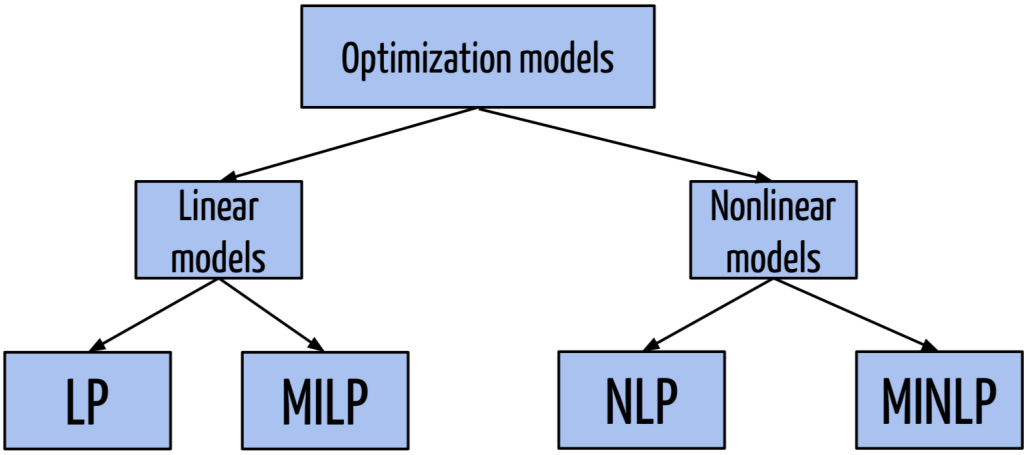
Part I

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).



[1]

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 **??**.

[1] Diagram by Diego Moran

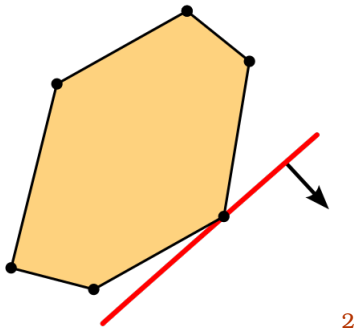


Figure 1.1: *Linear programming constraints and objective.*

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

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

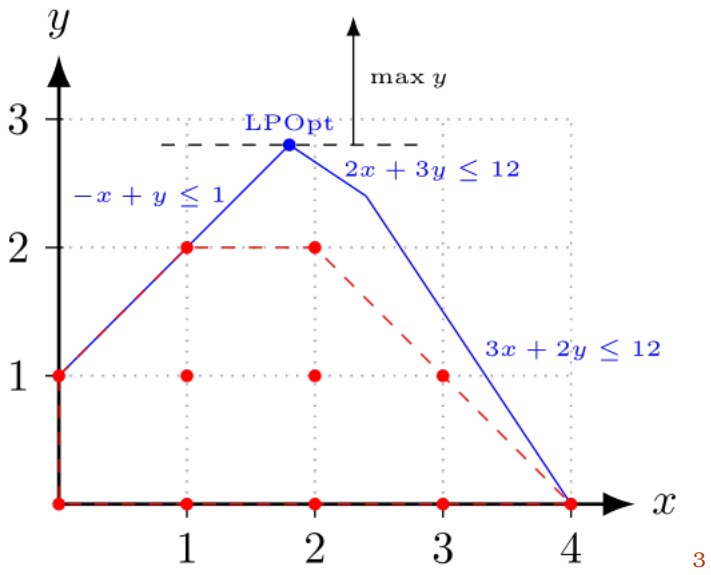
Linear programming can come in several forms, whether we are maximizing or minimizing, or if the constraints are $\leq, =$ or $\geq$. One form commonly used is *Standard Form* given as

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

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

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.*

[2] https://en.wikipedia.org/wiki/Linear_programming



[3]

Figure 1.2: *Comparing the LP relaxation to the IP solutions.*

1.2 Mixed-Integer Linear Programming (MILP)

Mixed-integer linear programming will be the focus of Sections 3,4, 5, and **??**. Recall that the notation $\mathbb{Z}$ means the set of integers and the set $\mathbb{R}$ means the set of real numbers. 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.

[3] Figure from https://en.wikipedia.org/wiki/Integer_programming.

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

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

An even more general class is *Mixed-Integer Linear Programming (MILP)*. This is where we have n integer variables $x_1, \ldots, x_n \in \mathbb{Z}$ and d continuous variables $x_{n+1}, \ldots, x_{n+d} \in \mathbb{R}$. 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 $f(x) \colon \mathbb{R}^d \to \mathbb{R}$ and other functions $f_i(x) \colon \mathbb{R}^d \to \mathbb{R}$ for $i = 1, \ldots, m$, the *nonlinear programming* problem is

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

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 $f(x)\colon \mathbb{R}^d \to \mathbb{R}$ and convex functions $f_i(x)\colon \mathbb{R}^d \to \mathbb{R}$ for $i = 1, \ldots, m$, the *convex programming* problem is

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

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

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

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

1.5.1 INFORMS Studies

https://www.informs.org/Impact
https://www.informs.org/Impact/O.R.-Analytics-Success-Stories/Optimized-school-bus-routing-helps-school-districts-design-better-policies

1.5.2 Employee Training Program

https://download.aimms.com/aimms/download/manuals/AIMMS3OM_EmployeeTraining.pdf

1.5.3 Media Selection Program

https://download.aimms.com/aimms/download/manuals/AIMMS3OM_MediaSelection.pdf

1.5.4 Diet Problem

https://download.aimms.com/aimms/download/manuals/AIMMS3OM_Diet.pdf

1.5.5 Farm Planning Problem

https://download.aimms.com/aimms/download/manuals/AIMMS3OM_FarmPlanning.pdf

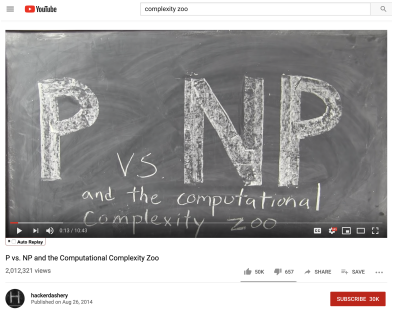
1.5.6 Pooling Problem

https://download.aimms.com/aimms/download/manuals/AIMMS3OM_Pooling.pdf

Chapter 2

Algorithms and Complexity

https://www.youtube.com/watch?v=YX40hbAHx3s



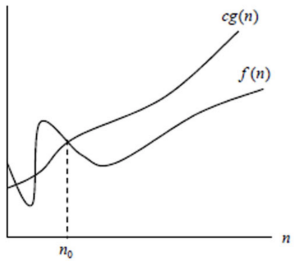
When considering a problem class, we want to know roughly how difficult the problem is. When considering an algorithm, we want to know roughly how fast the algorithm will run. These are questions that we can answer with complexity theory.

Binary: http://www.texample.net/tikz/examples/complement/

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*

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



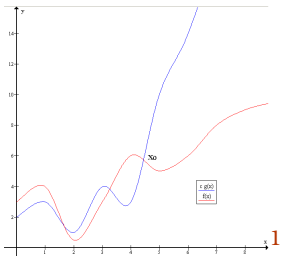


Figure 2.1: *Example of Big O notation:* $f(x) \in O(g(x))$ *as there exists* $c > 0$ *(e.g.* $c = 1$*) and* x_0 *(e.g.* $x_0 = 5$*) such that* $f(x) \leq cg(x)$ *whenever* $x \geq x_0$*.*

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 $7 \leq 7n^2$

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

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

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

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:

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

[1] Image borrowed from https://commons.wikimedia.org/wiki/File:Big-O-notation.png

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

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

$n^2 + n \leq 2n^3$ 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 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 is 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
	n	10	20
Exponential	$n^{\log n}$	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.*

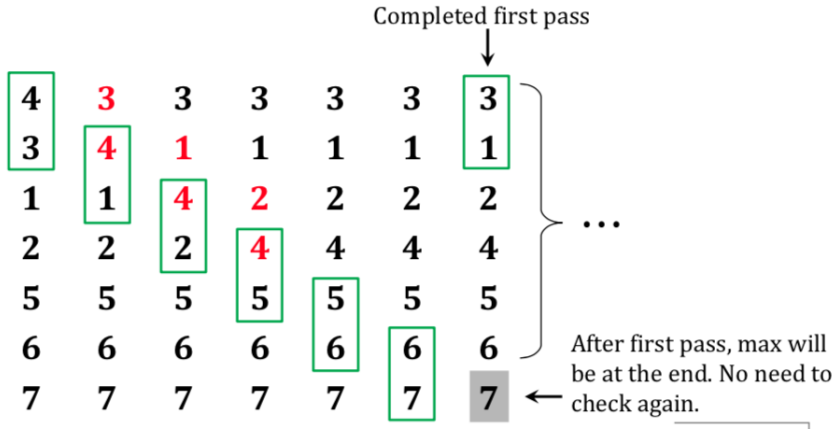
2.2.1 Sorting

https://en.wikipedia.org/wiki/Bubble_sort

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:

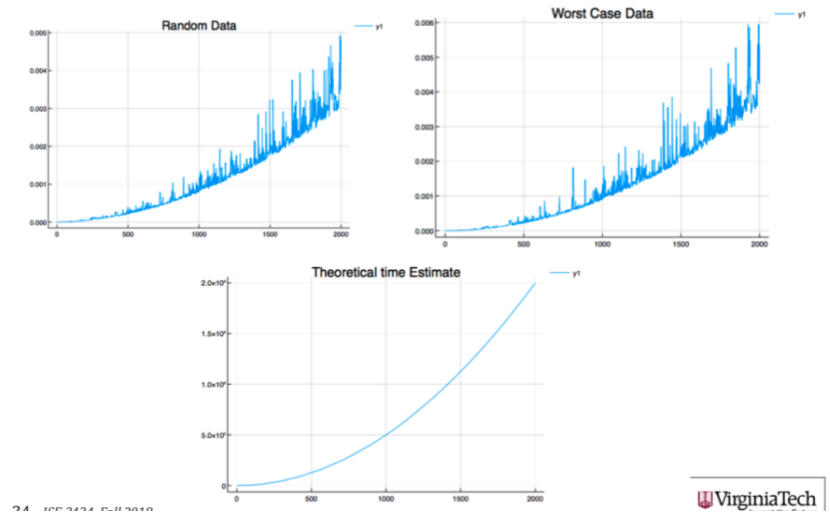
$$(n-1) + (n-2) + \cdots + 2 + 1 = \frac{n(n-1)}{2} = O(n^2)$$

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

We can also consider the best case and average case complexity:

Worst-case performance	$O(n^2)$ comparisons, $O(n^2)$ swaps
Best-case performance	$O(n)$ comparisons, $O(1)$ swaps
Average performance	$O(n^2)$ comparisons, $O(n^2)$ swaps
Worst-case space complexity	$O(1)$ auxiliary

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



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.

