Mathematical Programming and Operations Research

Modeling, Algorithms, and Complexity

Examples in Excel and Python

(Work in progress)

Edited by: Robert Hildebrand

Contributors: Robert Hildebrand, Laurent Poirrier, Douglas Bish, Diego Moran

Version Compilation date: February 16, 2024

Preface

This entire book is a working manuscript. The first draft of the book is yet to be completed.

This book is being written and compiled using a number of open source materials. We will strive to properly cite all resources used and give references on where to find these resources. Although the material used in this book comes from a variety of licences, everything used here will be CC-BY-SA 4.0 compatible, and hence, the entire book will fall under a CC-BY-SA 4.0 license.

MAJOR ACKNOWLEDGEMENTS

I would like to acknowledge that substantial parts of this book were borrowed under a CC-BY-SA license. These substantial pieces include:

- "A First Course in Linear Algebra" by Lyryx Learning (based on original text by Ken Kuttler). A majority of their formatting was used along with selected sections that make up the appendix sections on linear algebra. We are extremely grateful to Lyryx for sharing their files with us. They do an amazing job compiling their books and the templates and formatting that we have borrowed here clearly took a lot of work to set up. Thank you for sharing all of this material to make structuring and formating this book much easier! See subsequent page for list of contributors.

- "Foundations of Applied Mathmatics" with many contributors. See https://github.com/Foundations-of-Applied-Mathematics. Several sections from these notes were used along with some formatting. Some of this content has been edited or rearranged to suit the needs of this book. This content comes with some great references to code and nice formatting to present code within the book. See subsequent page with list of contributors.

- "Linear Inequalities and Linear Programming" by Kevin Cheung. See https://github.com/dataopt/lineqlpbook. These notes are posted on GitHub in a ".Rmd" format for nice reading online. This content was converted to LaTeX using Pandoc. These notes make up a substantial section of the Linear Programming part of this book.

- Linear Programming notes by Douglas Bish. These notes also make up a substantial section of the Linear Programming part of this book.

I would also like to acknowledge Laurent Porrier and Diego Moran for contributing various notes on linear and integer programming.

I would also like to thank Jamie Fravel for helping to edit this book and for contributing chapters, examples, and code.

Contents

Introduction

Letter to instructors

This is an introductory book for students to learn optimization theory, tools, and applications. The two main goals are (1) students are able to apply the of optimization in their future work or research (2) students understand the concepts underlying optimization solver and use this knowledge to use solvers effectively.

This book was based on a sequence of course at Virginia Tech in the Industrial and Systems Engineering Department. The courses are *Deterministic Operations Research I* and *Deterministic Operations Reserach II*. The first course focuses on linear programming, while the second course covers integer programming an nonlinear programming. As such, the content in this book is meant to cover 2 or more full courses in optimization.

The book is designed to be read in a linear fashion. That said, many of the chapters are mostly independent and could be rearrranged, removed, or shortened as desited. This is an open source textbook, so use it as you like. You are encouraged to share adaptations and improvements so that this work can evolve over time.

Letter to students

This book is designed to be a resource for students interested in learning what optimizaiton is, how it works, and how you can apply it in your future career. The main focus is being able to apply the techniques of optimization to problems using computer technology while understanding (at least at a high level) what the computer is doing and what you can claim about the output from the computer.

Quite importantly, keep in mind that when someone claims to have *optimized* a problem, we want to know what kind of gaurantees they have about how good their solution is. Far too often, the solution that is provided is suboptimal by 10%, 20%, or even more. This can mean spending excess amounts of money, time, or energy that could have been saved. And when problems are at a large scale, this can easily result in millions of dollars in savings.

For this reason, we will learn the perspective of *mathematical programming* (a.k.a. mathematical optimization). The key to this study is that we provide gaurantees on how good a solution is to a given problem. We will also study how difficult a problem is to solve. This will help us know (a) how long it might take to solve it and (b) how good a of a solution we might expect to be able to find in reasonable amount of time.

We will later study *heuristic* methods - these methods typically do not come with gaurantees, but tend to help find quality solutions.

Note: Although there is some computer programming required in this work, this is not a course on programming. Thanks to the fantastic modelling packages available these days, we are able to solve complicated problems with little programming effort. The key skill we will need to learn is *mathematical modeling*: converting words and ideas into numbers and variables in order to communicate problems to a compter so that it can solve a problem for you.

As a main element of this book, we would like to make the process of using code and software as easy as possible. Attached to most examples in the book, there will be links to code that implements and solves the problem using several different tools from Excel and Python. These examples can should make it easy to solve a similar problem with different data, or more generally, can serve as a basis for solving related problems with similar structure.

How to use this book

Skim ahead. We recommend that before you come across a topic in lecture, that you skim the relevant sections ahead of time to get a broad overview of what is to come. This may take only a fraction of the time that it may take for you to read it.

Read the expected outcomes. At the beginning of each section, there will be a list of expected outcomes from that section. Review these outcomes before reading the section to help guide you through what is most relevant for you to take away from the material. This will also provide a brief look into what is to follow in that section.

Read the text. Read carefully the text to understand the problems and techniques. We will try to provide a plethora of examples, therefore, depending on your understanding of a topic, you many need to go carefully over all of the examples.

Explore the resources. Lastly, we recognize that there are many alternative methods of learning given the massive amounts of information and resources online. Thus, at the end of each section, there will be a number of superb resources that available on the internet in different formats. There are other free textbooks, informational websites, and also number of fantastic videos posted to youtube. We encourage to explore the resources to get another perspective on the material or to hear/read it taught from a differnt point of view or in presentation style.

Outline of this book

This book is divided in to 4 Parts:

Part I Linear Programming,

Part II Integer Programming,

Part III Discrete Algorithms,

Part IV Nonlinear Programming.

There are also a number of chapters of background material in the Appendix.

The content of this book is desigened to encompass 2-3 full semester courses in an industrial engineering department.

Work in progress

This book is still a work in progress, so please feel free to send feedback, questions, comments, and edits to Robert Hildebrand at `open.optimization@gmail.com`.

1. Resources and Notation

Here are a list of resources that may be useful as alternative references or additional references.

FREE NOTES AND TEXTBOOKS

- Mathematical Programming with Julia by Richard Lusby & Thomas Stidsen

- Linear Programming by K.J. Mtetwa, David

- A first course in optimization by Jon Lee

- Introduction to Optimizaiton Notes by Komei Fukuda

- Convex Optimization by Bord and Vandenberghe

- LP notes of Michel Goemans from MIT

- Understanding and Using Linear Programming - Matousek and Gärtner [Downloadable from Springer with University account]

- Operations Research Problems Statements and Solutions - Raúl PolerJosefa Mula Manuel Díaz-Madroñero [Downloadable from Springer with University account]

NOTES, BOOKS, AND VIDEOS BY VARIOUS SOLVER GROUPS

- AIMMS Optimization Modeling

- Optimization Modeling with LINGO by Linus Schrage

- The AMPL Book

- Microsoft Excel 2019 Data Analysis and Business Modeling, Sixth Edition, by Wayne Winston - Available to read for free as an e-book through Virginia Tech library at Orielly.com.

- Lesson files for the Winston Book

- Video instructions for solver and an example workbook

- youtube-OR-course

GUROBI LINKS

- Go to `https://github.com/Gurobi` and download the example files.

- Essential ingredients

- Gurobi Linear Programming tutorial

- Gurobi tutorial MILP

- GUROBI - Python 1 - Modeling with GUROBI in Python

- GUROBI - Python II: Advanced Algebraic Modeling with Python and Gurobi

- GUROBI - Python III: Optimization and Heuristics

- Webinar Materials

- GUROBI Tutorials

HOW TO PROVE THINGS

- Hammack - Book of Proof

STATISTICS

- Open Stax - Introductory Statistics

LINEAR ALGEBRA

- Beezer - A first course in linear algebra

- Selinger - Linear Algebra

- Cherney, Denton, Thomas, Waldron - Linear Algebra

REAL ANALYSIS

- Mathematical Analysis I by Elias Zakon

DISCRETE MATHEMATICS, GRAPHS, ALGORITHMS, AND COMBINATORICS

- Levin - Discrete Mathematics - An Open Introduction, 3rd edition

- Github - Discrete Mathematics: an Open Introduction CC BY SA

- Keller, Trotter - Applied Combinatorics (CC-BY-SA 4.0)

- Keller - Github - Applied Combinatorics

MIXED INTEGER NONLINEAR PROGRAMMING

- Mixed-Integer Nonlinear Optimization Pietro Belotti, Christian Kirches, Sven Leyffer, Jeff Linderoth, Jim Luedtke, and Ashutosh Mahajan

PROGRAMMING WITH PYTHON

- A Byte of Python

- Github - Byte of Python (CC-BY-SA)

Also, go to `https://github.com/open-optimization/open-optimization-or-examples` to look at more examples.

Notation

- 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. $1, 1.5, \pi, e, -11/5$)

- $\setminus$ - setminus, (e.g. $\{0, 1, 2, 3\} \setminus \{0, 3\} = \{1, 2\}$)

- $\cup$ - union (e.g. $\{1, 2\} \cup \{3, 5\} = \{1, 2, 3, 5\}$)

- $\cap$ - intersection (e.g. $\{1,2,3,4\} \cap \{3,4,5,6\} = \{3,4\}$)

- $\{0,1\}^4$ - the set of 4 dimensional vectors taking values 0 or 1, (e.g. $[0,0,1,0]$ or $[1,1,1,1]$)

- $\mathbb{Z}^4$ - the set of 4 dimensional vectors taking integer values (e.g., $[1,-5,17,3]$ or $[6,2,-3,-11]$)

- $\mathbb{Q}^4$ - the set of 4 dimensional vectors taking rational values (e.g. $[1.5,3.4,-2.4,2]$)

- $\mathbb{R}^4$ - the set of 4 dimensional vectors taking real values (e.g. $[3,\pi,-e,\sqrt{2}]$)

- $\sum_{i=1}^4 i = 1+2+3+4$

- $\sum_{i=1}^4 i^2 = 1^2 + 2^2 + 3^2 + 4^4$

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

- $\square$ - 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."

2. Mathematical Programming

> **Outcomes**
> - *Identify reasons for studying operations research*
> - *Define "Mathematical Programming"*
> - *Learn about different applications of the tools in this book*
> - *Explore the different types of optimization models and what types we will see in this book.*

2.1 Why study operations research?

2.2 What is Mathematical Programming?

2.3 Applications

2.4 Types of Optimization problems

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

Along with each problem class, we will associate a complexity class for the general version of the problem. See **??** 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] *Tree of optimization problems.* from . Diego Moran CC0., 2017.
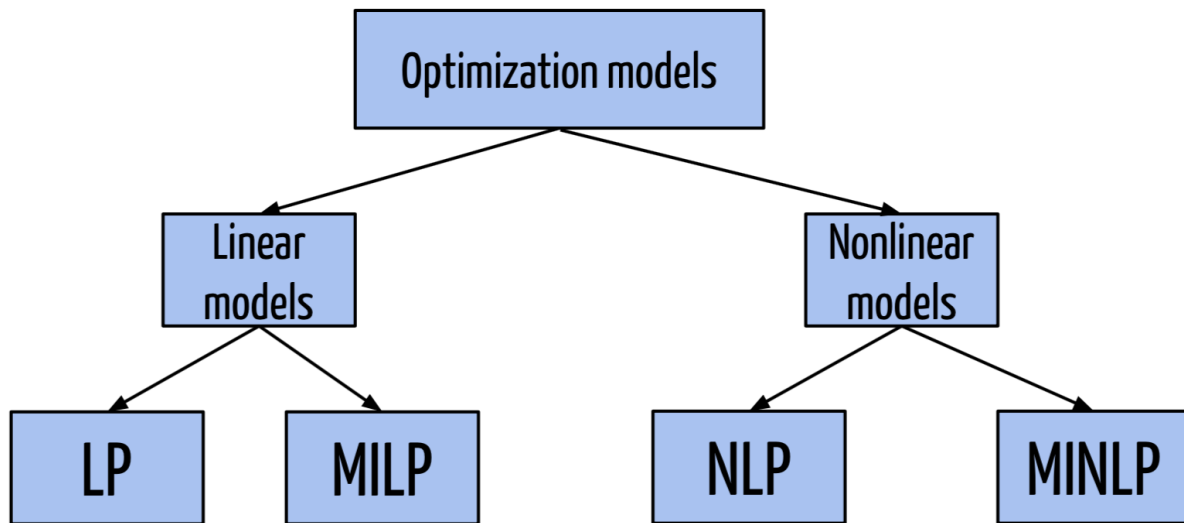
Figure 2.1: *Tree of optimization problems.*

2.5 Linear Programming (LP)

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

Linear Programming (LP):

Polynomial time (P)

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

$$
\begin{aligned}
\max \quad & c^\top x \\
\text{s.t.} \quad & Ax \leq b \\
& x \geq 0
\end{aligned}
\tag{2.1}
$$

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

$$\begin{aligned}
\max \quad & c^\top x \\
\text{s.t.} \quad & Ax = b \\
& x \geq 0
\end{aligned} \tag{2.2}$$
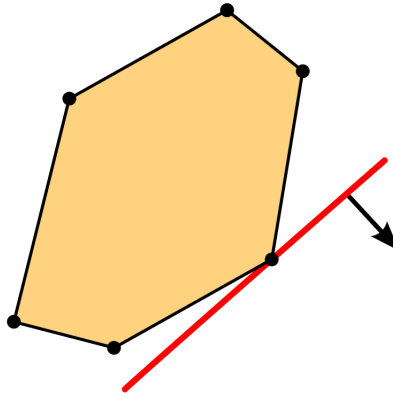
© Ylloh CC0.[2]
Figure 2.2: Linear programming constraints and objective.

Figure 2.2

> **Exercise 2.1:**
>
> *Start with a problem in form given as (2.1) and convert it to standard form (2.2) by adding at most m many new variables and by enlarging the constraint matrix A by at most m new columns.*

2.6 Mixed-Integer Linear Programming (MILP)

Mixed-integer linear programming will be the focus of Sections **??,??**, **??**, 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).

---

[2]*A pictorial representation of a simple linear program with two variables and six inequalities. The set of feasible solutions is depicted in yellow and forms a polygon, a 2-dimensional polytope. The linear cost function is represented by the red line and the arrow: The red line is a level set of the cost function, and the arrow indicates the direction in which we are optimizing.* from `https://en.wikipedia.org/wiki/Linear_programming#/media/File:Linear_optimization_in_a_2-dimensional_polytope.svg`. Ylloh CC0., 2012.