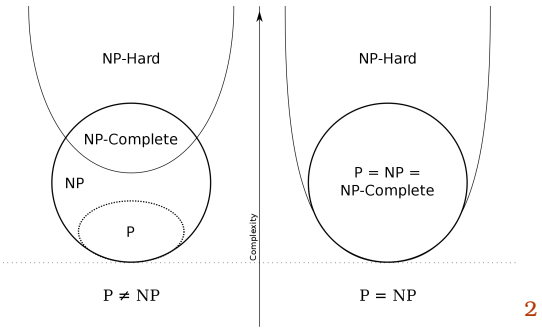


Figure 2.2: *Complexity class possibilities. Most academics agree that the case $P \neq NP$ is more likely.*

We will now discuss these classes more formally.

2.3.1 P

P is the class of polynomially solvable problems. P contains tall 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 by 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.

Example 4: Linear programming is in P. It can be solved by interior point methods in $O(n^{3.5}\phi)$ where ϕ 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]Figure from https://en.wikipedia.org/wiki/NP_(complexity)

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 $X_1, \ldots, X_n$ 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

$$\texttt{true} \wedge \texttt{true} = \texttt{true} \tag{2.3.1}$$
$$\texttt{true} \wedge \texttt{false} = \texttt{false} \tag{2.3.2}$$
$$\texttt{false} \wedge \texttt{false} = \texttt{false} \tag{2.3.3}$$
$$\texttt{false} \wedge \texttt{true} = \texttt{false} \tag{2.3.4}$$

$$\texttt{true} \vee \texttt{true} = \texttt{true} \tag{2.3.5}$$
$$\texttt{true} \vee \texttt{false} = \texttt{true} \tag{2.3.6}$$
$$\texttt{false} \vee \texttt{false} = \texttt{false} \tag{2.3.7}$$
$$\texttt{false} \vee \texttt{true} = \texttt{true} \tag{2.3.8}$$

$$\neg \texttt{ true } = \texttt{ false} \tag{2.3.9}$$
$$\neg \texttt{ false } = \texttt{ true} \tag{2.3.10}$$

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

$$(X_1 \wedge \neg X_2 \vee X_3) \wedge (X_1 \vee \neg X_3) \vee (X_1 \wedge X_2 \wedge X_3). \tag{2.3.11}$$

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

$$X_1 \vee \neg X_2 \vee X_3 \vee \neg X_4. \tag{2.3.12}$$

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

$$(X_1 \vee \neg X_2 \vee X_3 \vee \neg X_4) \wedge (X_1 \vee X_2 \vee X_3) \wedge (X_2 \vee \neg X_3 \vee \neg X_4 \vee X_5). \tag{2.3.13}$$

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

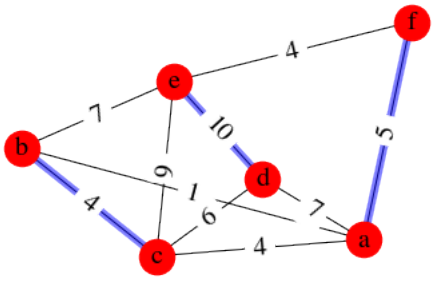
2.5 Matching Problem

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 were every vertex is connected to an edges in E'.*
A maximal matching *is a matching E' such that there is no matching E'' that strictly contains it.*

INCLUDE PICTURES OF MATCHINGS

Figure 2.3: *Two possible matchings. On the left, we have a perfect matchings (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.

[2]Figure from http://www.suhendry.net/blog/?p=1647.

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 $w(e_1) \geq w(e_2) \geq \cdots \geq w(e_m)$
3. For $i = 1, \ldots, m$
   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

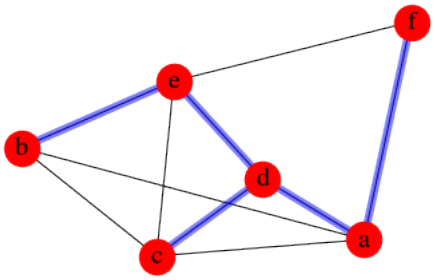
Theorem 6 ([**?**])**.** *The greedy algorithm finds a 2-approximation of the maximum weighted matching problem. That is,* $w(M_{greedy}) \geq \frac{1}{2}w(M^*)$.

2.5.2 Other algorithms to look at

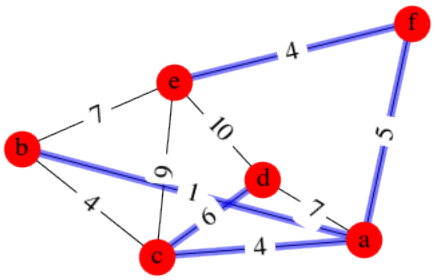
1. Improved Algorithm **[?]**
2. Blossom Algorithm https://en.wikipedia.org/wiki/Blossom_algorithm

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

https://en.wikipedia.org/wiki/Kruskal%27s_algorithm

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 $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_m)$
3. For $i = 1, \ldots, m$
   If $T \cup \{e_i\}$ is acyclic, then set $T \leftarrow T \cup \{e_i\}$
4. Return T

2.6.2 Prim's Algorithm

https://en.wikipedia.org/wiki/Prim%27s_algorithm
http://www.texample.net/tikz/examples/prims-algorithm/

2.7 Traveling Salesman Problem

https://www.youtube.com/watch?v=CPetTODX-FA https://www.youtube.com/watch?v=R_IfyticWKQ https://www.youtube.com/watch?v=2jncD54ryGs

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 Starting from any node, add the edge to the next closest node. Continue this process.

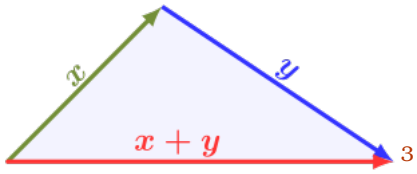
Nearest Neighbor:
Complexity: $O(n^2)$

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.

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. That is for side lengths x, y, z, we have

$$z \leq x + y$$



[3]

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 ever 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

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

[3] Figure borrowed from https://en.wikipedia.org/wiki/Triangle_inequality#/media/File:Vector-triangle-inequality.svg

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 https://en.wikipedia.org/wiki/Christofides_algorithm or https://resources.mpi-inf.mpg.de/conferences/adfocs-15/material/Ola-Lect1.pdf for more detail.

Chapter 3

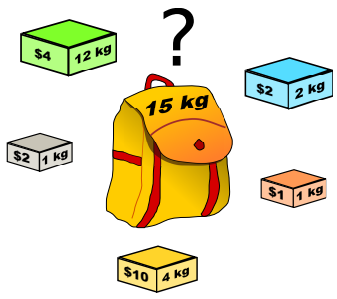
Integer Programming Formulations

3.1 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.

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 $c \in \mathbb{Q}^n$,

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



1

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

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?

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:

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

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

[1] https://en.wikipedia.org/wiki/Knapsack_problem

coefficients $c \in \mathbb{Q}^n$,

$$\max \quad c^\top x \tag{3.1.4}$$
$$\text{s.t.} \quad a^\top x = b \tag{3.1.5}$$
$$x \in \mathbb{Z}^n_+ \tag{3.1.6}$$

Example 7: Using pennies, nickels, dimes, and quarters, how can you minimize the number of coins you need to 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

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

3.2 Set Covering

Set Covering:

NP-Complete

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

The set cover problem can be modeled as

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

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.

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:

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

Example: Vertex Cover [Code: **??**]
Consider the graph with nodes

$$V = \text{["You", "Ginger", "Juan", "Jameis", "Bob", "Geoff", "Jane"]} \tag{3.2.3}$$

and edges

$$E = \text{[["You", "Ginger"], ["You", "Juan"], ["You", "Jameis"], ["You", "Bob"],} \tag{3.2.4}$$
$$\text{["You", "Jane"], ["You", "Geoff"], ["Ginger", "Jameis"], ["Jameis", "Bob"],} \tag{3.2.5}$$
$$\text{["Bob", "Jane"], ["Bob", "Geoff"], ["Geoff", "Jane"]]} \tag{3.2.6}$$

This problem can be modeled as

$$\begin{aligned} \min \quad & x_{\text{You}} + x_{\text{Ginger}} + x_{\text{Juan}} + x_{\text{Jameis}} + x_{\text{Bob}} + x_{\text{Geoff}} + x_{\text{Jane}} \\ \text{Subject to} \quad & x_{\text{You}} + x_{\text{Ginger}} \geq 1 \\ & x_{\text{You}} + x_{\text{Juan}} \geq 1 \\ & x_{\text{You}} + x_{\text{Jameis}} \geq 1 \\ & x_{\text{You}} + x_{\text{Bob}} \geq 1 \\ & x_{\text{You}} + x_{\text{Jane}} \geq 1 \\ & x_{\text{You}} + x_{\text{Geoff}} \geq 1 \\ & x_{\text{Ginger}} + x_{\text{Jameis}} \geq 1 \\ & x_{\text{Jameis}} + x_{\text{Bob}} \geq 1 \\ & x_{\text{Bob}} + x_{\text{Jane}} \geq 1 \\ & x_{\text{Bob}} + x_{\text{Geoff}} \geq 1 \\ & x_{\text{Geoff}} + x_{\text{Jane}} \geq 1 \\ & x_i \in \{0, 1\} \quad \forall i \in \{\text{You,Ginger,}\ldots\text{,Geoff,Jane}\} \end{aligned}$$

Example: Southwestern Airways[2] [Code: **??**]

Rework Example

Southwestern Airways needs to assign its crews to cover all its upcoming flights. We will focus on the problem of assigning three crews based in San Francisco to the flights listed in the first column of the below table. The other 12 columns show feasible sequences of flights for a crew. (The numbers in each column indicate the order of the flights: 1 = first stop, 2 = second stop, 3 = third stop...etc.) Exactly three of the sequences need to be chosen (one per crew) in such a way that every flight is covered. (It is permissible to have more than one crew on a flight, where extra crews would fly as passengers, but union contracts require that the extra crews would still need to be paid for their time as if they were working.) The cost of assigning a crew to a particular sequence of flights is given (in thousands of dollars) in the bottom row of the table. The objective is to minimize the total cost of the three crew assignments that cover all the flights.

Sets Let $I = \{1,\ldots,12\}$ denote flight sequences Let $F = \{1,\ldots,11\}$ denote the flights. For each flight i, let $F_i \subseteq I$ be the set of flight sequences that intersects flight i.

Variables Let $x_i = 1$ if flight sequence i is chosen and 0 otherwise

Model The model is to minimize cost while respecting that flights are covered. This can be modeled as

$$\begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & \sum_{j \in F_i} x_j \geq 1 && \text{for all } i = 1,\ldots,11 \\ & x_j \in \{0,1\} && \text{for all } j = 1,\ldots,12 \end{aligned}$$

Plugging in the data creates the following integer program:

$$
\begin{aligned}
\min \quad & 2x_1 + 3x_2 + 4x_3 + 6x_4 + 7x_5 + 5x_6 + 7x_7 + 8x_8 + 9x_9 + 9x_{10} + 8x_{11} + 9x_{12} \\
\text{Subject to} \quad & x_1 + x_4 + x_7 + x_{10} \geq 1 \\
& x_2 + x_5 + x_8 + x_{11} \geq 1 \\
& x_3 + x_6 + x_9 + x_{12} \geq 1 \\
& x_4 + x_7 + x_9 + x_{10} + x_{12} \geq 1 \\
& x_1 + x_6 + x_{10} + x_{11} \geq 1 \\
& x_4 + x_5 + x_9 \geq 1 \\
& x_7 + x_8 + x_{10} + x_{11} + x_{12} \geq 1 \\
& x_2 + x_4 + x_5 + x_9 \geq 1 \\
& x_5 + x_8 + x_{11} \geq 1 \\
& x_3 + x_7 + x_8 + x_{12} \geq 1 \\
& x_6 + x_9 + x_{10} + x_{11} + x_{12} \geq 1 \\
& x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} + x_{12} = 3 \\
& x_i \in \{0,1\} \quad \forall i \in \{1,2,\ldots,11,12\}
\end{aligned}
$$

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

$$
\begin{aligned}
\max \quad & c^\top x \\
\text{s.t..} \quad & Ax \geq \mathbb{1} \\
& x \in \{0,1\}^n.
\end{aligned}
\tag{3.2.7}
$$

Example: Vertex Cover with matrix [Code: **??**]

An alternate way to solve Example 8 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.

[2]This example was taken from Hillier and Lieberman

[2]This image table was taken from Hillier and Lieberman

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

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t..} \quad & Ax \geq b \\ & x \in \mathbb{Z}_+^n. \end{aligned} \tag{3.2.8}$$

Example 11: Nurse Scheduling— Adapted from "Applied Integer Programming". exercise 2.6 and 2.7

Nurses in large hospitals usually work 3 days a week. Daily demand for nurses is summarized in Table 2.4. Determine the number of nurses required per schedule type so that the total wage cost is minimized.

1. Use the numbers (not symbols) in the table to model this problem instance.
2. Solve this instance and submit the code.
3. Let S denote the set of schedule types. How many variables of each type (continuous, binary, integer) does your model have in terms of the number of schedules $|S|$?
4. If part-time nurses are hired at the rate of \$175/day, formulate the problem to minimize the total cost.
5. If part-time nurses must be accompanied by at least three full-time nurses, how would you formulate this constraint?

- Let
$$x_i = \text{The number of nurses to assign to schedule } i \;\forall\; i \in \{1,2,...,5\}.$$

The model is

$$\text{Minimize} \quad 525x_1 + 470x_2 + 550x_3 + 500x_4 + 425x_5$$

	Schedule Type					
Day	1	2	3	4	5	Nurses Required
Monday	X		X			20
Tuesday	X				X	25
Wednesday			X	X		26
Thursday		X			X	26
Friday		X		X	X	30
Saturday		X		X		30
Sunday	X		X			35
Weekly wage	525	470	550	500	425	

$$
\begin{aligned}
\text{s.t.} \quad & x_1 + x_3 \geq 20 \\
& x_1 + x_5 \geq 25 \\
& x_3 + x_4 \geq 26 \\
& x_2 + x_5 \geq 26 \\
& x_2 + x_4 + x_5 \geq 30 \\
& x_2 + x_4 \geq 30 \\
& x_1 + x_3 \geq 35 \\
& x_i \in \mathbb{Z}_+ \qquad\qquad \forall\, i \in \{1, 2, ..., 5\}
\end{aligned}
$$

The table tells us which schedule types contribute to each day's staff; i.e. Mondays are staffed entirely by nurses assigned to schedule types 1 and 3, Tuesdays by those assigned to type 1 and 5, and so on. At least 20 nurses are required to be on duty on Mondays; so the number of nurses assigned to schedules 1 and 3 should be at least 20. That is,

$$x_1 + x_3 \geq 20.$$

Extending this logic to each day gives us all seven constraints.

Alternately, by replacing the X's in the table with 1's we are left with a constraint matrix S. Let $\vec{x}^\top = [x_1, x_2, x_3, x_4, x_5]$ be the decision variables and $\vec{r}^\top = [20, 25, 26, 26, 30, 30, 35]$ describe the nurses required for each day, so that

$$S\ \vec{x} \geq \vec{r}$$

describes all seven constraints.

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}.

To Include picture

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

3.4 Facility Location

https://en.wikipedia.org/wiki/Facility_location_problem

3.4.1 Capacitated Facility Location

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

3.4.2 Uncapacitated Facility Location

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

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 $i = 1,\ldots,m$
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:

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

Consider the example given in the following table.

Project	$\mathbb{E}$[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

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

3.6 Network Flow

3.7 Transportation Problem

https://www.youtube.com/watch?v=Jr7LI-sUEmo

3.7.1 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.

3.7.2 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:

$$\text{Either} \quad a^\top x \leq b \quad \text{or} \quad c^\top x \leq d \quad \text{holds} \tag{3.7.1}$$

can be modeled as

$$\begin{aligned} a^\top x - b &\leq M_1 \delta \\ c^\top x - d &\leq M_2(1-\delta) \\ \delta &\in \{0,1\}, \end{aligned} \tag{3.7.2}$$

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 12: 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 an at most $M_2 = 20$ cars that could be available.

This constraint can be modeled as

$$\begin{aligned} x - 2 &\leq 5\delta \\ y - 10 &\leq 20(1-\delta) \\ \delta &\in \{0,1\}, \end{aligned} \tag{3.7.3}$$

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

$$\text{If} \quad \delta = 1, \text{ then} \quad a^\top x \leq b. \tag{3.7.4}$$

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

$$\begin{aligned} a^\top x - b &\leq M(1-\delta) \\ \delta &\in \{0,1\} \end{aligned} \tag{3.7.5}$$

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 ϵ 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

$$\text{If} \quad a^\top x \leq b \quad \text{then} \quad \delta = 1 \tag{3.7.6}$$

can be modeled as

$$a^\top x - b \geq \epsilon(1-\delta) + m\delta. \tag{3.7.7}$$

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

$$\text{If } \delta = 0, \text{ then } a^\top x - b > 0. \tag{3.7.8}$$

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

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

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 ϵ, 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 ϵ 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

- ϵ 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:

[3]This table was taken from notes by Laurent Lassard.

Logic statement	Constraint
if $z = 0$ then $a^T x \leq b$	$a^T x - b \leq Mz$
if $z = 0$ then $a^T x \geq b$	$a^T x - b \geq mz$
if $z = 1$ then $a^T x \leq b$	$a^T x - b \leq M(1 - z)$
if $z = 1$ then $a^T x \geq b$	$a^T x - b \geq m(1 - z)$
if $a^T x \leq b$ then $z = 1$	$a^T x - b \geq mz + \varepsilon(1 - z)$
if $a^T x \geq b$ then $z = 1$	$a^T x - b \leq Mz - \varepsilon(1 - z)$
if $a^T x \leq b$ then $z = 0$	$a^T x - b \geq m(1 - z) + \varepsilon z$
if $a^T x \geq b$ then $z = 0$	$a^T x - b \leq M(1 - z) - \varepsilon z$

Where M and m are upper and lower bounds on $a^T x - b$.

[3]

Table 3.1: *If/then models with a constraint and a binary variable.*

3.7.3 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

$$0 \leq x \leq u, \quad x \in \mathbb{Z}, \tag{3.7.9}$$

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

$$\begin{aligned} x &= \sum_{i=1}^{u} i z_i = z_1 + 2z_2 + \cdots + uz_u \\ 1 &\geq \sum_{i=1}^{u} z_i = z_1 + z_2 + \cdots + z_u \\ z_i &\in \{0, 1\} \quad \text{for } i = 1, \ldots, u \end{aligned} \tag{3.7.10}$$

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

$$0 \leq x \leq u, \quad x \in \mathbb{Z}, \tag{3.7.11}$$

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

$$\begin{aligned} x &= \sum_{i=0}^{\log(\lfloor u \rfloor)+1} 2^i z_i = z_0 + 2z_1 + 4z_2 + 8z_3 + \cdots + 2^{\log(\lfloor u \rfloor)+1} z_{\log(\lfloor u \rfloor)+1} \\ z_i &\in \{0,1\} \quad \text{for } i = 1, \ldots, \log(\lfloor u \rfloor) + 1 \end{aligned} \tag{3.7.12}$$

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.

3.7.4 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:

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

3.7.5 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:

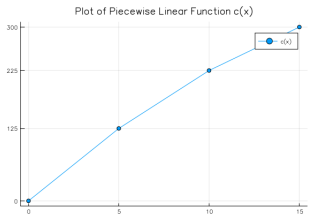
$$\begin{aligned} \text{maximize} \quad & 3x_1 + 4x_2 + x_3 + 5x_4 \\ \text{subject to} \quad & 0 \le x_i \le 5 \\ & \text{at most two of the } x_i \text{ can be nonzero} \\ & \text{and the nonzero } x_i \text{ must be consecutive} \end{aligned}$$

3.7.6 Piecewise linear functions with SOS2 constraint

Example: Piecewise Linear Function [Code: **??**]

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

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



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 $c(a)$.

$$
\begin{aligned}
\min \quad & 0 \\
\text{Subject to} \quad & x - 5z_2 - 10z_3 - 15z_4 = 0 \\
& y - 125z_2 - 225z_3 - 300z_4 = 0 \\
& z_1 + z_2 + z_3 + z_4 = 1 \\
& z_1 - w_1 \leq 0 \\
& z_2 - w_2 \leq 0 \\
& z_3 - w_3 \leq 0 \\
& z_4 - w_4 \leq 0 \\
& SOS2 : \{w_1, w_2, w_3, w_4\} \\
& 0 \leq z_i \leq 1 \quad \forall i \in \{1,2,3,4\} \\
& w_i \in \{0,1\} \quad \forall i \in \{1,2,3,4\} \\
& x = a
\end{aligned}
$$

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 15:

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

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

$$
\begin{aligned}
\max \quad & 12X_{1,1} + 12X_{2,1} + 14X_{1,2} + 14X_{2,2} - y \\
\text{Subject to} \quad & x - 5z_2 - 10z_3 - 15z_4 = 0 \\
& y - 125z_2 - 225z_3 - 300z_4 = 0 \\
& z_1 + z_2 + z_3 + z_4 = 1 \\
& z_1 - w_1 \leq 0 \\
& z_2 - w_2 \leq 0 \\
& z_3 - w_3 \leq 0 \\
& z_4 - w_4 \leq 0 \\
& X_{1,1} + X_{1,2} - x \leq 5 \\
& X_{2,1} + X_{2,2} \leq 10 \\
& 0.5X_{1,1} - 0.5X_{2,1} \geq 0 \\
& 0.4X_{1,2} - 0.6X_{2,2} \geq 0 \\
& SOS2 : \{w_1, w_2, w_3, w_4\} \\
& X_{i,j} \geq 0 \quad \forall i \in \{1,2\}, j \in \{1,2\} \\
& 0 \leq z_i \leq 1 \quad \forall i \in \{1,2,3,4\} \\
& w_i \in \{0,1\} \quad \forall i \in \{1,2,3,4\} \\
& 0 \leq x \leq 15 \\
& y
\end{aligned}
$$

3.7.7 Maximizing a minimum

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

Consider the problem

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

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

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

3.7.8 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,

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

3.8 Notes from AIMMS modeling book.

3.8.1 Guidelines for Integer Programming Modeling

Practical guidelines for solving difficult MILPs http://inside.mines.edu/~anewman/MIP_practice120212.pdf

3.8.2 Linear Programming Modeling

https://download.aimms.com/aimms/download/manuals/AIMMS3OM_LinearProgrammingTricks.pdf

3.8.3 From AIMMS

https://download.aimms.com/aimms/download/manuals/AIMMS3OM_FormulatingOptimizationModels.pdf

Linear Programming

Practical guidelines for solving difficult linear programs https://pdfs.semanticscholar.org/b01f/ad44c20c372fdda95cbfb980c0d37302de07.pdf

3.8.4 Further Topics

Precedence Constraints

https://or.stackexchange.com/questions/1319/best-model-for-precedence-constraints-within-schedul

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 $L_1, \ldots, L_k$. 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.