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

$$\begin{aligned}
\max \quad & c^\top x \\
\text{s.t.} \quad & Ax \le b \\
& x \in \{0,1\}^n
\end{aligned} \tag{2.1}$$

A slightly more general class is the class of *Integer Linear Programs* (ILP). Often this is referred to as *Integer Program* (IP), although this term could leave open the possibility of non-linear parts.
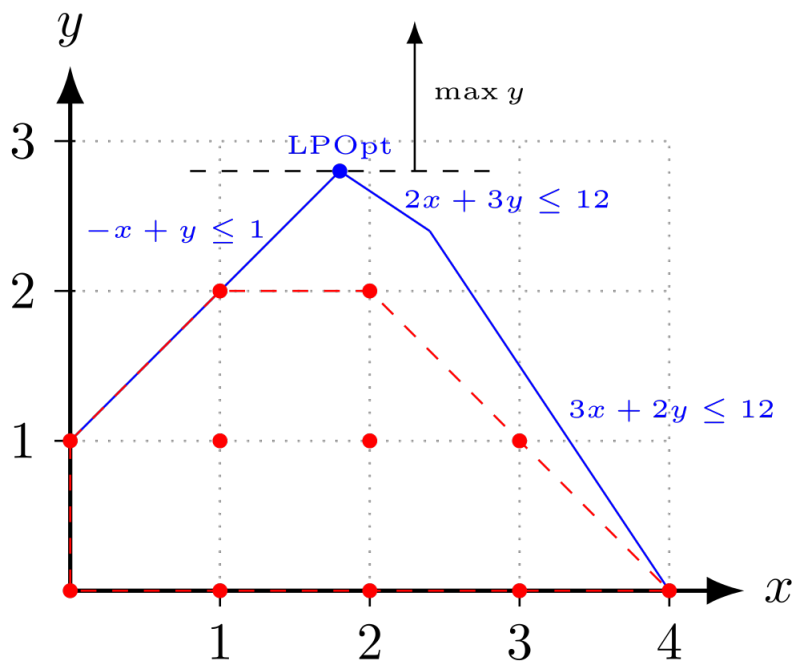
Figure 2.3: Comparing the LP relaxation to the IP solutions.

Figure 2.3

Integer Linear Programming (ILP):

NP-Complete

---

[3]*IP polytope with LP relaxation*, from `https://en.wikipedia.org/wiki/Integer_programming#/media/File:IP_polytope_with_LP_relaxation.svg`. Fanosta CC BY-SA 4.0., 2019.

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

$$
\begin{aligned}
\max \quad & c^\top x \\
\text{s.t.} \quad & Ax \leq b \\
& x \in \mathbb{Z}^n
\end{aligned}
\tag{2.2}
$$

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

$$
\begin{aligned}
\max \quad & c^\top x \\
\text{s.t.} \quad & Ax \leq b \\
& x \in \mathbb{Z}^n \times \mathbb{R}^d
\end{aligned}
\tag{2.3}
$$

2.7 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

$$
\begin{aligned}
\min \quad & f(x) \\
\text{s.t.} \quad & f_i(x) \leq 0 \quad \text{for } i = 1, \ldots, m \\
& x \in \mathbb{R}^d
\end{aligned}
\tag{2.1}
$$

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.

2.7.1. 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

$$
\begin{aligned}
\min \quad & f(x) \\
\text{s.t.} \quad & f_i(x) \leq 0 \quad \text{for } i = 1, \ldots, m \\
& x \in \mathbb{R}^d
\end{aligned}
\tag{2.2}
$$

Observe that convex programming is a generalization of linear programming. This can be seen by letting $f(x) = c^\top x$ and $f_i(x) = A_i x - b_i$.

2.7.2. Non-Convex Non-linear Programming

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

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

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

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

Binary Integer programming (BIP) as a NLP:

NP-Hard

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

is

$$\begin{aligned}
\max \quad & c^\top x \\
\text{s.t.} \quad & Ax \leq b \\
& \cancel{x \in \{0,1\}^n} \\
& x_i(1 - x_i) = 0 \quad \text{for } i = 1,\ldots,n
\end{aligned} \tag{2.3}$$

Alternatively, consider the transformation where $x_i \in \{-1,1\}$.

$$\begin{aligned}
\min \, & c^\top x \\
\text{s.t. } & Ax \leq b \\
& x \in \{-1,1\}^n
\end{aligned}$$

This can be reformulated with a single nonconvex constraint as

$$\begin{aligned}
\min \quad & c^\top x \\
\text{s.t.} \quad & Ax \leq b \\
& -1 \leq x_j \leq 1, \quad 1 \leq j \leq n, \\
& \|x\|^2 \geq n.
\end{aligned}$$

2.7.3. Machine Learning

Machine learning problems are often cast as continuous optimization problems, which involve adjusting parameters to minimize or maximize a particular objective. Frequently they are convex optimization problems, but many turn out to be nonconvex. Here are two examples of how these problems arise at a glance. We will see examples in greater detail later in the book.

Loss Function Minimization

In supervised learning, this objective is typically a loss function L that quantifies the discrepancy between the predictions of a model and the true data labels. The aim is to adjust the parameters θ of the model to minimize this loss. Mathematically, this can be represented as:

$$\min_\theta L(\theta) = \min_\theta \frac{1}{N} \sum_{i=1}^{N} l(y_i, f(x_i; \theta)) \tag{2.4}$$

where N is the number of data points, l is a per-data-point loss (e.g., squared error for regression or cross-entropy for classification), y_i is the true label for the i-th data point, and $f(x_i; \theta)$ is the model's prediction for the i-th data point with parameters θ.

Clustering Formulation

Clustering, on the other hand, seeks to group or partition data points such that data points in the same group are more similar to each other than those in other groups. One popular method is the k-means clustering algorithm. The objective of k-means is to partition the data into k clusters by minimizing the within-cluster sum of squares (WCSS). The mathematical formulation can be given as:

$$\min_{\mathbf{c}_1,\ldots,\mathbf{c}_k} \sum_{j=1}^{k} \sum_{x_i \in C_j} \left\| x_i - \mathbf{c}_j \right\|^2 \tag{2.5}$$

where C_j represents the j-th cluster and $\mathbf{c}_j$ is the centroid of that cluster.

This encapsulation presents a glimpse into how ML problems are framed mathematically. In practice, numerous algorithms, constraints, and regularizations add complexity to these basic formulations.

2.8 Mixed Integer Non-Linear Programming (MINLP)

MINLP:

NP-Hard

Mixed Integer Nonlinear Programming (MINLP) combines elements of integer programming and non-linear programming. In MINLP, some or all of the decision variables are constrained to be integers, and the objective function or constraints are nonlinear. A general MINLP problem can be formulated as:

$$
\begin{aligned}
\min \quad & f(x,y) \\
\text{s.t.} \quad & g_i(x,y) \leq 0 \quad \text{for } i = 1,\ldots,m \\
& h_j(x,y) = 0 \quad \text{for } j = 1,\ldots,p \\
& x \in \mathbb{R}^n, y \in \mathbb{Z}^k
\end{aligned} \tag{2.1}
$$

where f is the objective function, g_i and h_j are constraint functions, x is a vector of continuous variables, and y is a vector of integer variables.

MINLP is particularly challenging due to the nonlinearity in the objective and/or constraints and the discrete nature of some decision variables. It finds applications in various fields such as industry for process optimization, computational geometry, and machine learning for hyperparameter tuning.

2.8.1. Convex MINLP

In the convex case, both the objective function and the constraints are convex functions. This subclass is easier to solve compared to its non-convex counterpart.

Convex MINLP:

NP-Hard, but *Polynomial time (P)* in fixed dimension (typically)

For a convex MINLP, the problem is defined as:

$$\begin{aligned}
\min \quad & f(x,y) \quad \text{(convex)} \\
\text{s.t.} \quad & g_i(x,y) \leq 0 \quad \text{(convex constraints)} \\
& x \in \mathbb{R}^n, y \in \mathbb{Z}^k
\end{aligned} \tag{2.2}$$

Convex MINLPs, while still challenging, are typically more tractable due to the properties of convexity, which allow for more efficient solution methods.

2.8.2. Non-Convex MINLP

Non-convex MINLPs are significantly harder due to the presence of non-convex functions, which can lead to multiple local minima.

Non-Convex MINLP:

NP-Hard(in fact, undecidable)

The non-convex MINLP problem is formulated as:

$$\begin{aligned}
\min \quad & f(x,y) \quad \text{(non-convex)} \\
\text{s.t.} \quad & g_i(x,y) \leq 0 \quad \text{(possibly non-convex constraints)} \\
& x \in \mathbb{R}^n, y \in \mathbb{Z}^k
\end{aligned} \tag{2.3}$$

Non-convex MINLPs pose significant computational challenges due to the possibility of multiple local optima and the inherent complexity of integer constraints. These problems are common in real-world applications where decisions are discrete, and the system behavior is non-linear and complex.

Complexity and Applications

MINLP problems are known for their computational complexity, primarily due to the combination of non-linearity and integrality. The non-convex variants, in particular, are *NP-Hard*, making them some of the most challenging problems in optimization.

In practice, MINLP models find extensive applications across various domains. In industry, they are used for complex decision-making processes like supply chain optimization and production planning. In computational geometry, MINLP techniques help in solving problems like optimal packing or layout design. Furthermore, in the realm of machine learning, MINLPs are crucial for tasks like feature selection and hyperparameter optimization where discrete choices and nonlinear relationships are involved.

The versatility of MINLP models, combined with their inherent complexity, makes them a fascinating and essential area of study in the field of optimization.

Part I

Linear Programming

3. Modeling: Linear Programming

Outcomes

1. *Define what a linear program is*
2. *Understand how to model a linear program*
3. *View many examples and get a sense of what types of problems can be modeled as linear programs.*

Linear Programming, also known as Linear Optimization, is the starting point for most forms of optimization. It is the problem of optimization a linear function over linear constraints.

In this section, we will define what this means, how to setup a linear program, and discuss many examples. Examples will be connected with code in Excel and Python (using with PuLP or Gurobipy modeling tools) so that you can easily start solving optimization problems. Tutorials on these tools will come in later chapters.

We begin this section with a simple example.

Example: Toy Maker   Excel   PuLP   Gurobipy

Consider the problem of a toy company that produces toy planes and toy boats. The toy company can sell its planes for \$10 and its boats for \$8 dollars. It costs \$3 in raw materials to make a plane and \$2 in raw materials to make a boat. A plane requires 3 hours to make and 1 hour to finish while a boat requires 1 hour to make and 2 hours to finish. The toy company knows it will not sell anymore than 35 planes per week. Further, given the number of workers, the company cannot spend anymore than 160 hours per week finishing toys and 120 hours per week making toys. The company wishes to maximize the profit it makes by choosing how much of each toy to produce.

We can represent the profit maximization problem of the company as a linear programming problem. Let x_1 be the number of planes the company will produce and let x_2 be the number of boats the company will produce. The profit for each plane is $\$10 - \$3 = \$7$ per plane and the profit for each boat is $\$8 - \$2 = \$6$ per boat. Thus the total profit the company will make is:

$$z(x_1, x_2) = 7x_1 + 6x_2 \tag{3.1}$$

The company can spend no more than 120 hours per week making toys and since a plane takes 3 hours to make and a boat takes 1 hour to make we have:

$$3x_1 + x_2 \leq 120 \tag{3.2}$$

21

Likewise, the company can spend no more than 160 hours per week finishing toys and since it takes 1 hour to finish a plane and 2 hour to finish a boat we have:

$$x_1 + 2x_2 \leq 160 \tag{3.3}$$

Finally, we know that $x_1 \leq 35$, since the company will make no more than 35 planes per week. Thus the complete linear programming problem is given as:

$$
\begin{cases}
\max \ z(x_1, x_2) = 7x_1 + 6x_2 \\
s.t. \ 3x_1 + x_2 \leq 120 \\
\quad\ x_1 + 2x_2 \leq 160 \\
\quad\ x_1 \leq 35 \\
\quad\ x_1 \geq 0 \\
\quad\ x_2 \geq 0
\end{cases}
\tag{3.4}
$$

Exercise 3.1: Chemical Manufacturing

A chemical manufacturer produces three chemicals: A, B and C. These chemical are produced by two processes: 1 and 2. Running process 1 for 1 hour costs $4 and yields 3 units of chemical A, 1 unit of chemical B and 1 unit of chemical C. Running process 2 for 1 hour costs $1 and produces 1 units of chemical A, and 1 unit of chemical B (but none of Chemical C). To meet customer demand, at least 10 units of chemical A, 5 units of chemical B and 3 units of chemical C must be produced daily. Assume that the chemical manufacturer wants to minimize the cost of production. Develop a linear programming problem describing the constraints and objectives of the chemical manufacturer. [Hint: Let x_1 be the amount of time Process 1 is executed and let x_2 be amount of time Process 2 is executed. Use the coefficients above to express the cost of running Process 1 for x_1 time and Process 2 for x_2 time. Do the same to compute the amount of chemicals A, B, and C that are produced.]

3.1 Modeling and Assumptions in Linear Programming

Outcomes

1. *Address crucial assumptions when chosing to model a problem with linear programming.*

3.1.1. General models

A Generic Linear Program (LP)

Decision Variables:
x_i : continuous variables ($x_i \in \mathbb{R}$, i.e., a real number), $\forall i = 1, \ldots, 3$.

Parameters (known input parameters):
c_i : cost coefficients $\forall i = 1, \ldots, n$
a_{ij} : constraint coefficients $\forall i = 1, \ldots, n, \; j = 1, \ldots, m$
b_j : right hand side coefficient for constraint $j, \; j = 1, \ldots, m$

The problem we will consider is

$$
\begin{aligned}
\max \; z &= c_1 x_1 + \cdots + c_n x_n \\
\text{s.t.} \; a_{11} x_1 + \cdots + a_{1n} x_n &\leq b_1 \\
&\vdots \\
a_{m1} x_1 + \cdots + a_{mn} x_n &\leq b_m
\end{aligned}
\tag{3.1}
$$

For example, in 3 variables and 4 contraints this could look like the following. The following example considers other types of constraints, i.e., $\geq$ and $=$. We will show how all these forms can be converted later.

Decision Variables:
x_i : continuous variables ($x_i \in \mathbb{R}$, i.e., a real number), $\forall i = 1, \cdots, 3$.