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

Example 17: 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 i and $z_j = 0$ if we do not use pipe j, for $j = 1, \ldots, N$.
- Let x_{ij} be the number of cuts of length L_i in pipe j that we use.

Then we have the following model

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

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

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

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

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

$$z_1 \geq z_2 \geq z_3 \geq \cdots \geq z_N. \tag{4.1.3}$$

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

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

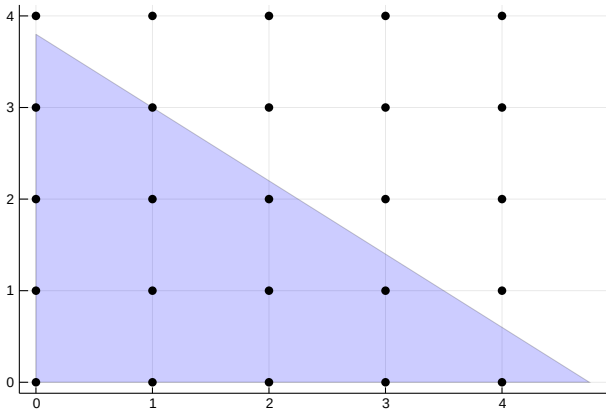
In our running example, we have

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

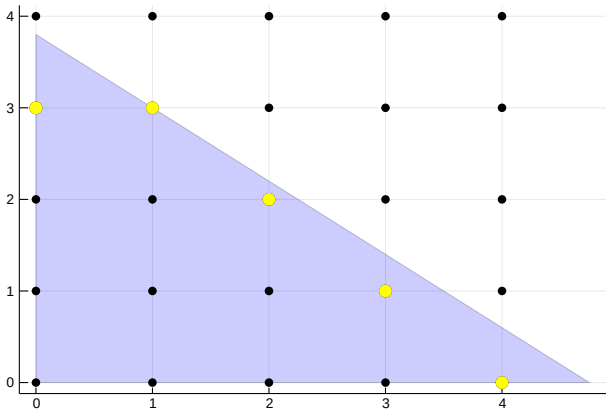
For visualization purposes, consider the patterns where $a_3 = 0$. That is, only patterns with cuts of length 4m or 5m. All patterns of this type are represented by an integer point in the polytope

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

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 patten $[4,0,0]$ (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 $i = 1, \ldots, k$. 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

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

Example 18: Pricing Problem(knapsack problem!) After choosing the initial columns ....
we can find the objective function....

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

4.1.3 Cutting Stock - Multiple widths

Gurobi has as excellent demonstration application to look at: https://www.gurobi.com/cutting-stock-problem-with-multiple-master-rolls/
https://demos.gurobi.com/cutstock/.
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:
https://download.aimms.com/aimms/download/manuals/AIMMS3OM_CuttingStock.pdf

4.2 Spanning Trees

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

4.3 Traveling Salesman Problem

See http://www.math.uwaterloo.ca/tsp/index.html for excellent material on the TSP.
See also this chapter https://www.math.uwaterloo.ca/~bico/papers/comp_chapter1.pdf.

Also, watch this excellent talk by Bill Cook "Postcards from the Edge of Possibility": https://m.youtube.com/watch?v=5VjphFYQKj8

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 $i \to j$.

A *tour* 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 city$i \to$ city$j$.

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

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

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:

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

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

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

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

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

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

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

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

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

$$u_2 < u_5 < u_3 < u_2, \tag{4.3.6}$$

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:

$$\min \sum_{i,j \in N} c_{ij} x_{ij} \tag{4.3.7}$$

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

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

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

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

$$u_1 = 1 \tag{4.3.12}$$

$$2 \leq u_i \leq n \qquad \text{for all } i \in N, i \neq 1 \tag{4.3.13}$$

$$u_i \in \mathbb{Z} \qquad \text{for all } i \in N \tag{4.3.14}$$

$$x_{ij} \in \{0,1\} \qquad \text{for all } i, j \in N \tag{4.3.15}$$

Example 19:
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

$$\min \quad x_{1,2} + 2x_{1,3} + 3x_{1,4} + x_{2,1} + x_{2,3} + 2x_{2,4} +$$
$$2x_{3,1} + x_{3,2} + 4x_{3,4} + 3x_{4,1} + 2x_{4,2} + 4x_{4,3}$$

Subject to

$$x_{1,1} + x_{1,2} + x_{1,3} + x_{1,4} = 1 \qquad \text{outgoing from node}$$
$$x_{2,1} + x_{2,2} + x_{2,3} + x_{2,4} = 1 \qquad \text{outgoing from node}$$
$$x_{3,1} + x_{3,2} + x_{3,3} + x_{3,4} = 1 \qquad \text{outgoing from node}$$
$$x_{4,1} + x_{4,2} + x_{4,3} + x_{4,4} = 1 \qquad \text{outgoing from node}$$

$$x_{1,1} + x_{2,1} + x_{3,1} + x_{4,1} = 1 \qquad \text{incoming to node}$$
$$x_{1,2} + x_{2,2} + x_{3,2} + x_{4,2} = 1 \qquad \text{incoming to node}$$
$$x_{1,3} + x_{2,3} + x_{3,3} + x_{4,3} = 1 \qquad \text{incoming to node}$$
$$x_{1,4} + x_{2,4} + x_{3,4} + x_{4,4} = 1 \qquad \text{incoming to node}$$

$$x_{1,1} = 0 \qquad \text{No self loop with node}$$
$$x_{2,2} = 0 \qquad \text{No self loop with node}$$
$$x_{3,3} = 0 \qquad \text{No self loop with node}$$
$$x_{4,4} = 0 \qquad \text{No self loop with node}$$

$$u_1 = 1 \qquad \text{Start at node}$$
$$2 \leq u_i \leq 4, \quad \forall i \in \{2,3,4\}$$
$$u_2 + 1 \leq u_3 + 4(1 - x_{2,3})$$
$$u_2 + 1 \leq u_4 + 4(1 - x_{2,4}) \leq 3$$
$$u_3 + 1 \leq u_2 + 4(1 - x_{3,2}) \leq 3$$
$$u_3 + 1 \leq u_4 + 4(1 - x_{3,4}) \leq 3$$
$$u_4 + 1 \leq u_2 + 4(1 - x_{4,2}) \leq 3$$
$$u_4 + 1 \leq u_3 + 4(1 - x_{4,3}) \leq 3$$
$$x_{i,j} \in \{0,1\} \quad \forall i \in \{1,2,3,4\}, j \in \{1,2,3,4\}$$
$$u_i \in \mathbb{Z}, \quad \forall i \in \{1,2,3,4\}$$

Example 20: 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

$$2 \to 5 \to 3 \to 2. \tag{4.3.16}$$

We can prevent this subtour by adding the constraint

$$x_{25} + x_{53} + x_{32} \leq 2 \tag{4.3.17}$$

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*

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

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:

$$\min \sum_{i,j\in N} c_{ij}x_{ij} \tag{4.3.19}$$

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

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

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

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

$$x_{ij} \in \{0,1\} \qquad \text{for all } i,j \in N \tag{4.3.24}$$

Example 21: 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

$$1\,\text{min} \quad x_{1,2} + 2x_{1,3} + 3x_{1,4} + x_{2,1} + x_{2,3} + 2x_{2,4} + 2x_{3,1} + x_{3,2} + 4x_{3,4} + 3x_{4,1} + 2x_{4,2} + 4x_{4,3}$$

Subject to

$x_{1,1} + x_{1,2} + x_{1,3} + x_{1,4} = 1$	outgoing from node 1
$x_{2,1} + x_{2,2} + x_{2,3} + x_{2,4} = 1$	outgoing from node 2
$x_{3,1} + x_{3,2} + x_{3,3} + x_{3,4} = 1$	outgoing from node 3
$x_{4,1} + x_{4,2} + x_{4,3} + x_{4,4} = 1$	outgoing from node 4
$x_{1,1} + x_{2,1} + x_{3,1} + x_{4,1} = 1$	incoming to node 1
$x_{1,2} + x_{2,2} + x_{3,2} + x_{4,2} = 1$	incoming to node 2
$x_{1,3} + x_{2,3} + x_{3,3} + x_{4,3} = 1$	incoming to node 3
$x_{1,4} + x_{2,4} + x_{3,4} + x_{4,4} = 1$	incoming to node 4
$x_{1,1} = 0$	No self loop with node 1
$x_{2,2} = 0$	No self loop with node 2
$x_{3,3} = 0$	No self loop with node 3
$x_{4,4} = 0$	No self loop with node 4
$x_{1,2} + x_{2,1} \leq 1$	$S = [(1,2),(2,1)]$
$x_{1,3} + x_{3,1} \leq 1$	$S = [(1,3),(3,1)]$
$x_{1,4} + x_{4,1} \leq 1$	$S = [(1,4),(4,1)]$
$x_{2,3} + x_{3,2} \leq 1$	$S = [(2,3),(3,2)]$
$x_{2,4} + x_{4,2} \leq 1$	$S = [(2,4),(4,2)]$
$x_{3,4} + x_{4,3} \leq 1$	$S = [(3,4),(4,3)]$
$x_{2,1} + x_{1,3} + x_{3,2} \leq 2$	$S = [(2,1),(1,3),(3,2)]$
$x_{1,2} + x_{2,3} + x_{3,1} \leq 2$	$S = [(1,2),(2,3),(3,1)]$
$x_{3,1} + x_{1,4} + x_{4,3} \leq 2$	$S = [(3,1),(1,4),(4,3))]$
$x_{1,3} + x_{3,4} + x_{4,1} \leq 2$	$S = [(1,3),(3,4),(4,1)]$
$x_{2,1} + x_{1,4} + x_{4,2} \leq 2$	$S = [(2,1),(1,4),(4,2)]$
$x_{1,2} + x_{2,4} + x_{4,1} \leq 2$	$S = [(1,2),(2,4),(4,1)]$
$x_{3,2} + x_{2,4} + x_{4,3} \leq 2$	$S = [(3,2),(2,4),(4,3)]$
$x_{2,3} + x_{3,4} + x_{4,2} \leq 2$	$S = [(2,3),(3,4),(4,2)]$

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

Example 22:
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 Google maps data

https://www.geeksforgeeks.org/python-calculate-distance-duration-two-places-using-google-distan

4.5 Literature

Gilmore-Gomory Cutting Stock [**?**]
http://www.optimization-online.org/DB_HTML/2018/06/6648.html
http://www.optimization-online.org/DB_HTML/2018/06/6670.html
http://www.optimization-online.org/DB_FILE/2017/11/6331.pdf

Chapter 5

Algorithms to Solve Integer Programs

5.1 LP to solve IP

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

Integer Program:		Linear Relaxation:	
max	$z_{IP} = c^\top x$	max	$z_{LP} = c^\top x$
	$Ax \leq b$		$Ax \leq b$
	$x \in \mathbb{Z}^n$		$x \in \mathbb{R}^n$

Theorem 14. *It always holds that*

$$z^*_{IP} \leq z^*_{LP}. \tag{5.1.1}$$

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

$$x^*_{LP} = x^*_{IP} \quad \textit{and } z^*_{LP} = z^*_{IP}. \tag{5.1.2}$$

Example 23:
Consider the problem

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

5.1.1 Rounding LP Solution can be bad!

Consider the two variable knapsack problem

$$\max 3x_1 + 100x_2 \tag{5.1.3}$$

$$x_1 + 10x_2 \leq 10 \tag{5.1.4}$$

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

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 algorothm.

https://www.youtube.com/watch?time_continue=424&v=m1p-eWxrt6g

https://www.geeksforgeeks.org/fractional-knapsack-problem/

5.2 Branch and Bound

See http://web.tecnico.ulisboa.pt/mcasquilho/compute/_linpro/TaylorB_module_c.pdf for some nice notes on branch and bound.