Parameters (known input parameters):
c_i : cost coefficients $\forall i = 1, \ldots, 3$
a_{ij} : constraint coefficients $\forall i = 1, \ldots, 3, \; j = 1, \ldots, 4$
b_j : right hand side coefficient for constraint $j, \; j = 1, \ldots, 4$

$$
\begin{aligned}
\text{Min} \; z &= c_1 x_1 + c_2 x_2 + c_3 x_3 & (3.2) \\
\text{s.t.} \; a_{11} x_1 + a_{12} x_2 + a_{13} x_3 &\geq b_1 & (3.3) \\
a_{21} x_1 + a_{22} x_2 + a_{23} x_3 &\leq b_2 & (3.4) \\
a_{31} x_1 + a_{32} x_2 + a_{33} x_3 &= b_3 & (3.5) \\
a_{41} x_1 + a_{42} x_2 + a_{43} x_3 &\geq b_4 & (3.6) \\
x_1 \geq 0, x_2 \leq 0, x_3 \; &urs. & (3.7)
\end{aligned}
$$

> **Definition 3.2: Linear Function**
>
> *A function $z : \mathbb{R}^n \to \mathbb{R}$ is linear if there are constants $c_1, \ldots, c_n \in \mathbb{R}$ so that:*
>
> $$z(x_1, \ldots, x_n) = c_1 x_1 + \cdots + c_n x_n \qquad (3.8)$$

For the time being, we will eschew the general form and focus exclusively on linear programming problems with two variables. Using this limited case, we will develop a graphical method for identifying optimal solutions, which we will generalize later to problems with arbitrary numbers of variables.

3.1.2. Assumptions

Inspecting Example 1 (or the more general Problem 3.1) we can see there are several assumptions that must be satisfied when using a linear programming model. We enumerate these below:

Proportionality Assumption A problem can be phrased as a linear program only if the contribution to the objective function *and* the left-hand-side of each constraint by each decision variable $(x_1, \ldots, x_n)$ is proportional to the value of the decision variable.

Additivity Assumption A problem can be phrased as a linear programming problem only if the contribution to the objective function *and* the left-hand-side of each constraint by any decision variable x_i $(i = 1, \ldots, n)$ is completely independent of any other decision variable x_j $(j \neq i)$ and additive.

Divisibility Assumption A problem can be phrased as a linear programming problem only if the quantities represented by each decision variable are infinitely divisible (i.e., fractional answers make sense).

Certainty Assumption A problem can be phrased as a linear programming problem only if the coefficients in the objective function and constraints are known with certainty.

The first two assumptions simply assert (in English) that both the objective function and functions on the left-hand-side of the (in)equalities in the constraints are linear functions of the variables $x_1, \ldots, x_n$.

The third assumption asserts that a valid optimal answer could contain fractional values for decision variables. It's important to understand how this assumption comes into play–even in the toy making example. Many quantities can be divided into non-integer values (ounces, pounds etc.) but many other quantities cannot be divided. For instance, can we really expect that it's reasonable to make $\frac{1}{2}$ a plane in the toy making example? When values must be constrained to true integer values, the linear programming problem is called an *integer programming problem*. There is a *vast* literature dealing with these problems [**PS98**, **WN99**]. For many problems, particularly when the values of the decision variables may become large, a fractional optimal answer could be obtained and then rounded to the nearest integer to obtain a reasonable answer. For example, if our toy problem were re-written so that the optimal answer was to make 1045.3 planes, then we could round down to 1045.

The final assumption asserts that the coefficients (e.g., profit per plane or boat) is known with absolute

certainty. In traditional linear programming, there is no lack of knowledge about the make up of the objective function, the coefficients in the left-hand-side of the constraints or the bounds on the right-hand-sides of the constraints. There is a literature on *stochastic programming* [**KW94**, **BN02**] that relaxes some of these assumptions, but this too is outside the scope of the course.

> **Exercise 3.3**
>
> *In a short sentence or two, discuss whether the problem given in Example 1 meets all of the assumptions of a scenario that can be modeled by a linear programming problem. Do the same for Exercise 3.* [Hint: Can you make $\frac{2}{3}$ of a toy? Can you run a process for $\frac{1}{3}$ of an hour?]

3.2 Examples

> **Outcomes**
>
> A. *Learn how to format a linear optimization problem.*
> B. *Identify and understand common classes of linear optimization problems.*

We will begin with a few examples, and then discuss specific problem types that occur often.

Example: Production with welding robot   Excel PuLP Gurobipy

You have 21 units of transparent aluminum alloy (TAA), LazWeld1, a joining robot leased for 23 hours, and CrumCut1, a cutting robot leased for 17 hours of aluminum cutting. You also have production code for a bookcase, desk, and cabinet, along with commitments to buy any of these you can produce for $18, $16, and $10 apiece, respectively. A bookcase requires 2 units of TAA, 3 hours of joining, and 1 hour of cutting, a desk requires 2 units of TAA, 2 hours of joining, and 2 hour of cutting, and a cabinet requires 1 unit of TAA, 2 hours of joining, and 1 hour of cutting. Formulate an LP to maximize your revenue given your current resources.

Solution.

Sets:

- The types of objects = { bookcase, desk, cabinet}.

Parameters:

- Purchase cost of each object

- Units of TAA needed for each object

- Hours of joining needed for each object

- Hours of cutting needed for each object

- Hours of TAA, Joining, and Cutting available on robots

Decision variables:

x_i : number of units of product i to produce,
for all $i =$bookcase, desk, cabinet.

Objective and Constraints:

$$
\begin{aligned}
\max \ z = {} & 18x_1 + 16x_2 + 10x_3 && \text{(profit)} \\
\text{s.t.} \ \ & 2x_1 + 2x_2 + 1x_3 \le 21 && (TAA) \\
& 3x_1 + 2x_2 + 2x_3 \le 23 && (LazWeld1) \\
& 1x_1 + 2x_2 + 1x_3 \le 17 && (CrumCut1) \\
& x_1, x_2, x_3 \ge 0.
\end{aligned}
$$

♠

Example: The Diet Problem   Gurobipy

In the future (as envisioned in a bad 70's science fiction film) all food is in tablet form, and there are four types, green, blue, yellow, and red. A balanced, futuristic diet requires, at least 20 units of Iron, 25 units of Vitamin B, 30 units of Vitamin C, and 15 units of Vitamin D. Formulate an LP that ensures a balanced diet at the minimum possible cost.

Tablet	Iron	B	C	D	Cost ($)
green (1)	6	6	7	4	1.25
blue (2)	4	5	4	9	1.05
yellow (3)	5	2	5	6	0.85
red (4)	3	6	3	2	0.65

Solution. Now we formulate the problem:

Sets:

- Set of tablets $\{1,2,3,4\}$

Parameters:

- Iron in each tablet

- Vitamin B in each tablet

- Vitamin C in each tablet

- Vitamin D in each tablet

- Cost of each tablet

Decision variables:
x_i : number of tablet of type i to include in the diet, $\forall i \in \{1,2,3,4\}$.

Objective and Constraints:

$$\text{Min} \quad z = 1.25x_1 + 1.05x_2 + 0.85x_3 + 0.65x_4$$
$$\text{s.t.} \quad 6x_1 + 4x_2 + 5x_3 + 3x_4 \geq 20$$
$$6x_1 + 5x_2 + 2x_3 + 6x_4 \geq 25$$
$$7x_1 + 4x_2 + 5x_3 + 3x_4 \geq 30$$
$$4x_1 + 9x_2 + 6x_3 + 2x_4 \geq 15$$
$$x_1, x_2, x_3, x_4 \geq 0.$$

♠

Example: The Next Diet Problem   Gurobipy

Progress is important, and our last problem had too many tablets, so we are going to produce a single, purple, 10 gram tablet for our futuristic diet requires, which are at least 20 units of Iron, 25 units of Vitamin B, 30 units of Vitamin C, and 15 units of Vitamin D, and 2000 calories. The tablet is made from blending 4 nutritious chemicals; the following table shows the units of our nutrients per, and cost of, grams of each chemical. Formulate an LP that ensures a balanced diet at the minimum possible cost.

Tablet	Iron	B	C	D	Calories	Cost ($)
Chem 1	6	6	7	4	1000	1.25
Chem 2	4	5	4	9	250	1.05
Chem 3	5	2	5	6	850	0.85
Chem 4	3	6	3	2	750	0.65

Solution.

Sets:

- Set of chemicals $\{1,2,3,4\}$

Parameters:

- Iron in each chemical

- Vitamin B in each chemical

- Vitamin C in each chemical

- Vitamin D in each chemical

- Cost of each chemical

Decision variables:
x_i : grams of chemical i to include in the purple tablet, $\forall i = 1,2,3,4$.
Objective and Constraints:

$$\min z = 1.25x_1 + 1.05x_2 + 0.85x_3 + 0.65x_4$$
$$s.t. 6x_1 + 4x_2 + 5x_3 + 3x_4 \geq 20$$
$$6x_1 + 5x_2 + 2x_3 + 6x_4 \geq 25$$
$$7x_1 + 4x_2 + 5x_3 + 3x_4 \geq 30$$
$$4x_1 + 9x_2 + 6x_3 + 2x_4 \geq 15$$
$$1000x_1 + 250x_2 + 850x_3 + 750x_4 \geq 2000$$
$$x_1 + x_2 + x_3 + x_4 = 10$$
$$x_1, x_2, x_3, x_4 \geq 0.$$

♠

Example: Work Scheduling Problem

Gurobipy

You are the manager of LP Burger. The following table shows the minimum number of employees required to staff the restaurant on each day of the week. Each employees must work for five consecutive days. Formulate an LP to find the minimum number of employees required to staff the restaurant.

Day of Week	Workers Required
1 = Monday	6
2 = Tuesday	4
3 = Wednesday	5
4 = Thursday	4
5 = Friday	3
6 = Saturday	7
7 = Sunday	7

Solution. Decision variables:

Decision variables:

x_i : the number of workers that start 5 consecutive days of work on day i, $i = 1, \cdots, 7$

$$\text{Min} \quad z = x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7$$
$$\text{s.t.} \quad x_1 + x_4 + x_5 + x_6 + x_7 \geq 6$$
$$x_2 + x_5 + x_6 + x_7 + x_1 \geq 4$$
$$x_3 + x_6 + x_7 + x_1 + x_2 \geq 5$$
$$x_4 + x_7 + x_1 + x_2 + x_3 \geq 4$$
$$x_5 + x_1 + x_2 + x_3 + x_4 \geq 3$$
$$x_6 + x_2 + x_3 + x_4 + x_5 \geq 7$$
$$x_7 + x_3 + x_4 + x_5 + x_6 \geq 7$$
$$x_1, x_2, x_3, x_4, x_5, x_6, x_7 \geq 0.$$

The solution is as follows:

LP Solution	IP Solution
$z_{LP} = 7.333$	$z_I = 8.0$
$x_1 = 0$	$x_1 = 0$
$x_2 = 0.333$	$x_2 = 0$
$x_3 = 1$	$x_3 = 0$
$x_4 = 2.333$	$x_4 = 3$
$x_5 = 0$	$x_5 = 0$
$x_6 = 3.333$	$x_6 = 4$
$x_7 = 0.333$	$x_7 = 1$

Example: LP Burger - extended   Gurobipy

LP Burger has changed it's policy, and allows, at most, two part time workers, who work for two consecutive days in a week. Formulate this problem.

Solution. Decision variables:

x_i : the number of workers that start 5 consecutive days of work on day i, $i = 1, \cdots, 7$

y_i : the number of workers that start 2 consecutive days of work on day i, $i = 1, \cdots, 7$.

$$
\begin{aligned}
\text{Min} \quad z = {}& 5(x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7) \\
& + 2(y_1 + y_2 + y_3 + y_4 + y_5 + y_6 + y_7) \\
\text{s.t.} \quad & x_1 + x_4 + x_5 + x_6 + x_7 + y_1 + y_7 \geq 6 \\
& x_2 + x_5 + x_6 + x_7 + x_1 + y_2 + y_1 \geq 4 \\
& x_3 + x_6 + x_7 + x_1 + x_2 + y_3 + y_2 \geq 5 \\
& x_4 + x_7 + x_1 + x_2 + x_3 + y_4 + y_3 \geq 4 \\
& x_5 + x_1 + x_2 + x_3 + x_4 + y_5 + y_4 \geq 3 \\
& x_6 + x_2 + x_3 + x_4 + x_5 + y_6 + y_5 \geq 7 \\
& x_7 + x_3 + x_4 + x_5 + x_6 + y_7 + y_6 \geq 7 \\
& y_1 + y_2 + y_3 + y_4 + y_5 + y_6 + y_7 \leq 2 \\
& x_i \geq 0, y_i \geq 0, \forall i = 1, \cdots, 7.
\end{aligned}
$$

♠

3.2.1. Knapsack Problem

Example: Capital allocation   Excel   PuLP   Gurobipy

3.2.2. Capital Investment

Example: Capital Investment[1]   Excel   PuLP   Gurobipy

3.2.3. Work Scheduling

3.2.4. Assignment Problem

Consider the assignment of n teams to n projects, where each team ranks the projects, where their favorite project is given a rank of n, their next favorite $n-1$, and their least favorite project is given a rank of 1. The assignment problem is formulated as follows (we denote ranks using the R-parameter):

Variables:
x_{ij} : 1 if project i assigned to team j, else 0.

$$\text{Max } z = \sum_{i=1}^{n} \sum_{j=1}^{n} R_{ij} x_{ij}$$

$$\text{s.t. } \sum_{i=1}^{n} x_{ij} = 1, \quad \forall j = 1, \cdots, n$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad \forall i = 1, \cdots, n$$

$$x_{ij} \geq 0, \quad \forall i = 1, \cdots, n, j = 1, \cdots, n.$$

Example: Hiring for tasks   Excel   PuLP   Gurobipy

In this assignment problem, we need to hire three people (Person 1, Person 2, Person 3) to three tasks (Task 1, Task 2, Task 3). In the table below, we list the cost of hiring each person for each task, in dollars. Since each person has a different cost for each task, we must make an assignment to minimize our total cost.

Cost	Task 1	Task 2	Task 3
Person 1	40	47	80
Person 2	72	36	58
Person 3	24	61	71

The assignment problem has an integrality property, such that if we remove the binary restriction on the x variables (now just non-negative, i.e., $x_{ij} \geq 0$) then we still get binary assignments, despite the fact that it is now an LP. This property is very interesting and useful. Of course, the objective function might not quite what we want, we might be interested ensuring that the team with the worst assignment is as good as possible (a fairness criteria). One way of doing this is to modify the assignment problem using a max-min objective:

Max-min Assignment-like Formulation

$$\text{Max} \quad z$$
$$\text{s.t.} \quad \sum_{i=1}^{n} x_{ij} = 1, \quad \forall j = 1, \cdots, n$$
$$\sum_{j=1}^{n} x_{ij} = 1, \quad \forall i = 1, \cdots, n$$
$$x_{ij} \geq 0, \quad \forall i = 1, \cdots, n, J = 1, \cdots, n$$
$$z \leq \sum_{i=1}^{n} R_{ij} x_{ij}, \quad \forall j = 1, \cdots, n.$$

Does this formulation have the integrality property (it is not an assignment problem)? Consider a very simple example where two teams are to be assigned to two projects and the teams give the projects the following rankings: Both teams prefer Project 2. For both problems, if we remove the binary restriction on

	Project 1	Project 2
Team 1	2	1
Team 2	2	1

the x-variable, they can take values between (and including) zero and one. For the assignment problem the optimal solution will have $z = 3$, and fractional x-values will not improve z. For the max-min assignment problem this is not the case, the optimal solution will have $z = 1.5$, which occurs when each team is assigned half of each project (i.e., for Team 1 we have $x_{11} = 0.5$ and $x_{21} = 0.5$).