5.2.1 Algorithm

Algorithm 2 Branch and Bound - Maximization

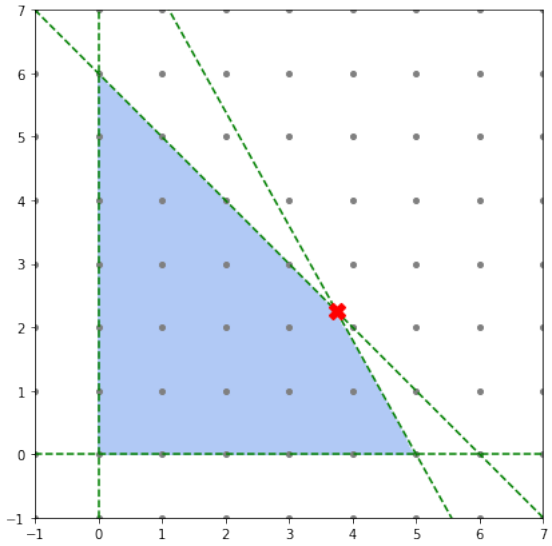
Input: Integer Linear Problem with max objective
Output: Exact Optimal Solution x^*

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.

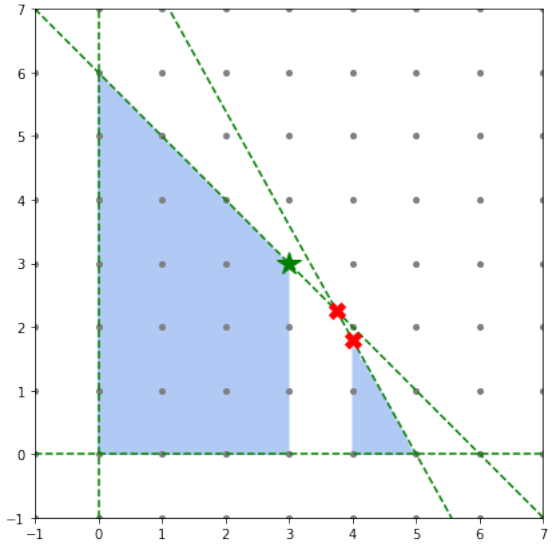
5.2.2 General Branching

Example 24: See Example 9 in Chapter 9 of the textbook (Winston - Operations Research Applications and Algorithms).

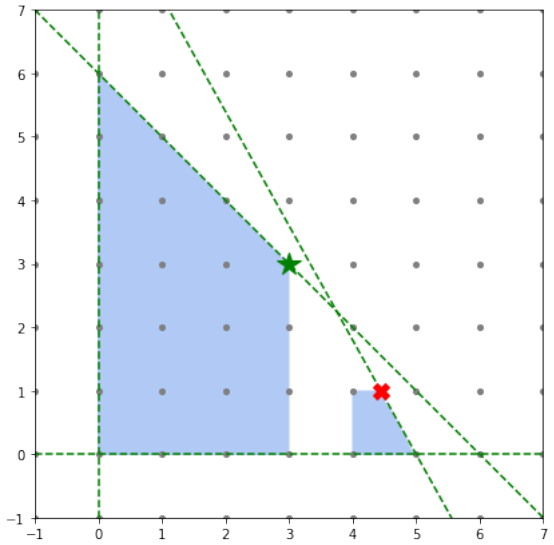
$x = [3.75, 2.25]$, $obj = 41.25$



$x = [3, 3]$ $obj = 39.0$
$x = [4, 1.8]$,, $obj = 41.0$

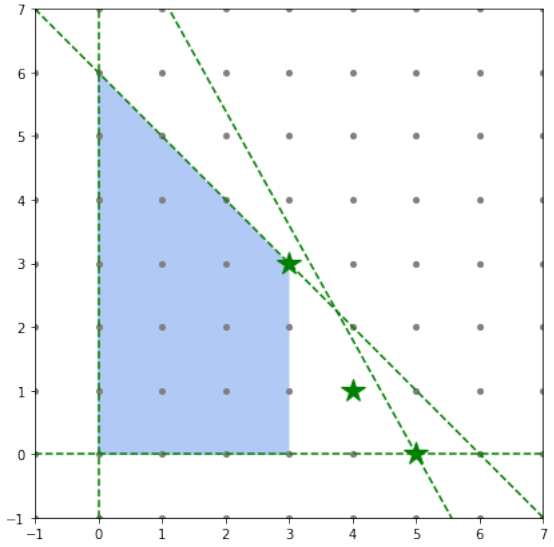


$x = [3, 3]$, $obj = 39.0$
$x = [4.44, 1]$ $obj = 40.55$.
Infeasible Region



$x = [3,3]$, $obj = 39.0$
$x = [4,1]$, $obj = 37.0$
$x = [5,0]$, $obj = 40.0$

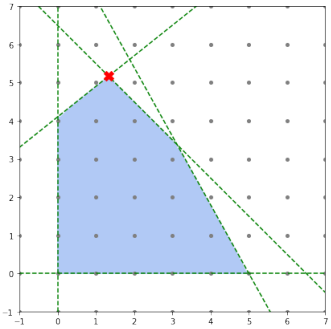
Infeasible Region



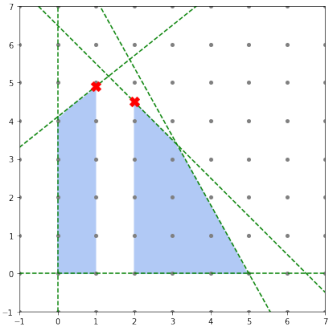
Example 25: Consider the two variable example with

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

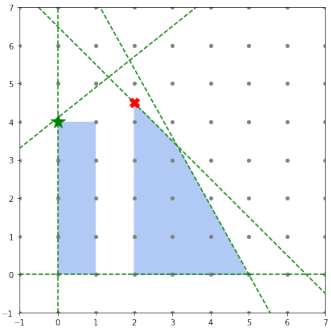
$x = [1.33, 5.167] obj = 16.664$



$x = [1, 4.9] obj = 16.5998$
$x = [2, 4.5] obj = 12.0$



Infeasible Region
$x = [0.4.] obj = 16.0$
$x = [2.4.5] obj = 12.0$



5.2.3 Knapsack Problem and 0/1 branching

Consider the problem

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

What is the optimal solution if we remove the binary constraints?

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

How do I find the solution to this problem?

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

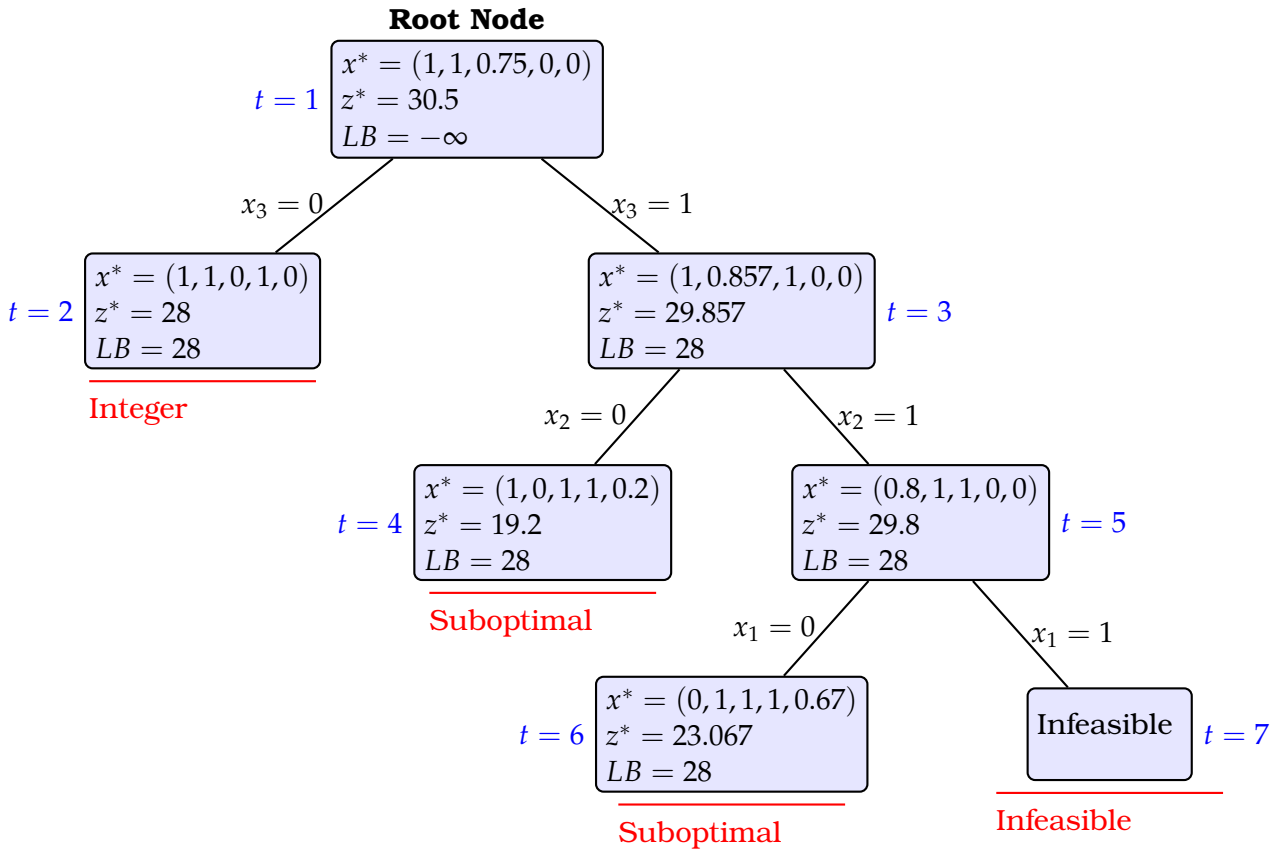
How do I find the solution to this problem?

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.

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



5.2.4 Traveling Salesman Problem solution via Branching

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 5.1. 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.