3.2.5. Multi period Models

3.2.5.1. Production Planning

3.2.5.2. Crop Planning

3.2.6. Mixing Problems
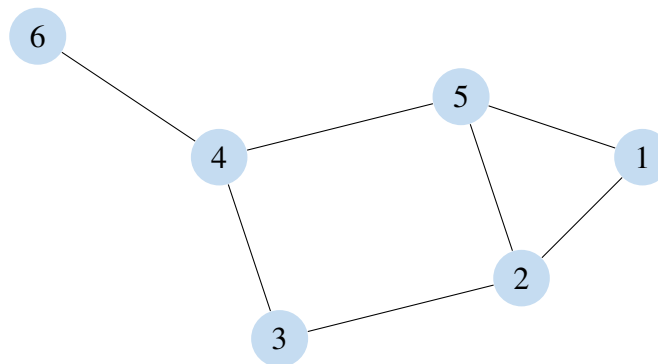
3.2.7. Financial Planning

3.2.8. Network Flow

To begin a discussion on Network flow, we first need to discuss graphs.

3.2.8.1. Graphs

A graph $G = (V, E)$ is defined by a set of vertices V and a set of edges E that contains pairs of vertices.

For example, the following graph G can be described by the vertex set $V = \{1, 2, 3, 4, 5, 6\}$ and the edge set $E = \{(4, 6), (4, 5), (5, 1)(1, 2), (2, 5), (2, 3), (3, 4)\}$.

In an undirected graph, we do not distinguish the direction of the edge. That is, for two vertices $i, j \in V$, we can equivalently write (i, j) or (j, i) to represent the edge.

Alternatively, we will want to consider directed graphs. We denote these as $G = (V, \mathscr{A})$ where $\mathscr{A}$ is a set of arcs where an arc is a directed edge.
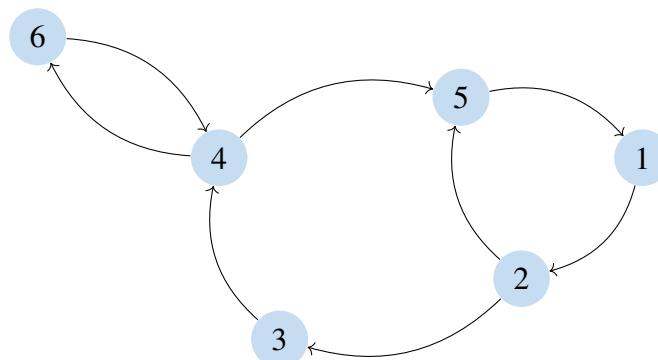
For example, the following directed graph G can be described by the vertex set $V = \{1, 2, 3, 4, 5, 6\}$ and the edge set $\mathscr{A} = \{(4, 6), (6, 4), (4, 5), (5, 1)(1, 2), (2, 5), (2, 3), (3, 4)\}$.

SETS A finite network G is described by a finite set of vertices V and a finite set $\mathscr{A}$ of arcs. Each arc (i, j) has two key attributes, namely its tail $j \in V$ and its head $i \in V$.

We think of a (single) commodity as being allowed to "flow" along each arc, from its tail to its head.

VARIABLES Indeed, we have "flow" variables

$$x_{ij} := \text{ amount of flow on } \operatorname{arc}(i, j) \text{ from vertex } i \text{ to vertex } j,$$

for all $(i, j) \in \mathscr{A}$.

3.2.8.2. Maximum Flow Problem

$$
\begin{array}{lll}
\max & \displaystyle\sum_{(s,i)\in\mathscr{A}} x_{si} & \text{max total flow from source} \quad (3.1)\\[3ex]
\text{s.t.} & \displaystyle\sum_{i:(i,v)\in\mathscr{A}} x_{iv} - \sum_{j:(v,j)\in\mathscr{A}} x_{vj} = 0 & v \in V\setminus\{s,t\} \quad (3.2)\\[3ex]
& 0 \le x_{ij} \le u_{ij} \quad \forall (i,j) \in \mathscr{A} & (3.3)
\end{array}
$$

SHORTEST PATH PROBLEM

$$
\begin{array}{ll}
\text{minimize} & \displaystyle\sum_{u\to v} \ell_{u\to v}\cdot x_{u\to v}\\[3ex]
\text{subject to } \displaystyle\sum_{u} x_{u\to s} - \sum_{w} x_{s\to w} & = 1\\[3ex]
\displaystyle\sum_{u} x_{u\to t} - \sum_{w} x_{t\to w} = -1 &\\[3ex]
\displaystyle\sum_{u} x_{u\to v} - \sum_{w} x_{v\to w} = 0 & \text{for every vertex } v \neq s,t\\[3ex]
x_{u\to v} \ge 0 & \text{for every edge } u \to v
\end{array}
$$

Or maybe write it like this:

$$\min \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{3.4}$$

$$\text{s.t.} \sum_{(i,j)\in\delta^+(i)} x_{ij} = 0 \forall i \in V \setminus \{s,t\} \tag{3.5}$$

$$\sum_{(i,j)\in\delta^+(s)} x_{ij} = -1 \tag{3.6}$$

$$\sum_{(i,j)\in\delta^+(t)} x_{ij} = 1 \tag{3.7}$$

$$x_{ij} \in \{0,1\} \forall (i,j) \in A, \tag{3.8}$$

Example 3.4: Max flow example
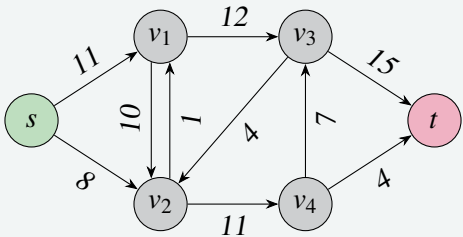
Example 3.5: Min Cost Network Flow
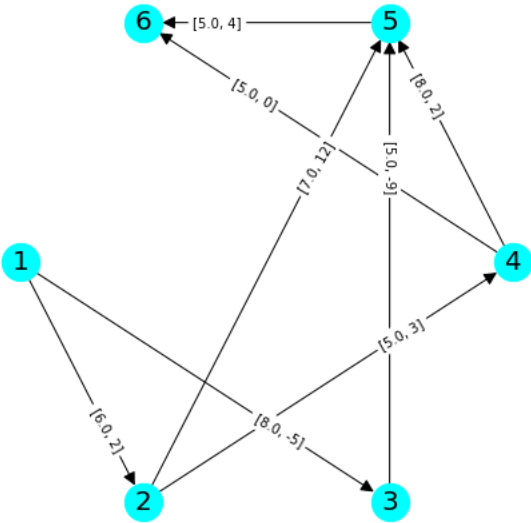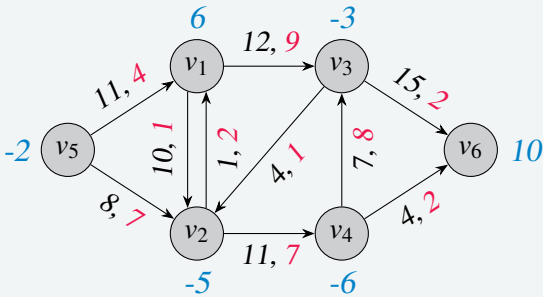
© network-flow[2]
Figure 3.1: network-flow

[2]**network-flow**, from **network-flow**. **network-flow**, **network-flow**.

[3]**network-flow-solution**, from **network-flow-solution**. **network-flow-solution**, **network-flow-solution**.
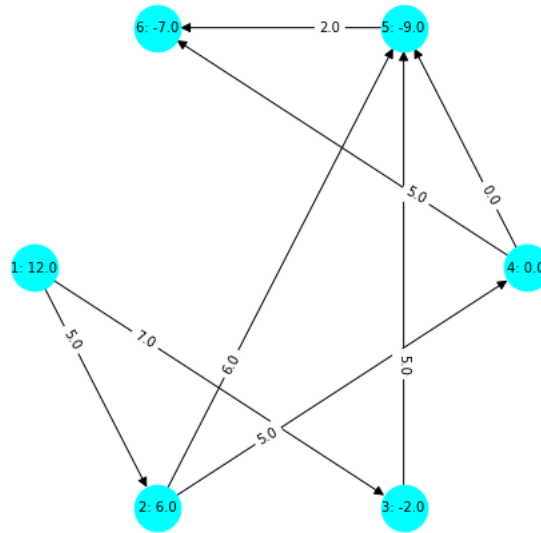
© network-flow-solution[3]
Figure 3.2: network-flow-solution

3.2.8.3. Minimum Cost Network Flow

PARAMETERS   We assume that flow on arc (i, j) should be non-negative and should not exceed

$$u_{ij} := \text{ the flow upper bound on arc}(i, j),$$

for $(i, j) \in \mathscr{A}$. Associated with each arc (i, j) is a cost

$$c_{ij} := \text{ cost per-unit-flow on arc } (i, j),$$

for $(i, j) \in \mathscr{A}$. The (total) cost of the flow x is defined to be

$$\sum_{(i,j)\in\mathscr{A}} c_{ij}x_{ij}.$$

We assume that we have further data for the nodes. Namely,

$$b_v := \text{ the net supply at node } v,$$

for $v \in V$.

A flow is conservative if the net flow out of node v, minus the net flow into node v, is equal to the net supply at node v, for all nodes $v \in V$.

The (single-commodity min-cost) network-flow problem is to find a minimumcost conservative flow that is non-negative and respects the flow upper bounds on the arcs.

OBJECTIVE AND CONSTRAINTS   We can formulate this as follows:

$$\min \sum_{(i,j)\in\mathscr{A}} c_{ij}x_{ij} \qquad\qquad\qquad \text{minimize cost}$$

$$\sum_{(i,v)\in\mathscr{A}} x_{iv} - \sum_{(v,i)\in\mathscr{A}} x_{vi} = b_v, \quad \text{for all } v \in V, \quad \text{flow conservation}$$

$$0 \le x_{ij} \le u_{ij}, \quad \text{for all } (i,j) \in \mathscr{A}.$$

Theorem 3.6: Integrality of Network Flow

If the capacities and demands are all integer values, then there always exists an optimal solution to the LP that has integer values.

3.2.9. Multi-Commodity Network Flow

In the same vein as the Network Flow Problem

$$\min \sum_{k=1}^{K} \sum_{e\in\mathscr{A}} c_e^k x_e^k$$

$$\sum_{e\in\mathscr{A}\ :\ t(e)=v} x_e^k - \sum_{e\in\mathscr{A}\ :\ h(e)=v} x_e^k = b_v^k, \ \text{ for } v \in \mathscr{N},\ k = 1,2,\ldots,K;$$

$$\sum_{k=0}^{K} x_e^k \le u_e, \ \text{ for } e \in \mathscr{A};$$

$$x_e^k \ge 0, \ \text{ for } e \in \mathscr{A},\ k = 1,2,\ldots,K$$

Notes:

K=1 is ordinary single-commodity network flow. Integer solutions for free when node-supplies and arc capacities are integer. K=2 example below with integer data gives a fractional basic optimum. This example doesn't have any feasible integer flow at all.

Remark 3.7

Unfortunately, the same integrality theorem does not hold in the multi-commodity network flow probem. Nontheless, if the quanties in each flow are very large, then the LP solution will likely be very close to an integer valued solution.

3.3 Modeling Tricks

3.3.1. 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

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

Example 3.8: Minimizing an Absolute Value

Note that
$$|t| = \max(t,-t),$$

Thus, if we need to minimize $|t|$ we can instead write

$$
\begin{aligned}
&\min z && (3.1)\\
&s.t. && (3.2)\\
&t \leq z - t \leq z && (3.3)
\end{aligned}
$$

3.4 Other examples

Food manfacturing - GUROBI

Optimization Methods in Finance - Corneujols, Tütüncü

4. Graphically Solving Linear Programs

Outcomes

    A. *Learn how to plot the feasible region and the objective function.*
    B. *Identify and compute extreme points of the feasible region.*
    C. *Find the optimal solution(s) to a linear program graphically.*
    D. *Classify the type of result of the problem as infeasible, unbounded, unique optimal solution, or infinitely many optimal solutions.*

Linear Programs (LP's) with two variables can be solved graphically by plotting the feasible region along with the level curves of the objective function.[1] We will show that we can find a point in the feasible region that maximizes the objective function using the level curves of the objective function.

We will begin with an easy example that is bounded and investigate the structure of the feasible region. We will then explore other examples.

4.1 Nonempty and Bounded Problem

Consider the problem

$$
\begin{aligned}
\max \quad & 2X + 5Y \\
\text{s.t.} \quad & X + 2Y \leq 16 \\
& 5X + 3Y \leq 45 \\
& X, Y \geq 0
\end{aligned}
$$

We want to start by plotting the *feasible region*, that is, the set points (X,Y) that satisfy all the constraints.

We can plot this by first plotting the four lines

- $X + 2Y = 16$

- $5X + 3Y = 45$

- $X = 0$

- $Y = 0$

---

[1]Special thanks to Joshua Emmanual and Christopher Griffin for sharing their content to help put this section together. Proper citations and referenes are forthcoming.

and then shading in the side of the space cut out by the corresponding inequality.

The resulting feasible region can then can be shaded in as the region that satisfies all the inequalties.

$X + 2Y \le 16$

X	Y
0	8
16	0

$5X + 3Y \le 45$

X	Y
0	15
9	0
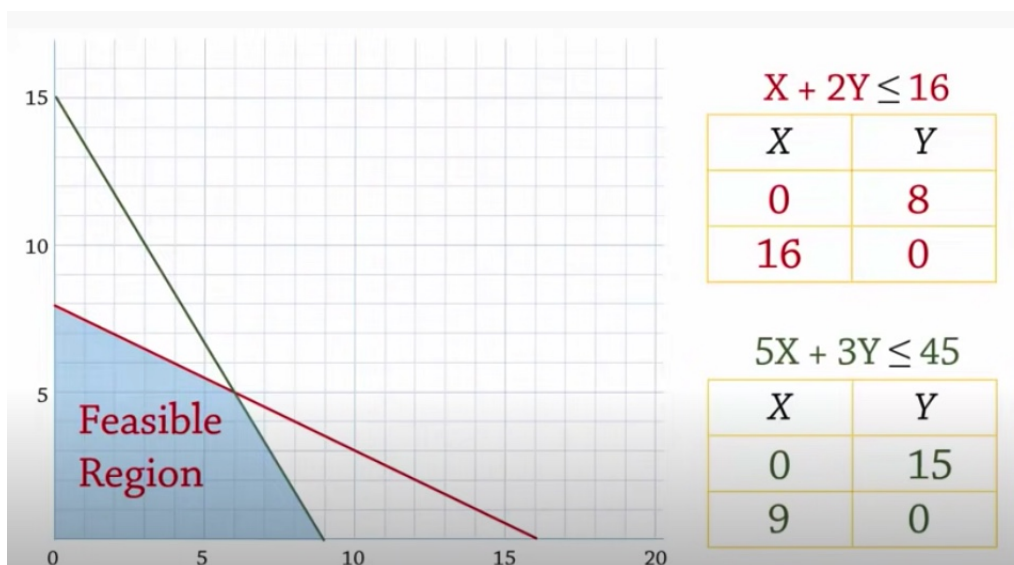
Notice that the feasible region is nonempty (it has points that satisfy all the inequalities) and also that it is bounded (the feasible points dont continue infinitly in any direction).