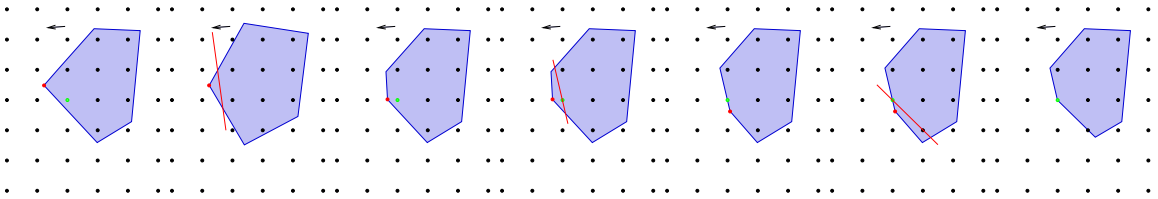


Figure 5.1: *The cutting plane procedure.*

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

$$a_1 x_1 + \cdots + a_n x_n \leq d \tag{5.3.1}$$

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

$$\lfloor a_1 \rfloor x_1 + \cdots + \lfloor a_n \rfloor x_n \leq \lfloor d \rfloor \tag{5.3.2}$$

is valid for the integer points in P, that is, it is valid for the set $P \cap \mathbb{Z}^n$. Equation (5.3.2) is called a Chvátal Cut.

We will illustrate this idea with an example.

Example 26: Recall example 7. The model was

Model

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

From the equality constraint we can derive several inequalities.

1. Divide by 25 and round down both sides:

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

2. Divide by 10 and round down both sides:

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

3. Divide by 5 and round down both sides:

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

4. Multiply by 0.12 and round down both sides:

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

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

New Model

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

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

Gomory cuts are a type of Chvátal cut that is derived from the simplex tableau. Specifically, suppose that

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

is an equation in the optimal simplex tableau.

Gomory Cut:
The Gomory cut corresponding to the tableau row (5.3.3) is

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

We will solve the following problem using only Gomory Cuts.

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

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

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

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

Initial Basis

Basis	RHS	x_1	x_2	s_1	s_2
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

$\vdots$ $\qquad\qquad$ $\vdots$

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 27: (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

$$x_1 - 0.1s_1 + -0.6s_2 = 1.5. \tag{5.3.6}$$

Applying the Gomory Cut (5.3.4), we have the inequality

$$0.9s_1 + 0.4s_2 \geq 0.5. \tag{5.3.7}$$

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 28: (Gomory Cut in Original Space)

The Gomory Cut (5.3.7) can be rewritten in the original variables using the equations from (5.3.5). That is, we can use the equations

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

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

$$-3.2x_1 + 5.8x_2 \leq 9.2. \tag{5.3.9}$$

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

Example 29: (Gomory cuts plus new tableau) Now we add the slack variable $s_3 \geq 0$ to make the equation

$$0.9s_1 + 0.4s_2 - s_3 = 0.5. \tag{5.3.10}$$

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

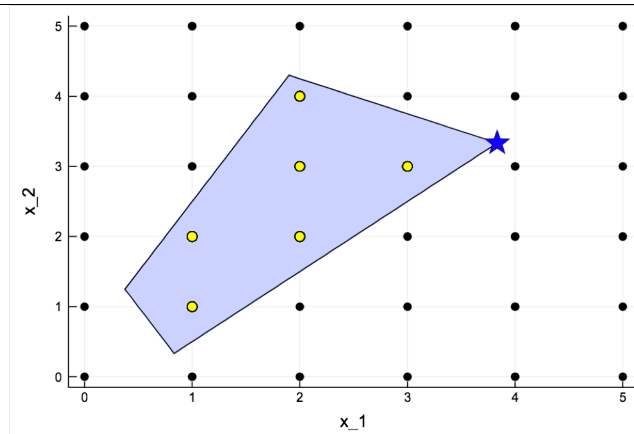
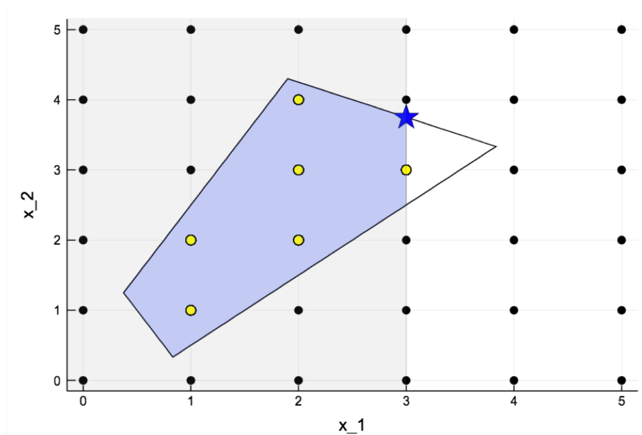
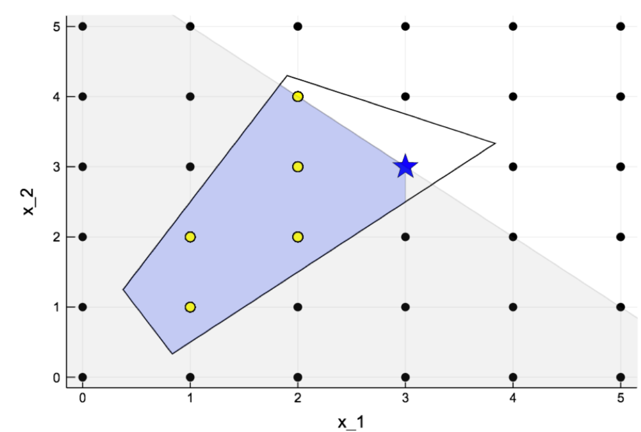
5.5 Lagrangian Relaxation for Branch and Bound

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

See [**?**] for a description of this.

For a great tutorial, see this: https://my.eng.utah.edu/~kalla/phy_des/lagrange-relax-tutorial-fisher.pdf

5.6 Literature

Model	**LP Solution**
$\begin{array}{rl} \max & x_1 + x_2 \\ \text{Subject to} & -2x_1 + x_2 \leq 0.5 \\ & x_1 + 2x_2 \leq 10.5 \\ & x_1 - x_2 \leq 0.5 \\ & -2x_1 - x_2 \leq -2 \\ & x_i \quad \forall i \in \{1,2\} \end{array}$	
$\begin{array}{rl} \max & x_1 + x_2 \\ \text{Subject to} & -2x_1 + x_2 \leq 0.5 \\ & x_1 + 2x_2 \leq 10.5 \\ & x_1 - x_2 \leq 0.5 \\ & -2x_1 - x_2 \leq -2 \\ & x_i \quad \forall i \in \{1,2\} \end{array}$	
$\begin{array}{rl} \max & x_1 + x_2 \\ \text{Subject to} & -2x_1 + x_2 \leq 0.5 \\ & x_1 + 2x_2 \leq 10.5 \\ & x_1 - x_2 \leq 0.5 \\ & -2x_1 - x_2 \leq -2 \\ & x_i \quad \forall i \in \{1,2\} \end{array}$	

Chapter 6

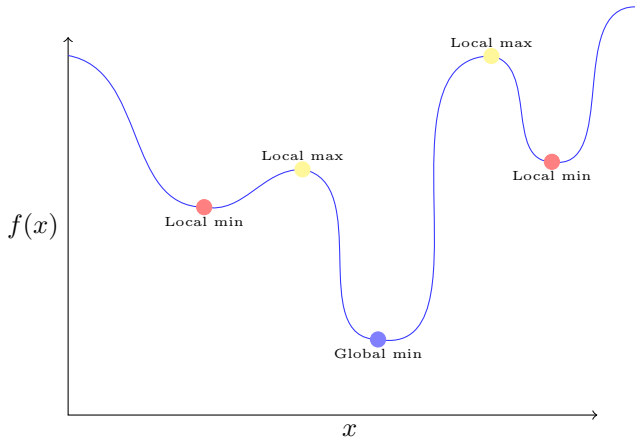
Non-linear Programming (NLP)

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

- **objective function** $f : \mathbb{R}^n \to \mathbb{R}$
- 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 $f : S \to \mathbb{R}$ is continuous. Then the problem $\min\{f(x) : x \in S\}$ attains its minimum.*

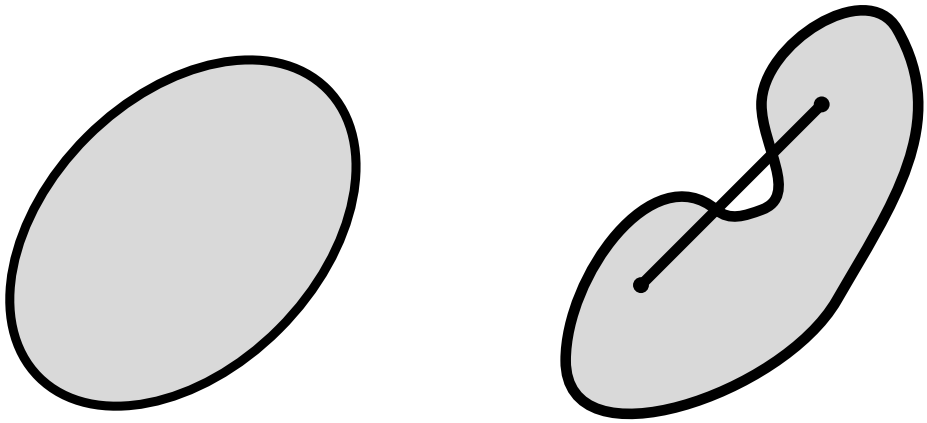
Definition 17. *A* critical point *is a point $\bar{x}$ where $\nabla f(\bar{x}) = 0$.*

Theorem 18. *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ 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 $\lambda \in [0,1]$.*



(a) *Convex Set* ***(b)*** *Non-convex set*

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 $x, y \in S$*

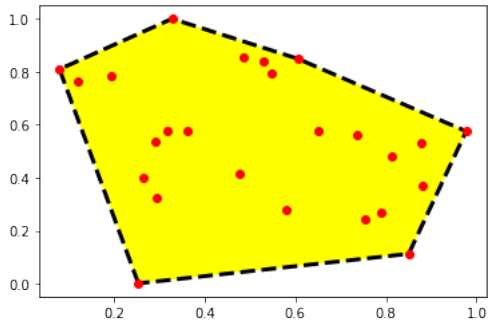
$$\lambda x + (1 - \lambda y) \in S \quad \textit{for all } \lambda \in [0,1].$$

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 $x^1, \ldots, x^k \in S$ such that $x \in \text{conv}(\{x^1, \ldots, x^k\})$ 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.

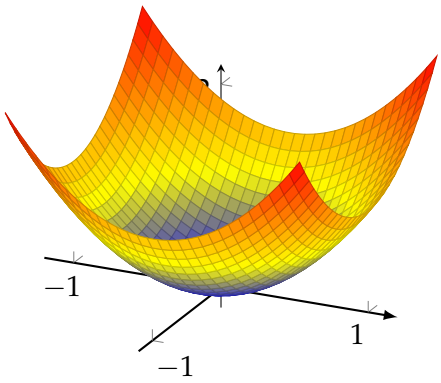
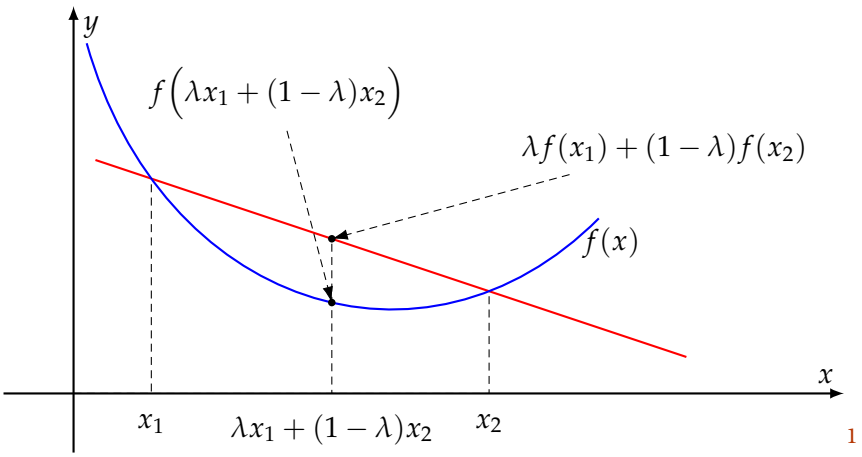


Figure 6.2: *Convex Function $f(x,y) = x^2 + y^2$.*

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

$$\lambda f(x) + (1-\lambda) f(y) \geq f(\lambda x + (1-\lambda) y). \tag{6.2.1}$$

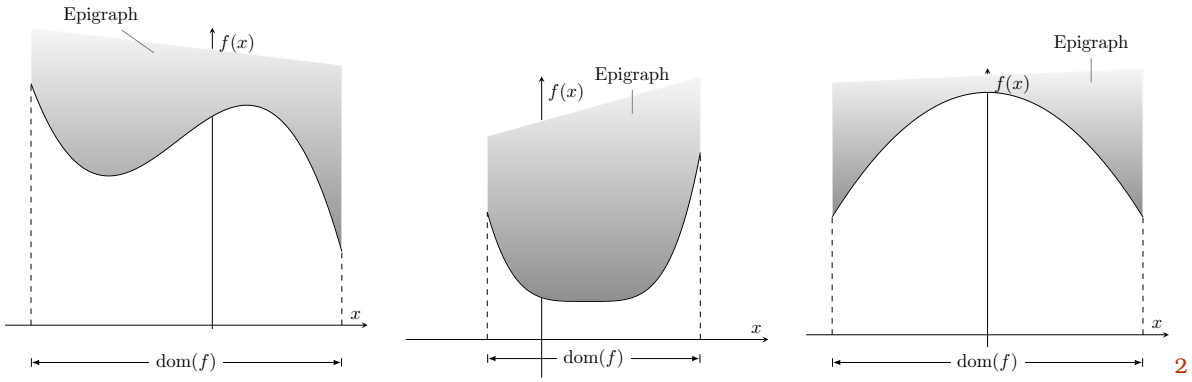


[1]

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.*

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



[2]

Example 30: Examples of Convex functions
Some examples are

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

[1] https://tex.stackexchange.com/questions/394923/how-one-can-draw-a-convex-function
[2] https://tex.stackexchange.com/questions/261501/function-epigraph-possibly-using-fillbetween

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

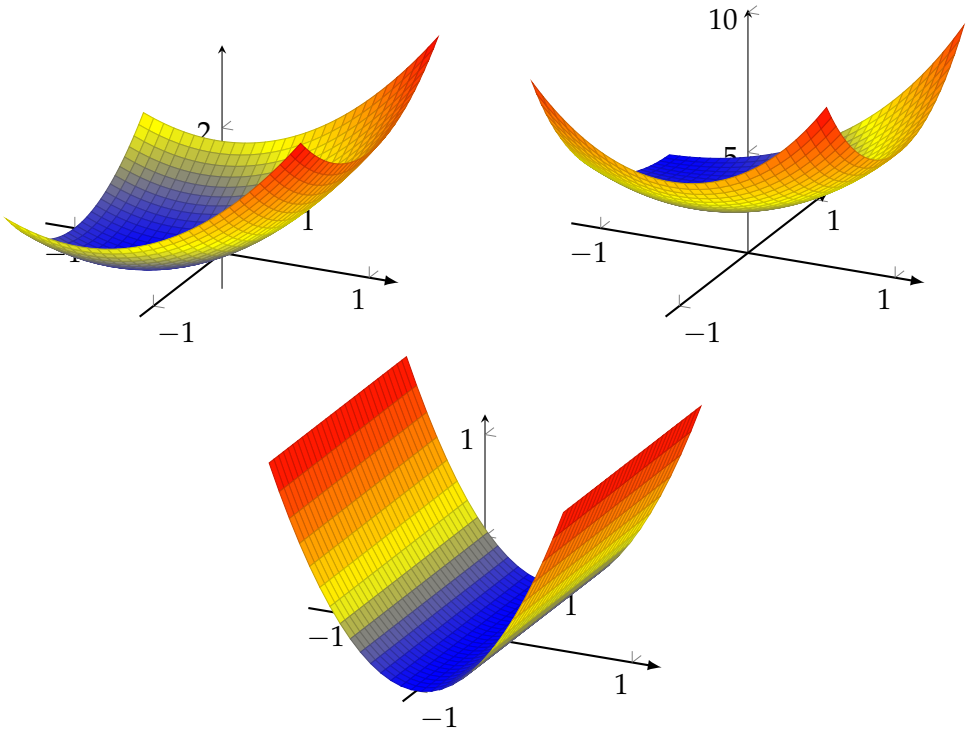
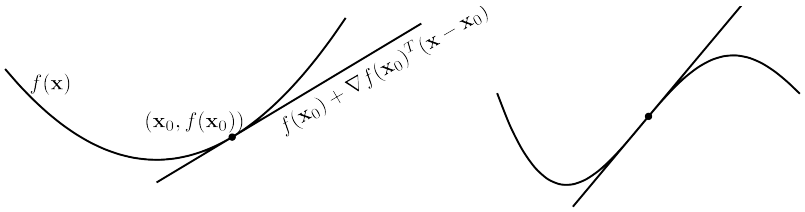


Figure 6.3: *Convex Functions* $f(x, y) = x^2 + y^2 + x$, $f(x, y) = e^{x+y} + e^{x-y} + e^{-x-y}$, *and* $f(x, y) = x^2$.

6.2.1 Proving Convexity - Characterizations

Theorem 27 (Convexity: First order characterization - linear underestimates)**.** *Suppose that* $f : \mathbb{R}^n \to \mathbb{R}$ *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,*

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



[3]

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

- *Suppose $f : \mathbb{R} \to \mathbb{R}$ is twice differentiable. Then f is convex if and only if $f''(x) \geq 0$ for all $x \in \mathbb{R}$.*
- *Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is twice differentiable. Then f is convex if and only if $\nabla^2 f(x) \preceq 0$ 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 αf 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 $i = 1, \ldots, t$, then $f(x) = \max_{i=1,\ldots,t} f_i(x)$ 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.

[3] https://machinelearningcoban.com/2017/03/12/convexity/

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.

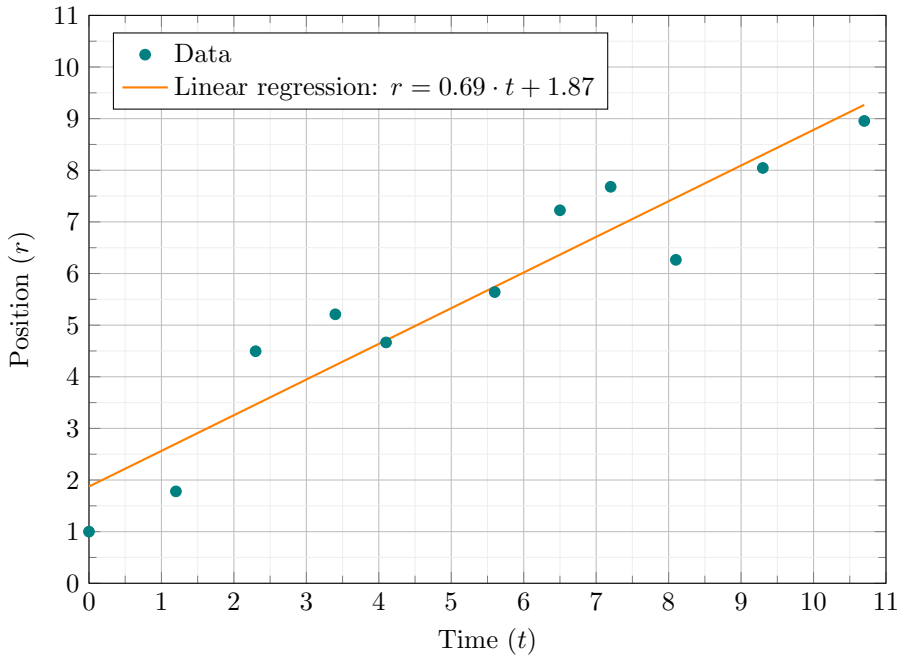
Example 1: $g(x) = x^4$ is convex, $h(x) = e^x$ is convex and non-decreasing. Therefore, $f(x) = e^{x^4}$ is also convex.

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 $x^1, \ldots, x^N \in \mathbb{R}^d$ 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 $x =$ (time) and the output could be $y =$ position.



[4]

As is standard, we choose the error (or "loss") from each data point as the squared

[4] https://latexdraw.com/linear-regression-in-latex-using-tikz/

error. Hence, we can model this as the optimization problem:

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

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* (6.3.1) *is*

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

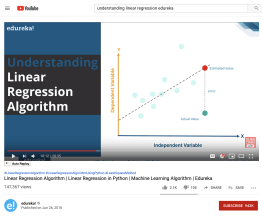
where

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

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

To be completed.... □

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



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 he 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

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

There are three versions to consider:

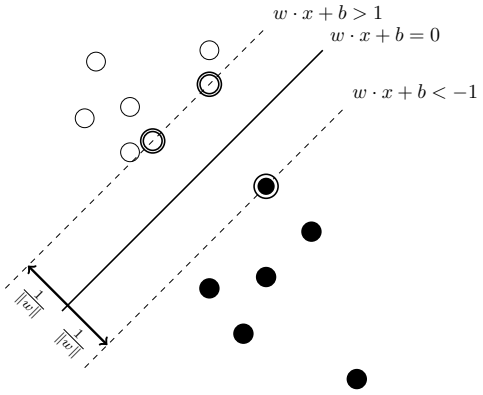
Feasible separation

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

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

SVM

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



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

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

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

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

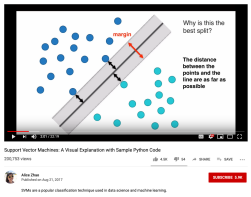
This creates the following optimization problem

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

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

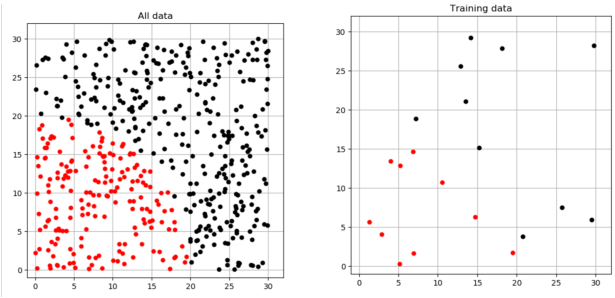
https://www.youtube.com/watch?time_continue=6&v=N1vOgolbjSc