We want to identify the *extreme points* (i.e., the corners) of the feasible region. Understanding these points will be critical to understanding the optimal solutions of the model. Notice that all extreme points can be computed by finding the intersection of 2 of the lines. But! Not all intersections of any two lines are feasible.

We will later use the terminology *basic feasible solution* for an extreme point of the feasible region, and *basic solution* as a point that is the intersection of 2 lines, but is actually infeasible (does not satisfy all the constraints).

Point	$Z = 2X + 5Y$
❶ $(0, 0)$	0
❷ $(0, 8)$	40
❸ $(6, 5)$	37
❹ $(9, 0)$	18
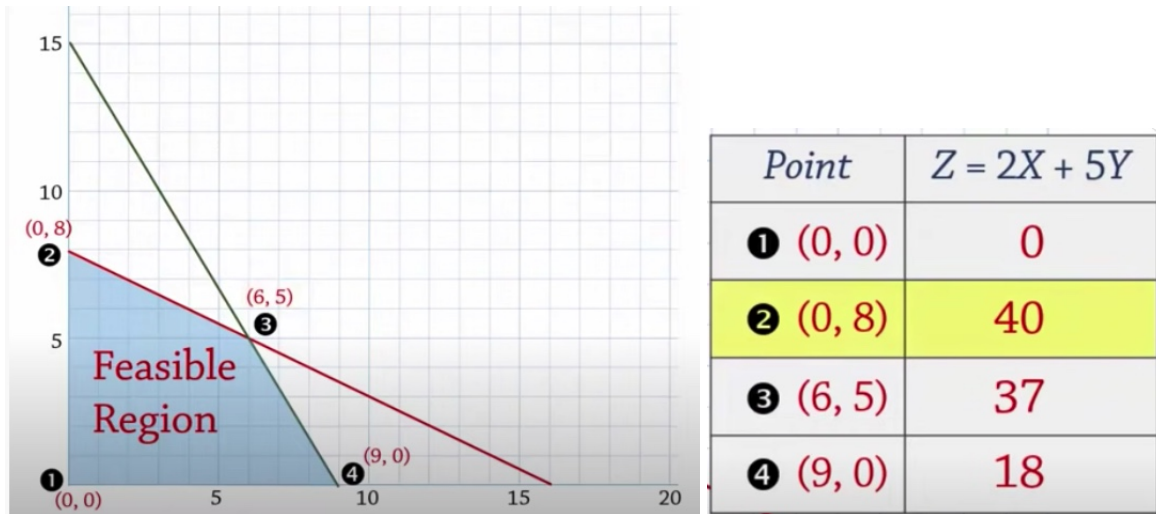
Theorem 4.1: Optimal Extreme Point

If the feasible region is nonempty and bounded, then there exists an optimal solution at an extreme point of the feasible region.

We will explore why this theorem is true, and also what happens when the feasible region does not satisfy the assumtions of either nonempty or bounded. We illustrate the idea first using the problem from Example 1.

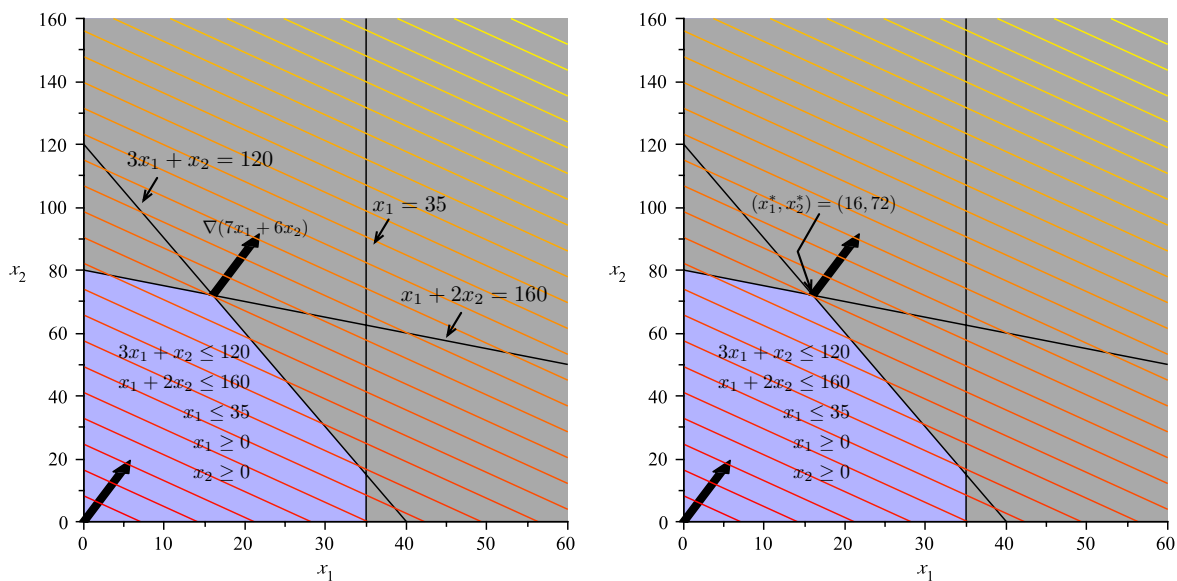**Figure 4.1: Feasible Region and Level Curves of the Objective Function: The shaded region in the plot is the feasible region and represents the intersection of the five inequalities constraining the values of x_1 and x_2. On the right, we see the optimal solution is the "last" point in the feasible region that intersects a level set as we move in the direction of increasing profit.**

> **Example 4.2: Continuation of Example**
>
> *Let's continue the example of the Toy Maker begin in Example 1. Solve this problem graphically.*

Solution. To solve the linear programming problem graphically, begin by drawing the feasible region. This is shown in the blue shaded region of Figure 4.1.

After plotting the feasible region, the next step is to plot the level curves of the objective function. In our problem, the level sets will have the form:

$$7x_1 + 6x_2 = c \implies x_2 = \frac{-7}{6}x_1 + \frac{c}{6}$$

This is a set of parallel lines with slope $-7/6$ and intercept $c/6$ where c can be varied as needed. The level curves for various values of c are parallel lines. In Figure 4.1 they are shown in colors ranging from red to yellow depending upon the value of c. Larger values of c are more yellow.

To solve the linear programming problem, follow the level sets along the gradient (shown as the black arrow) until the last level set (line) intersects the feasible region. If you are doing this by hand, you can draw a single line of the form $7x_1 + 6x_2 = c$ and then simply draw parallel lines in the direction of the gradient $(7,6)$. At some point, these lines will fail to intersect the feasible region. The last line to intersect the feasible region will do so at a point that maximizes the profit. In this case, the point that maximizes $z(x_1, x_2) = 7x_1 + 6x_2$, subject to the constraints given, is $(x_1^*, x_2^*) = (16, 72)$.

Note the point of optimality $(x_1^*, x_2^*) = (16, 72)$ is at a corner of the feasible region. This corner is formed by the intersection of the two lines: $3x_1 + x_2 = 120$ and $x_1 + 2x_2 = 160$. In this case, the constraints

$$3x_1 + x_2 \leq 120$$
$$x_1 + 2x_2 \leq 160$$

are both *binding*, while the other constraints are non-binding. In general, we will see that when an optimal solution to a linear programming problem exists, it will always be at the intersection of several binding constraints; that is, it will occur at a corner of a higher-dimensional polyhedron. ♠

We can now define an algorithm for identifying the solution to a linear programing problem in two variables with a *bounded* feasible region (see Algorithm 1):

Algorithm 1 Algorithm for Solving a Two Variable Linear Programming Problem Graphically–Bounded Feasible Region, Unique Solution Case

Algorithm for Solving a Linear Programming Problem Graphically

Bounded Feasible Region, Unique Solution

1. Plot the feasible region defined by the constraints.
2. Plot the level sets of the objective function.
3. For a maximization problem, identify the level set corresponding the greatest (least, for minimization) objective function value that intersects the feasible region. This point will be at a corner.
4. The point on the corner intersecting the greatest (least) level set is a solution to the linear programming problem.

The example linear programming problem presented in the previous section has a single optimal solution. In general, the following outcomes can occur in solving a linear programming problem:

1. The linear programming problem has a unique solution. (We've already seen this.)
2. There are infinitely many alternative optimal solutions.
3. There is no solution and the problem's objective function can grow to positive infinity for maximization problems (or negative infinity for minimization problems).
4. There is no solution to the problem at all.

Case 3 above can only occur when the feasible region is unbounded; that is, it cannot be surrounded by a ball with finite radius. We will illustrate each of these possible outcomes in the next four sections. We will prove that this is true in a later chapter.

4.2 Infinitely Many Optimal Solutions

It can happen that there is more than one solution. In fact, in this case, there are infinitely many optimal solutions. We'll study a specific linear programming problem with an infinite number of solutions by modifying the objective function in Example 1.

Example 4.3: Toy Maker Alternative Solutions

Suppose the toy maker in Example 1 finds that it can sell planes for a profit of $18 each instead of $7 each. The new linear programming problem becomes:

$$\begin{cases} \max \ z(x_1,x_2) = 18x_1 + 6x_2 \\ \text{s.t.} \ 3x_1 + x_2 \leq 120 \\ \qquad x_1 + 2x_2 \leq 160 \\ \qquad x_1 \leq 35 \\ \qquad x_1 \geq 0 \\ \qquad x_2 \geq 0 \end{cases} \qquad (4.1)$$

Solution. Applying our graphical method for finding optimal solutions to linear programming problems yields the plot shown in Figure 4.2. The level curves for the function $z(x_1,x_2) = 18x_1 + 6x_2$ are *parallel* to one face of the polygon boundary of the feasible region. Hence, as we move further up and to the right in the direction of the gradient (corresponding to larger and larger values of $z(x_1,x_2)$) we see that there is not *one* point on the boundary of the feasible region that intersects that level set with greatest value, but instead a side of the polygon boundary described by the line $3x_1 + x_2 = 120$ where $x_1 \in [16,35]$. Let:

$$S = \{(x_1,x_2) | 3x_1 + x_2 \leq 120, \ x_1 + 2x_2 \leq 160, \ x_1 \leq 35, \ x_1,x_2 \geq 0\}$$

that is, S is the feasible region of the problem. Then for any value of $x_1^* \in [16,35]$ and any value x_2^* so that $3x_1^* + x_2^* = 120$, we will have $z(x_1^*,x_2^*) \geq z(x_1,x_2)$ for all $(x_1,x_2) \in S$. Since there are infinitely many values that x_1 and x_2 may take on, we see this problem has an infinite number of alternative optimal solutions.
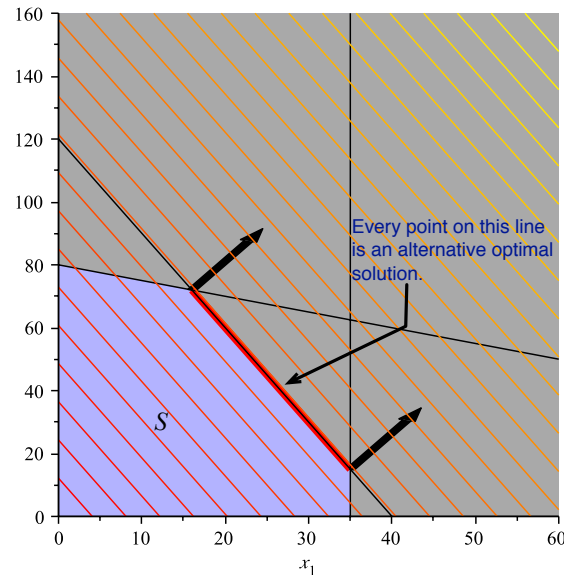
Figure 4.2: **An example of infinitely many alternative optimal solutions in a linear programming problem. The level curves for** $z(x_1,x_2) = 18x_1 + 6x_2$ **are** *parallel* **to one face of the polygon boundary of the feasible region. Moreover, this side contains the points of greatest value for** $z(x_1,x_2)$ **inside the feasible region. Any combination of** (x_1,x_2) **on the line** $3x_1 + x_2 = 120$ **for** $x_1 \in [16,35]$ **will provide the largest possible value** $z(x_1,x_2)$ **can take in the feasible region** S**.**

♠

Exercise 4.4

Use the graphical method for solving linear programming problems to solve the linear programming problem you defined in Exercise 3.

Based on the example in this section, we can modify our algorithm for finding the solution to a linear programming problem graphically to deal with situations with an infinite set of alternative optimal solutions (see Algorithm 2):

Algorithm 2 Algorithm for Solving a Two Variable Linear Programming Problem Graphically–Bounded Feasible Region Case

Algorithm for Solving a Linear Programming Problem Graphically

Bounded Feasible Region

1. Plot the feasible region defined by the constraints.
2. Plot the level sets of the objective function.
3. For a maximization problem, identify the level set corresponding the greatest (least, for minimization) objective function value that intersects the feasible region. This point will be at a corner.
4. The point on the corner intersecting the greatest (least) level set is a solution to the linear programming problem.
5. **If the level set corresponding to the greatest (least) objective function value is parallel to a side of the polygon boundary next to the corner identified, then there are infinitely many alternative optimal solutions and any point on this side may be chosen as an optimal solution.**

> **Exercise 4.5**
>
> *Modify the linear programming problem from Exercise 3 to obtain a linear programming problem with an infinite number of alternative optimal solutions. Solve the new problem and obtain a description for the set of alternative optimal solutions. [Hint: Just as in the example, x_1 will be bound between two value corresponding to a side of the polygon. Find those values and the constraint that is binding. This will provide you with a description of the form for any $x_1^* \in [a,b]$ and x_2^* is chosen so that $cx_1^* + dx_2^* = v$, the point (x_1^*, x_2^*) is an alternative optimal solution to the problem. Now you fill in values for a, b, c, d and v.]*

4.3 Problems with No Solution

Recall for *any* mathematical programming problem, the feasible set or region is simply a subset of $\mathbb{R}^n$. If this region is empty, then there is no solution to the mathematical programming problem and the problem is said to be *over constrained*. In this case, we say that the problem is *infeasible*. We illustrate this case for linear programming problems with the following example.

> **Example 4.6: Infeasible Problem**
>
> *onsider the following linear programming problem:*
>
> $$\begin{cases} \max \ z(x_1, x_2) = 3x_1 + 2x_2 \\ s.t. \ \dfrac{1}{40}x_1 + \dfrac{1}{60}x_2 \leq 1 \\ \quad \dfrac{1}{50}x_1 + \dfrac{1}{50}x_2 \leq 1 \\ \quad x_1 \geq 30 \\ \quad x_2 \geq 20 \end{cases} \qquad (4.1)$$

Solution. The level sets of the objective and the constraints are shown in Figure 4.3.
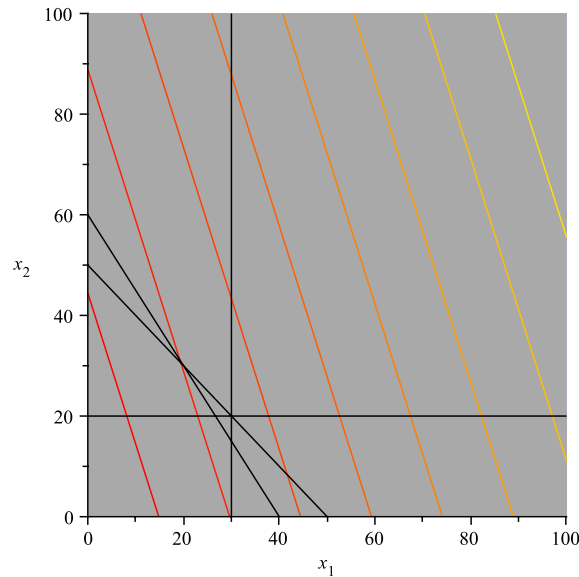
Figure 4.3: A Linear Programming Problem with no solution. The feasible region of the linear programming problem is empty; that is, there are no values for x_1 and x_2 that can simultaneously satisfy all the constraints. Thus, no solution exists.

The fact that the feasible region is empty is shown by the fact that in Figure 4.3 there is no blue region–i.e., all the regions are gray indicating that the constraints are not satisfiable. ♠

Based on this example, we can modify our previous algorithm for finding the solution to linear programming problems graphically (see Algorithm 3):

Algorithm 3 Algorithm for Solving a Two Variable Linear Programming Problem Graphically–Bounded Feasible Region Case

Algorithm for Solving a Linear Programming Problem Graphically

Bounded Feasible Region

1. Plot the feasible region defined by the constraints.
2. **If the feasible region is empty, then no solution exists.**
3. Plot the level sets of the objective function.
4. For a maximization problem, identify the level set corresponding the greatest (least, for minimization) objective function value that intersects the feasible region. This point will be at a corner.
5. The point on the corner intersecting the greatest (least) level set is a solution to the linear programming problem.
6. **If the level set corresponding to the greatest (least) objective function value is parallel to a side of the polygon boundary next to the corner identified, then there are infinitely many alternative optimal solutions and any point on this side may be chosen as an optimal solution.**

4.4 Problems with Unbounded Feasible Regions

Consider the problem

$$\min \quad Z = 5X + 7Y$$
$$\text{s.t.} \quad X + 3Y \geq 6$$
$$\quad 5X + 2Y \geq 10$$
$$\quad Y \leq 4$$
$$\quad X, Y \geq 0$$

As you can see, the feasible region is *unbounded*. In particular, from any point in the feasible region, one can always find another feasible point by increasing the X coordinate (i.e., move to the right in the picture). However, this does not necessarily mean that the optimization problem is unbounded.
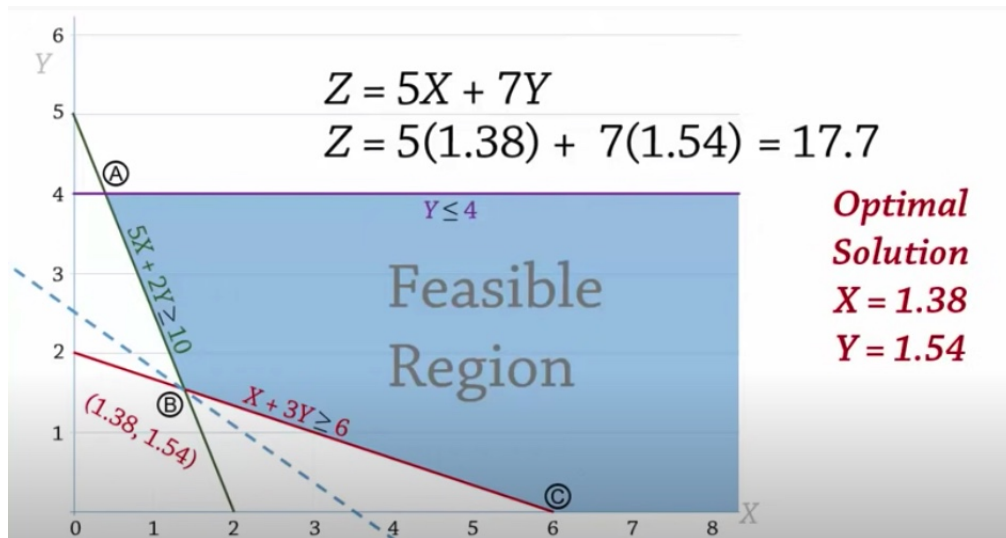
Indeed, the optimal solution is at the B, the extreme point in the lower left hand corner.

Consider however, if we consider a different problem where we try to maximize the objective

$$\begin{aligned} \max \quad & Z = 5X + 7Y \\ \text{s.t.} \quad & X + 3Y \geq 6 \\ & 5X + 2Y \geq 10 \\ & Y \leq 4 \\ & X, Y \geq 0 \end{aligned}$$

Solution. This optimization problem is unbounded! For example, notice that the point $(X,Y) = (n,0)$ is feasible for all $n = 1, 2, 3, \ldots,$. Then the objective function $Z = 5n + 0$ follows the sequence $5, 10, 15, \ldots,$ which diverges to infinity. ♠

Again, we'll tackle the issue of linear programming problems with unbounded feasible regions by illustrating the possible outcomes using examples.

Example 4.7

Consider the linear programming problem below:

$$\begin{cases} \max \ z(x_1, x_2) = 2x_1 - x_2 \\ \text{s.t. } x_1 - x_2 \leq 1 \\ \qquad 2x_1 + x_2 \geq 6 \\ \qquad x_1, x_2 \geq 0 \end{cases} \tag{4.1}$$

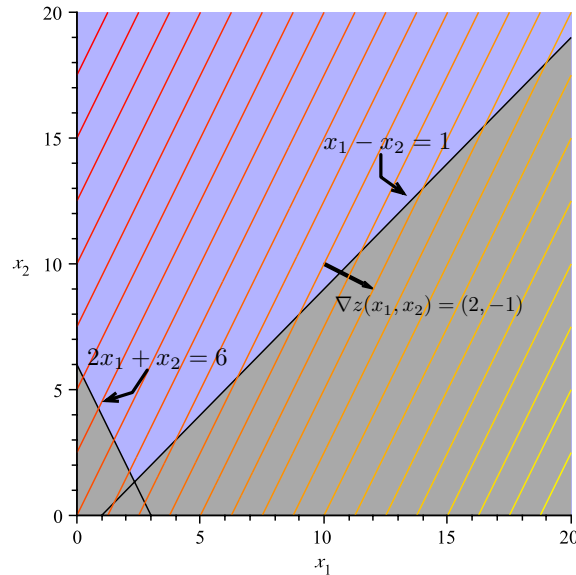**Solution.** The feasible region and level curves of the objective function are shown in Figure 4.4.

Figure 4.4: A Linear Programming Problem with Unbounded Feasible Region: Note that we can continue to make level curves of $z(x_1,x_2)$ corresponding to larger and larger values as we move down and to the right. These curves will continue to intersect the feasible region for any value of $v = z(x_1,x_2)$ we choose. Thus, we can make $z(x_1,x_2)$ as large as we want and still find a point in the feasible region that will provide this value. Hence, the optimal value of $z(x_1,x_2)$ subject to the constraints $+\infty$. That is, the problem is unbounded.

The feasible region in Figure 4.4 is clearly unbounded since it stretches upward along the x_2 axis infinitely far and also stretches rightward along the x_1 axis infinitely far, bounded below by the line $x_1 - x_2 = 1$. There is no way to enclose this region by a disk of finite radius, hence the feasible region is not bounded.

We can draw more level curves of $z(x_1,x_2)$ in the direction of increase (down and to the right) as long as we wish. There will always be an intersection point with the feasible region because it is infinite. That is, these curves will continue to intersect the feasible region for any value of $v = z(x_1,x_2)$ we choose. Thus, we can make $z(x_1,x_2)$ as large as we want and still find a point in the feasible region that will provide this value. Hence, the largest value $z(x_1,x_2)$ can take when (x_1,x_2) are in the feasible region is $+\infty$. That is, the problem is unbounded. ♠

Just because a linear programming problem has an unbounded feasible region does not imply that there is not a finite solution. We illustrate this case by modifying example 4.4.

> **Example 4.8: Continuation of Example**
>
> *Consider the linear programming problem from Example 4.4 with the new objective function:*

$z(x_1,x_2) = (1/2)x_1 - x_2$. *Then we have the new problem:*

$$
\begin{cases}
\max\ z(x_1,x_2) = \dfrac{1}{2}x_1 - x_2 \\[2mm]
s.t.\ x_1 - x_2 \le 1 \\[1mm]
\quad\quad 2x_1 + x_2 \ge 6 \\[1mm]
\quad\quad x_1, x_2 \ge 0
\end{cases}
\tag{4.2}
$$

Solution. The feasible region, level sets of $z(x_1,x_2)$ and gradients are shown in Figure 4.5. In this case note, that the direction of increase of the objective function is *away* from the direction in which the feasible region is unbounded (i.e., downward). As a result, the point in the feasible region with the largest $z(x_1,x_2)$ value is $(7/3, 4/3)$. Again this is a vertex: the binding constraints are $x_1 - x_2 = 1$ and $2x_1 + x_2 = 6$ and the solution occurs at the point these two lines intersect.
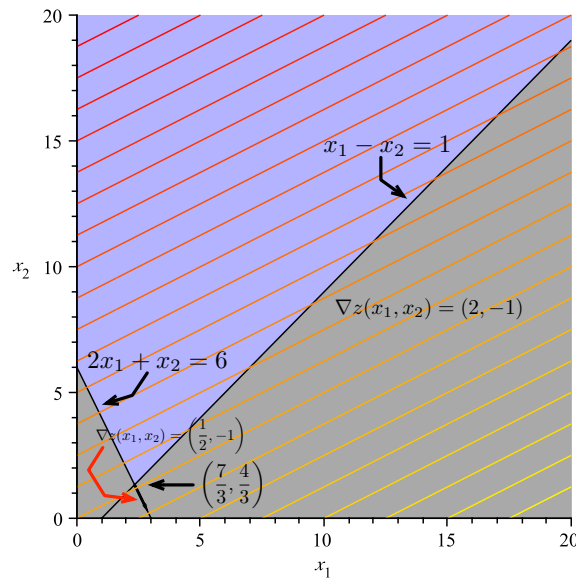
Figure 4.5: A Linear Programming Problem with Unbounded Feasible Region and Finite Solution: In this problem, the level curves of $z(x_1,x_2)$ increase in a more "southernly" direction that in Example 4.4–that is, *away* from the direction in which the feasible region increases without bound. The point in the feasible region with largest $z(x_1,x_2)$ value is $(7/3, 4/3)$. Note again, this is a vertex.

♠

Based on these two examples, we can modify our algorithm for graphically solving a two variable linear programming problems to deal with the case when the feasible region is unbounded.

Algorithm 4 Algorithm for Solving a Linear Programming Problem Graphically–Bounded and Unbounded Case

Algorithm for Solving a Two Variable Linear Programming Problem Graphically

1. Plot the feasible region defined by the constraints.
2. If the feasible region is empty, then no solution exists.
3. If the feasible region is unbounded goto Line 8. Otherwise, Goto Line 4.
4. Plot the level sets of the objective function.
5. For a maximization problem, identify the level set corresponding the greatest (least, for minimization) objective function value that intersects the feasible region. This point will be at a corner.
6. The point on the corner intersecting the greatest (least) level set is a solution to the linear programming problem.
7. **If the level set corresponding to the greatest (least) objective function value is parallel to a side of the polygon boundary next to the corner identified, then there are infinitely many alternative optimal solutions and any point on this side may be chosen as an optimal solution.**
8. (The feasible region is unbounded): Plot the level sets of the objective function.
9. If the level sets intersect the feasible region at larger and larger (smaller and smaller for a minimization problem), then the problem is unbounded and the solution is $+\infty$ ($-\infty$ for minimization problems).
10. Otherwise, identify the level set corresponding the greatest (least, for minimization) objective function value that intersects the feasible region. This point will be at a corner.
11. The point on the corner intersecting the greatest (least) level set is a solution to the linear programming problem. **If the level set corresponding to the greatest (least) objective function value is parallel to a side of the polygon boundary next to the corner identified, then there are infinitely many alternative optimal solutions and any point on this side may be chosen as an optimal solution.**

Exercise 4.9

Does the following problem have a bounded solution? Why?

$$\begin{cases} \min & z(x_1,x_2) = 2x_1 - x_2 \\ s.t. & x_1 - x_2 \le 1 \\ & 2x_1 + x_2 \ge 6 \\ & x_1, x_2 \ge 0 \end{cases} \tag{4.3}$$

[Hint: Use Figure 4.5 and Algorithm 4.]

Exercise 4.10

Modify the objective function in Example 4.4 or Example 4.4 to produce a problem with an infinite number of solutions.

Exercise 4.11

*Modify the objective function in Exercise 4.4 to produce a **minimization** problem that has a finite solution. Draw the feasible region and level curves of the objective to "prove" your example works. [Hint: Think about what direction of increase is required for the level sets of $z(x_1, x_2)$ (or find a trick using Exercise ??).]*

4.5 Formal Mathematical Statements

Vectors and Linear and Convex Combinations

Vectors: Vector **n** has n-elements and represents a point (or an arrow from the origin to the point, denoting a direction) in $\mathcal{R}^n$ space (Euclidean or real space). Vectors can be expressed as either row or column vectors.

Vector Addition: Two vectors of the same size can be added, componentwise, e.g., for vectors $\mathbf{a} = (2, 3)$ and $\mathbf{b} = (3, 2)$, $\mathbf{a} + \mathbf{b} = (2 + 3, 3 + 2) = (5, 5)$.

Scalar Multiplication: A vector can be multiplied by a scalar k (constant) component-wise. If $k > 0$ then this does not change the direction represented by the vector, it just scales the vector.

Inner or Dot Product: Two vectors of the same size can be multiplied to produce a real number. For example, $\mathbf{ab} = 2 * 3 + 3 * 2 = 10$.