Suppose for instance you are given data $x^1, \ldots, x^N \in \mathbb{R}^2$ (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

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

Example 31:



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

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

In this way, we convert the data $x^1, \ldots, x^N$ into data $X^1, \ldots, X^N$ 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

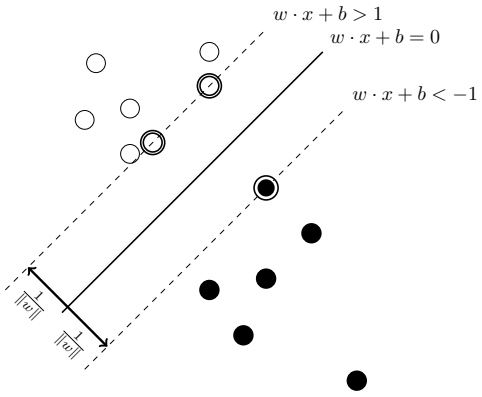
Support Vector Machine - Exact Classification:

Given labeled data (x^i, y_i) for $i = 1, \ldots, N$, 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

$$x^i \cdot w + b > 0 \qquad \text{if } y^i = 1 \tag{6.4.4}$$

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

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 size, we set the right hand sides to 1 and -1 and then compute the margin form 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 $\|w\| = \sqrt{w_1^2 + \cdots + w_d^2}$. Hence, maximizing the margin is equivalent to minimizing $w_1^2 + \cdots + w_d^2$. We arrive at the model

$$\min \sum_{i=1}^{d} w_i^2 \tag{6.4.6}$$

$$x^i \cdot w + b \geq 1 \qquad \text{if } y^i = 1 \tag{6.4.7}$$

$$x^i \cdot w + b \leq -1 \qquad \text{if } y^i = -1 \tag{6.4.8}$$

Or even more compactly written as

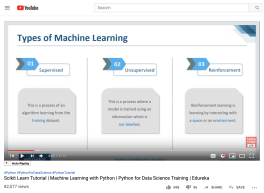
$$\min \sum_{i=1}^{d} w_i^2 \tag{6.4.9}$$

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

6.5 Classification

6.5.1 Machine Learning

https://www.youtube.com/watch?v=bwZ3Qiuj3i8&list=PL9ooVrP1hQOHUfd-g8GUpKI3hHOwM_



9Dn&index=13

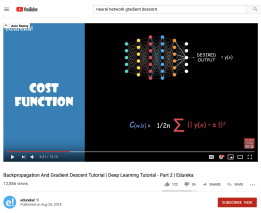
https://towardsdatascience.com/solving-a-simple-classification-problem-with-python-fruits-lo

6.5.2 Neural Networks

https://www.youtube.com/watch?v=bVQUSndDllU

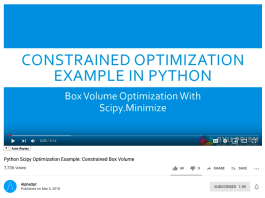
https://www.youtube.com/watch?v=8bNIkfRJZpo

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 $x^2 + \bar{y}^2$.

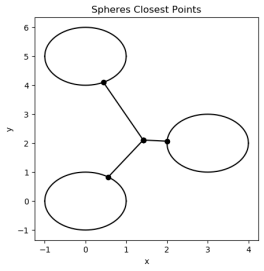
7. **Provide derivatives.** Working out gradient and hessian information by hand can save the solver time. Particularly when these are sparse (many zeros). These can often be provided directly to the solver.

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 $i = 1, \ldots, n$, 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:

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

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 an 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 $f(z) = \|z\|_2^2$, $z = (x_i - c_x, y_i - x_y)$. Since $f(z)$ is convex and the transformation into variables x_i, c_x, y_i, c_y is linear, we have that $f(x_i - c_x, y_i - x_y)$ is convex. Then since $-w_i$ is linear, we have that

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

is a convex function. Thus, the constraint

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

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:

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

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:

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

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

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 chose 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 be a factor γ, 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 γ 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 of $(1-\gamma)\gamma$, which we set as the length between x_1 and x_2.

Considering the geometry, we have

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

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 α, tolerance ϵ
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 α, tolerance ϵ
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 Optimizaition

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

Choice of α_t

There are many different ways to choose the step length α_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, μ, γ are some numbers. The number μ 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

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

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 33: 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

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

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,

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

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

Choice of Δ_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 $f(x)\colon \mathbb{R}^d \to \mathbb{R}$ and convex functions $f_i(x)\colon \mathbb{R}^d \to \mathbb{R}$ for $i = 1, \ldots, m$, the *convex programming* problem is

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

7.4.1 Barrier Method

Constrained Convex Programming via Barrier Method:
We convert (7.4.1) into the unconstrained minimization problem:

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

Here $\phi > 0$ is some number that we choose. As $\phi \to 0$, the optimal solution $x(\phi)$ to (7.4.2) tends to the optimal solution of (7.4.1). That is $x(\phi) \to x^*$ as $\phi \to 0$.

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.

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

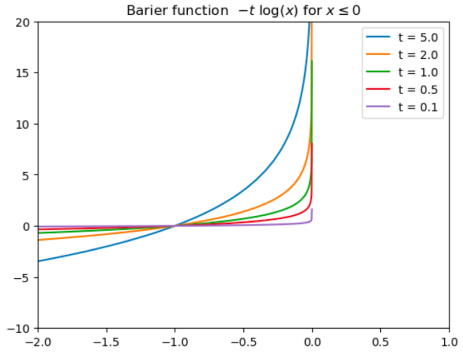
Note that this problem is feasible for all x values since s can always be made larger. If there exists a solution with $s \le 0$, then we can use the corresponding x solution as an initial feasible solution. Otherwise, the problem is infeasible.

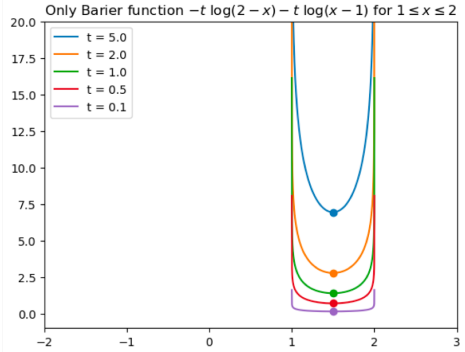
Now, convert this problem into the unconstrained minimization problem:

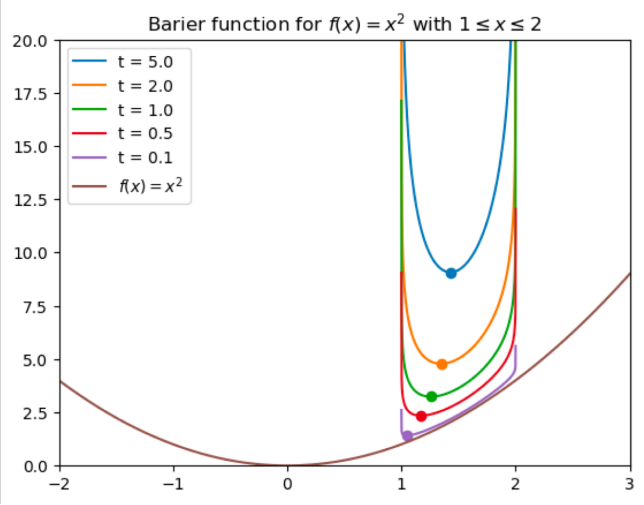
$$\begin{aligned} \min \quad & f(x) - \phi \sum_{i=1}^{m} \log(-(f_i(x) - s)) \\ & x \in \mathbb{R}^d, s \in \mathbb{R} \end{aligned} \tag{7.4.4}$$

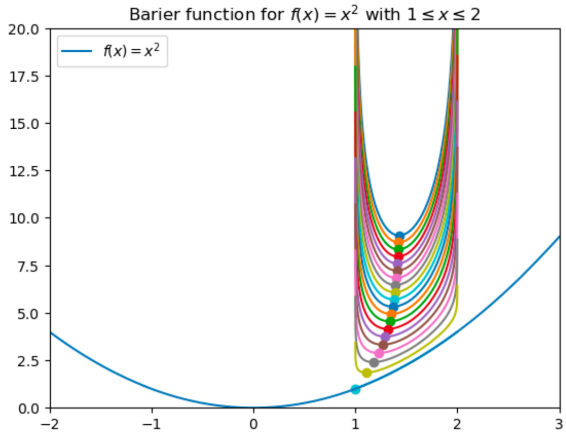
This problem has an easy time of finding an initial feasible solution. For instance, let $x = 0$, and then $s = \max_i f_i(x) + 1$.

Images below: the value t is the value ϕ discussed above

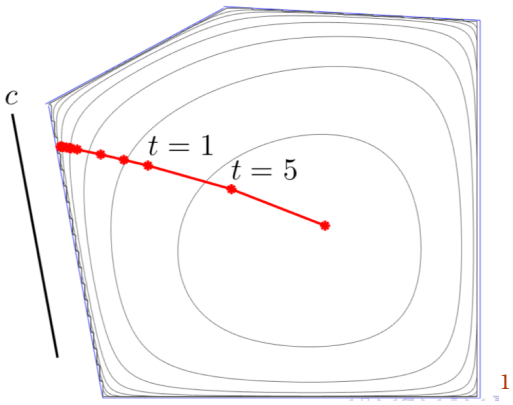








Minimizing $c^T x$ subject to $Ax \leq b$.



[1]

[1]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)

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

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)

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

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 $f(x)\colon \mathbb{R}^d \to \mathbb{R}$ and convex functions $f_i(x)\colon \mathbb{R}^d \to \mathbb{R}$ for $i = 1, \ldots, m$, the *convex programming* problem is

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

Example 34: 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

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

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 $f(x)\colon \mathbb{R}^d \to \mathbb{R}$ and convex functions $g_i(x)\colon \mathbb{R}^d \to \mathbb{R}$ for $i = 1, \ldots, m$, the *convex programming* problem is

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

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).
$$\bar{\lambda}_i g_i(\bar{x}) = 0 \text{ for } i = 1, \ldots, m \tag{8.5.3}$$
3. (Primal Feasibility).
$$g_i(\bar{x}) \leq 0 \text{ for } i = 1, \ldots, m \tag{8.5.4}$$
4. (Dual Feasibility).
$$\bar{\lambda}_i \geq 0 \text{ for } i = 1, \ldots, m \tag{8.5.5}$$

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

8.6 Gradient Free Algorithms

8.6.1 Needler-Mead

https://en.wikipedia.org/wiki/Nelder%E2%80%93Mead_method https://youtube/NI3WllrvWoc?t=96

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

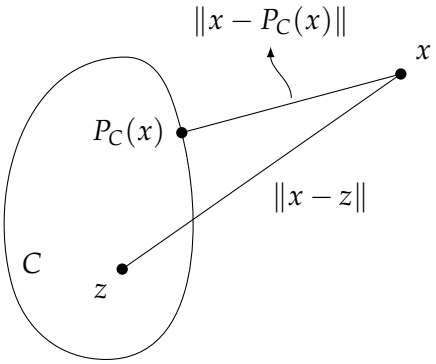
https://www.youtube.com/watch?v=N_ZRcLheNv0

https://www.youtube.com/watch?v=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

9.2 Projection Gradient Methods



h