Linear Combination: The vector **b** is a **linear combination** of $\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_k$ if $\mathbf{b} = \sum_{i=1}^{k} \lambda_i \mathbf{a}_i$ for $\lambda_1, \lambda_2, \cdots, \lambda_k \in \mathcal{R}$. If $\lambda_1, \lambda_2, \cdots, \lambda_k \in \mathcal{R}_{\geq 0}$ then **b** is a *non-negative linear combination* of $\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_k$.

Convex Combination: The vector **b** is a **convex combination** of $\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_k$ if $\mathbf{b} = \sum_{i=1}^{k} \lambda_i \mathbf{a}_i$, for $\lambda_1, \lambda_2, \cdots, \lambda_k \in \mathcal{R}_{\geq 0}$ and $\sum_{i=1}^{k} \lambda_i = 1$. For example, any convex combination of two points will lie on the line segment between the points.

Linear Independence: Vectors $\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_k$ are *linearly independent* if the following linear combination $\sum_{i=1}^{k} \lambda_i \mathbf{a}_i = 0$ implies that $\lambda_i = 0$, $i = 1, 2, \cdots, k$. In $\mathcal{R}^2$ two vectors are only linearly dependent if they lie on the same line. Can you have three linearly independent vectors in $\mathcal{R}^2$?

Spanning Set: Vectors $\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_k$ span $\mathcal{R}^m$ is any vector in $\mathcal{R}^m$ can be represented as a linear combination of $\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_k$, i.e., $\sum_{i=1}^{m} \lambda_i \mathbf{a_i}$ can represent any vector in $\mathcal{R}^m$.

Basis: Vectors $\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_k$ form a basis of $\mathcal{R}^m$ if they span $\mathcal{R}^m$ and any smaller subset of these vectors does not span $\mathcal{R}^m$. Vectors $\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_k$ can only form a basis of $\mathcal{R}^m$ if $k = m$ and they are linearly independent.

Convex and Polyhedral Sets

Convex Set: Set $\mathscr{S}$ in $\mathscr{R}^n$ is a *convex set* if a line segment joining any pair of points $\mathbf{a}_1$ and $\mathbf{a}_2$ in $\mathscr{S}$ is completely contained in $\int$, that is, $\lambda\mathbf{a}_1 + (1-\lambda)\mathbf{a}_2 \in \mathscr{S}, \forall \lambda \in [0,1]$.

Hyperplanes and Half-Spaces: A hyperplane in $\mathscr{R}^n$ divides $\mathscr{R}^n$ into 2 half-spaces (like a line does in $\mathscr{R}^2$). A hyperplane is the set $\{\mathbf{x} : \mathbf{px} = k\}$, where $\mathbf{p}$ is the gradient to the hyperplane (i.e., the coefficients of our linear expression). The corresponding half-spaces is the set of points $\{\mathbf{x} : \mathbf{px} \geq k\}$ and $\{\mathbf{x} : \mathbf{px} \leq k\}$.

Polyhedral Set: A *polyhedral set* (or polyhedron) is the set of points in the intersection of a finite set of half-spaces. Set $\mathscr{S} = \{\mathbf{x} : \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \geq 0\}$, where $\mathbf{A}$ is an $m \times n$ matrix, $\mathbf{x}$ is an n-vector, and $\mathbf{b}$ is an m-vector, is a *polyhedral set* defined by $m+n$ hyperplanes (i.e., the intersection of $m+n$ half-spaces).

- Polyhedral sets are convex.

- A polytope is a bounded polyhedral set.

- A polyhedral cone is a polyhedral set where the hyperplanes (that define the half-spaces) pass through the origin, thus $\mathscr{C} = \{\mathbf{x} : \mathbf{Ax} \leq 0\}$ is a polyhedral cone.

Edges and Faces: An *edge* of a polyhedral set $\mathscr{S}$ is defined by $n-1$ hyperplanes, and a *face* of $\mathscr{S}$ by one of more defining hyperplanes of $\mathscr{S}$, thus an extreme point and an edge are faces (an extreme point is a zero-dimensional face and an edge a one-dimensional face). In $\mathscr{R}^2$ faces are only edges and extreme points, but in $\mathscr{R}^3$ there is a third type of face, and so on...

Extreme Points: $\mathbf{x} \in \mathscr{S}$ is an extreme point of $\mathscr{S}$ if:

Definition 1: $\mathbf{x}$ is not a convex combination of two other points in $\mathscr{S}$, that is, all line segments that are completely in $\mathscr{S}$ that contain $\mathbf{x}$ must have $\mathbf{x}$ as an endpoint.

Definition 2: $\mathbf{x}$ lies on n linearly independent defining hyperplanes of $\mathscr{S}$.

If more than n hyperplanes pass through an extreme points then it is a degenerate extreme point, and the polyhedral set is considered degenerate. This just adds a bit of complexity to the algorithms we will study, but it is quite common.

Unbounded Sets:

Rays: A ray in $\mathscr{R}^n$ is the set of points $\{\mathbf{x} : \mathbf{x}_0 + \lambda\mathbf{d}, \lambda \geq 0\}$, where $\mathbf{x}_0$ is the vertex and $\mathbf{d}$ is the direction of the ray.

Convex Cone: A *Convex Cone* is a convex set that consists of rays emanating from the origin. A convex cone is completely specified by its extreme directions. If $\mathscr{C}$ is convex cone, then for any $\mathbf{x} \in \mathscr{C}$ we have
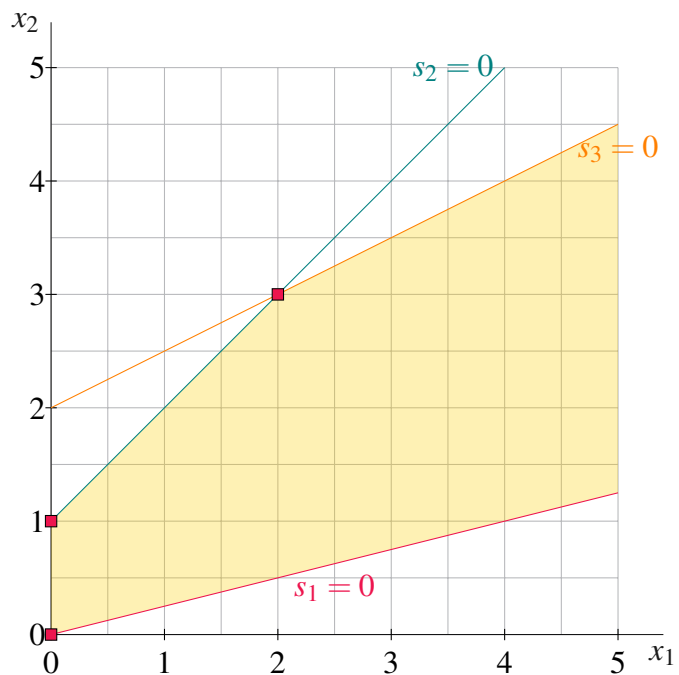
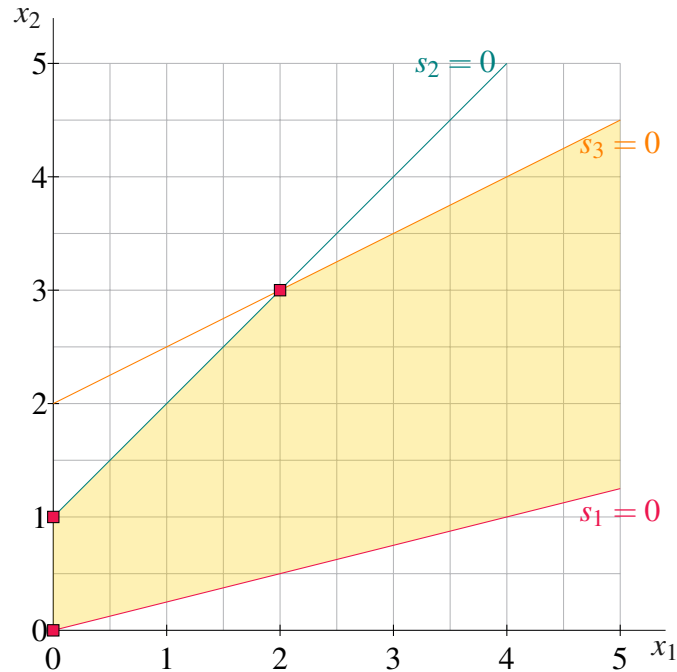$\lambda \mathbf{x} \in \mathscr{C}, \lambda \geq 0.$

Unbounded Polyhedral Sets: If $\mathscr{S}$ is unbounded, it will have *directions*. $\mathbf{d}$ is a direction of $\mathscr{S}$ only if $\mathbf{Ax} + \lambda \mathbf{d} \leq \mathbf{b}, \mathbf{x} + \lambda \mathbf{d} \geq 0$ for all $\lambda \geq 0$ and all $\mathbf{x} \in \mathscr{S}$. In other words, consider the ray $\{\mathbf{x} : \mathbf{x}_0 + \lambda \mathbf{d}, \lambda \geq 0\}$ in $\mathscr{R}^n$, where $\mathbf{x}_0$ is the vertex and $\mathbf{d}$ is the direction of the ray. $\mathbf{d} \neq 0$ is a **direction** of set $\mathscr{S}$ if for each $\mathbf{x}_0$ in $\mathscr{S}$ the ray $\{\mathbf{x}_0 + \lambda \mathbf{d}, \lambda \geq 0\}$ also belongs to $\mathscr{S}$.

Extreme Directions: An *extreme direction* of $\mathscr{S}$ is a direction that *cannot* be represented as positive linear combination of other directions of $\mathscr{S}$. A non-negative linear combination of extreme directions can be used to represent all other directions of $\mathscr{S}$. A polyhedral cone is completely specified by its extreme directions.

Let's define a procedure for finding the extreme directions, using the following LP's feasible region. Graphically, we can see that the extreme directions should follow the the $s_1 = 0$ (red) line and the $s_3 = 0$ (orange) line.

$$\max \ z = -5x_1 - x_2$$
$$\text{s.t. } x_1 - 4x_2 + s_1 = 0$$
$$-x_1 + x_2 + s_2 = 1$$
$$-x_1 + 2x_2 + s_3 = 4$$
$$x_1, x_2, s_1, s_2, s_3 \geq 0.$$

E.g., consider the $s_3 = 0$ (orange) line, to find the extreme direction start at extreme point (2,3) and find another feasible point on the orange line, say (4,4) and subtract (2,3) from (4,4), which yields (2,1).

This is related to the slope in two-dimensions, as discussed in class, the rise is 1 and the run is 2. So this direction has a slope of 1/2, but this does not carry over easily to higher dimensions where directions cannot be defined by a single number.

To find the extreme directions we can change the right-hand-side to $\mathbf{b} = 0$, which forms a polyhedral cone (in yellow), and then add the constraint $x_1 + x_2 = 1$. The intersection of the cone and $x_1 + x_2 = 1$ form a line segment.

$$\max \; z = -5x_1 - x_2$$
$$\text{s.t.} \; x_1 - 4x_2 + s_1 = 0$$
$$-x_1 + x_2 + s_2 = 0$$
$$-x_1 + 2x_2 + s_3 = 0$$
$$x_1 + x_2 = 1$$
$$x_1, x_2, s_1, s_2, s_3 \geq 0.$$

Magnifying for clarity, and removing the $s_2 = 0$ (teal) line, as it is redundant, and marking the extreme points of the new feasible region, (4/5, 1/5) and (2/3, 1/3), with red boxes, we have:
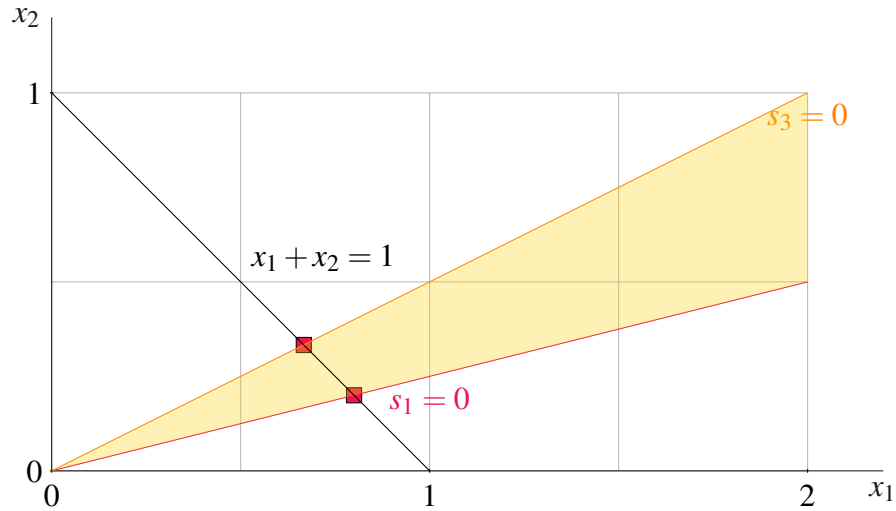
The extreme directions are thus (4/5, 1/5) and (2/3, 1/3).

Representation Theorem: Let $\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x_k}$ be the set of extreme points of $\mathscr{S}$, and if $\mathscr{S}$ is unbounded, $\mathbf{d}_1, \mathbf{d}_2, \cdots \mathbf{d_l}$ be the set of extreme directions. Then any $\mathbf{x} \in \mathscr{S}$ is equal to a convex combination of the extreme points and a non-negative linear combination of the extreme directions: $\mathbf{x} = \sum_{j=1}^{k} \lambda_j \mathbf{x_j} + \sum_{j=1}^{l} \mu_j \mathbf{d_j}$, where $\sum_{j=1}^{k} \lambda_j = 1$, $\lambda_j \geq 0$, $\forall j = 1, 2, \cdots, k$, and $\mu_j \geq 0$, $\forall j = 1, 2, \cdots, l$.

$$
\begin{aligned}
\max\ &z = -5x_1 - x_2 \\
\text{s.t.}\ &x_1 - 4x_2 + s_1 = 0 \\
&-x_1 + x_2 + s_2 = 1 \\
&-x_1 + 2x_2 + s_3 = 4 \\
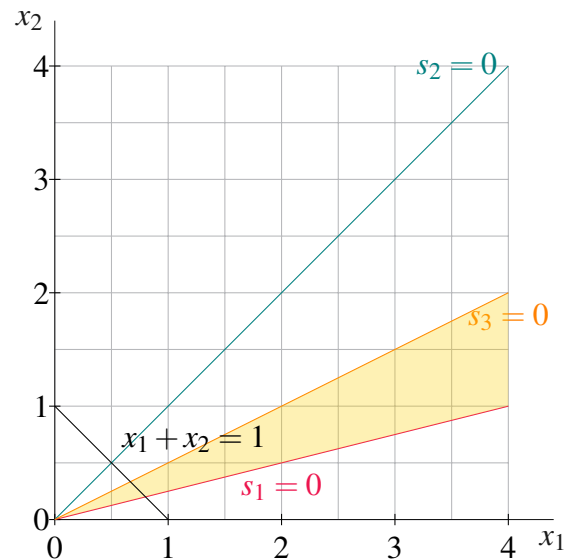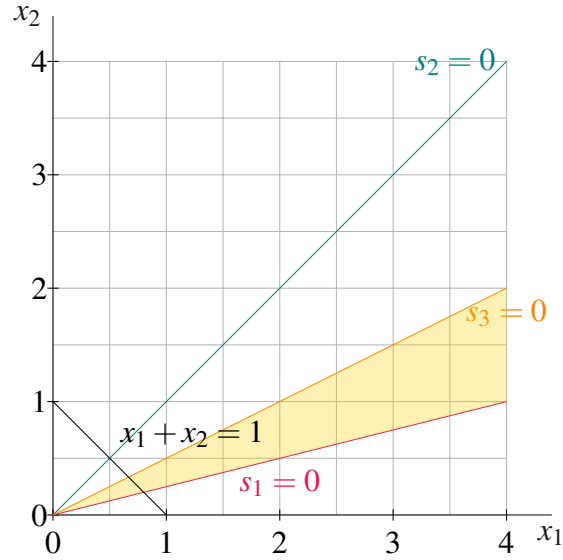&x_1, x_2, s_1, s_2, s_3 \geq 0.
\end{aligned}
$$

Represent point (1/2, 1) as a convex combination of the extreme points of the above LP. Find λs to solve the following system of equations:

$$
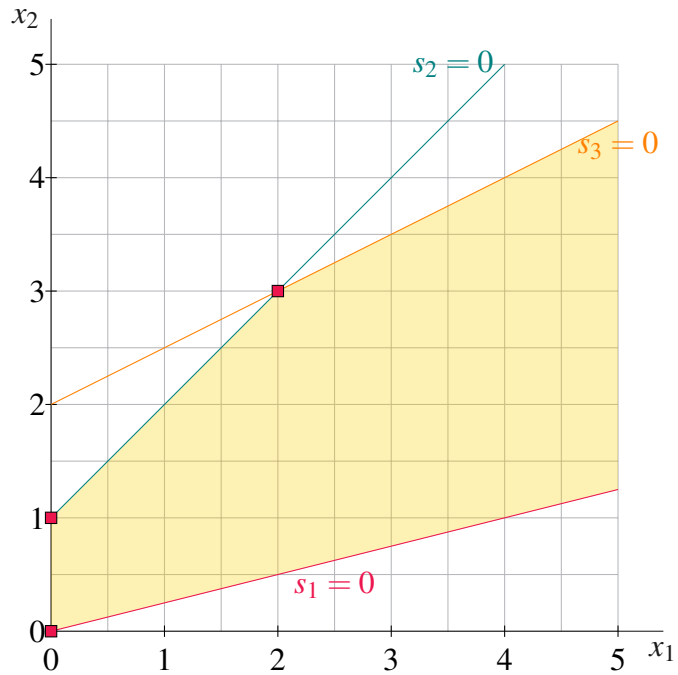\lambda_1 \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \lambda_2 \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \lambda_3 \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 1/2 \\ 1 \end{bmatrix}
$$

4.6 Graphical example

To motivate the subject of linear programming (LP), we begin with a planning problem that can be solved graphically.

Example 4.12: Lemonade Vendor

Say you are a vendor of lemonade and lemon juice. Each unit of lemonade requires 1 lemon and 2 litres of water. Each unit of lemon juice requires 3 lemons and 1 litre of water. Each unit of lemonade gives a profit of three dollars. Each unit of lemon juice gives a profit of two dollars. You have 6 lemons and 4 litres of water available. How many units of lemonade and lemon juice should you make to maximize profit?

If we let x denote the number of units of lemonade to be made and let y denote the number of units of lemon juice to be made, then the profit is given by $3x + 2y$ dollars. We call $3x + 2y$ the objective function. Note that there are a number of constraints that x and y must satisfied. First of all, x and y should be nonnegative. The number of lemons needed to make x units of lemonade and y units of lemon juice is $x + 3y$ and cannot exceed 6. The number of litres of water needed to make

x units of lemonade and y units of lemon juice is $2x + y$ and cannot exceed 4. Hence, to determine the maximum profit, we need to maximize $3x + 2y$ subject to x and y satisfying the constraints $x + 3y \leq 6$, $2x + y \leq 4$, $x \geq 0$, and $y \geq 0$.

A more compact way to write the problem is as follows:

$$
\begin{array}{rrcrcl}
\textit{maximize} & 3x & + & 2y & & \\
\textit{subject to} & x & + & 3y & \leq & 6 \\
& 2x & + & y & \leq & 4 \\
& x & & & \geq & 0 \\
& & & y & \geq & 0.
\end{array}
$$

We can solve this maximization problem graphically as follows. We first sketch the set of $\begin{bmatrix} x \\ y \end{bmatrix}$ satisfying the constraints, called the feasible region, on the (x, y)-plane. We then take the objective function $3x + 2y$ and turn it into an equation of a line $3x + 2y = z$ where z is a parameter. Note that as the value of z increases, the line defined by the equation $3x + 2y = z$ moves in the direction of the normal vector $\begin{bmatrix} 3 \\ 2 \end{bmatrix}$. We call this direction the direction of improvement. Determining the maximum value of the objective function, called the optimal value, subject to the contraints amounts to finding the maximum value of z so that the line defined by the equation $3x + 2y = z$ still intersects the feasible region.
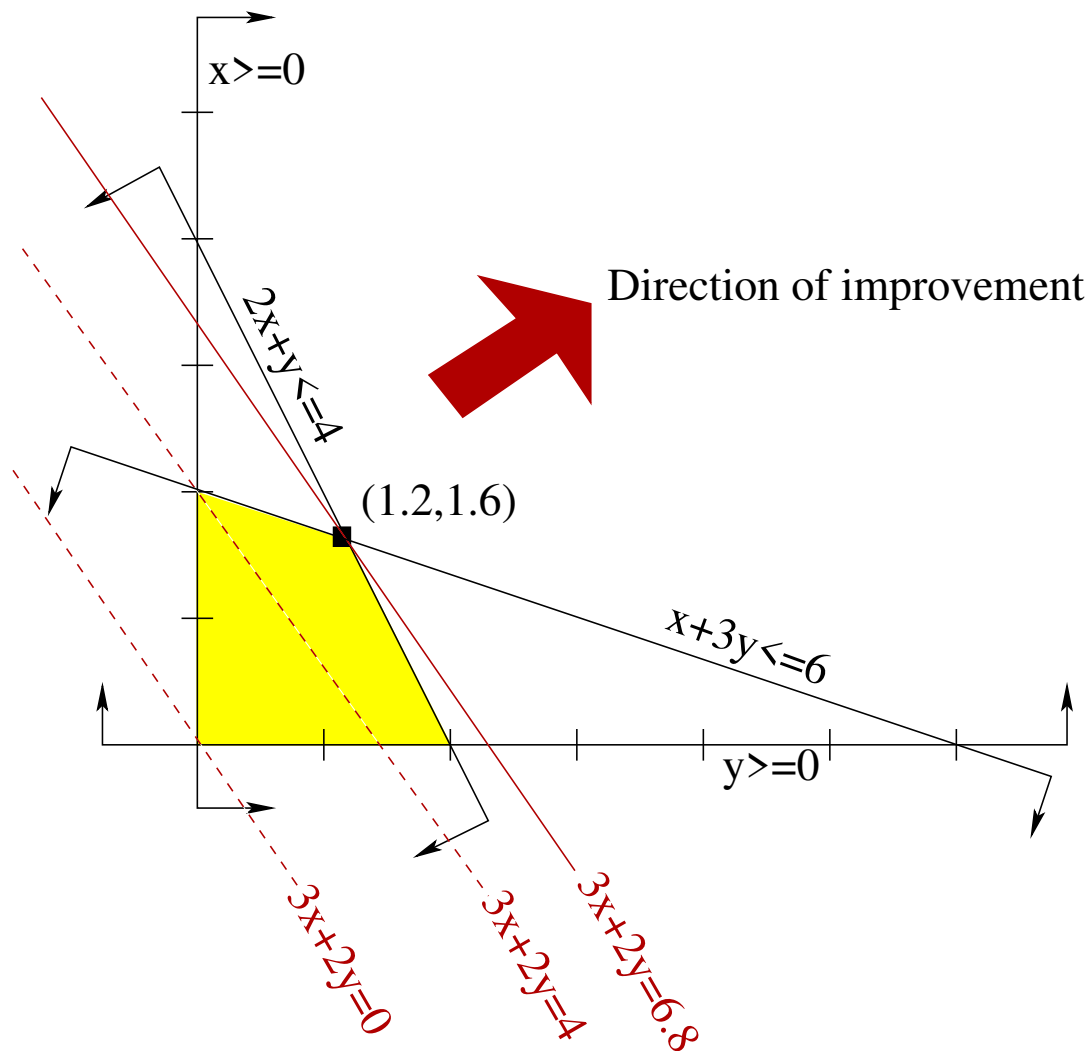
In the figure above, the lines with z at 0, 4 and 6.8 have been drawn. From the picture, we can see that if z is greater than 6.8, the line defined by $3x + 2y = z$ will not intersect the feasible region. Hence, the profit cannot exceed 6.8 dollars.

As the line $3x + 2y = 6.8$ does intersect the feasible region, 6.8 is the maximum value for the objective function. Note that there is only one point in the feasible region that intersects the line $3x + 2y = 6.8$, namely $\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1.2 \\ 1.6 \end{bmatrix}$. In other words, to maximize profit, we want to make 1.2 units of lemonade and 1.6 units of lemon juice.

The above solution method can hardly be regarded as rigorous because we relied on a picture to conclude that $3x + 2y \leq 6.8$ for all $\begin{bmatrix} x \\ y \end{bmatrix}$ satisfying the constraints. But we can actually show this *algebraically*.

Note that multiplying both sides of the constraint $x + 3y \leq 6$ gives $0.2x + 0.6y \leq 1.2$, and multiplying both sides of the constraint $2x + y \leq 4$ gives $2.8x + 1.4y \leq 5.6$. Hence, any $\begin{bmatrix} x \\ y \end{bmatrix}$ that satisfies both $x + 3y \leq 6$ and $2x + y \leq 4$ must also satisfy $(0.2x + 0.6y) + (2.8x + 1.4y) \leq 1.2 + 5.6$, which simplifies to $3x + 2y \leq 6.8$ as desired! (Here, we used the fact that if $a \leq b$ and $c \leq d$, then $a + c \leq b + d$.)

Now, one might ask if it is always possible to find an algebraic proof like the one above for similar problems. If the answer is yes, how does one find such a proof? We will see answers to this question later on.

Before we end this segment, let us consider the following problem:

$$
\begin{array}{rrcl}
\text{minimize} & -2x & + & y \\
\text{subject to} & -x & + & y \le 3 \\
& x & - & 2y \le 2 \\
& x & & \ge 0 \\
& & y & \ge 0.
\end{array}
$$

Note that for any $t \ge 0$, $\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} t \\ t \end{bmatrix}$ satisfies all the constraints. The value of the objective function at $\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} t \\ t \end{bmatrix}$ is $-t$. As $t \to \infty$, the value of the objective function tends to $-\infty$. Therefore, there is no minimum value for the objective function. The problem is said to be unbounded. Later on, we will see how to detect unboundedness algorithmically.

As an exercise, check that unboundedness can also be established by using $\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2t+2 \\ t \end{bmatrix}$ for $t \ge 0$.

Exercises

1. Sketch all $\begin{bmatrix} x \\ y \end{bmatrix}$ satisfying

$$x - 2y \le 2$$

   on the (x, y)-plane.

2. Determine the optimal value of

$$
\begin{array}{rl}
\text{Minimize} & x + y \\
\text{Subject to} & 2x + y \ge 4 \\
& x + 3y \ge 1.
\end{array}
$$

3. Show that the problem

$$
\begin{array}{rl}
\text{Minimize} & -x + y \\
\text{Subject to} & 2x - y \ge 0 \\
& x + 3y \ge 3
\end{array}
$$

   is unbounded.

4. Suppose that you are shopping for dietary supplements to satisfy your required daily intake of 0.40mg of nutrient M and 0.30mg of nutrient N. There are three popular products on the market. The costs and the amounts of the two nutrients are given in the following table:

	Product 1	Product 2	Product 3
Cost	$27	$31	$24
Daily amount of M	0.16 mg	0.21 mg	0.11 mg
Daily amount of N	0.19 mg	0.13 mg	0.15 mg

You want to determine how much of each product you should buy so that the daily intake require-ments of the two nutrients are satisfied at minimum cost. Formulate your problem as a linear pro-gramming problem, assuming that you can buy a fractional number of each product.

Solisions

1. The points (x,y) satisfying $x - 2y \leq 2$ are precisely those above the line passing through $(2,0)$ and $(0,-1)$.

2. We want to determine the minimum value z so that $x + y = z$ defines a line that has a nonempty intersection with the feasible region. However, we can avoid referring to a sketch by setting $x = z - y$ and substituting for x in the inequalities to obtain:

$$2(z - y) + y \geq 4$$
$$(z - y) + 3y \geq 1,$$

or equivalently,

$$z \geq 2 + \frac{1}{2}y$$
$$z \geq 1 - 2y,$$

Thus, the minimum value for z is $\min\{2 + \frac{1}{2}y, 1 - 2y\}$, which occurs at $y = -\frac{2}{5}$. Hence, the optimal value is $\frac{9}{5}$.

We can verify our work by doing the following. If our calculations above are correct, then an optimal solution is given by $x = \frac{11}{5}$, $y = -\frac{2}{5}$ since $x = z - y$. It is easy to check that this satisfies both inequalities and therefore is a feasible solution.

Now, taking $\frac{2}{5}$ times the first inequality and $\frac{1}{5}$ times the second inequality, we can infer the inequality $x + y \geq \frac{9}{5}$. The left-hand side of this inequality is precisely the objective function. Hence, no feasible solution can have objective function value less than $\frac{9}{5}$. But $x = \frac{11}{5}$, $y = -\frac{2}{5}$ is a feasible solution with objective function value equal to $\frac{9}{5}$. As a result, it must be an optimal solution.

Remark. We have not yet discussed how to obtain the multipliers $\frac{2}{5}$ and $\frac{1}{5}$ for inferring the inequality $x + y \geq \frac{9}{5}$. This is an issue that will be taken up later. In the meantime, think about how one could have obtained these multipliers for this particular exercise.

3. We could glean some insight by first making a sketch on the (x,y)-plane.

   The line defined by $-x+y=z$ has x-intercept $-z$. Note that for $z \le -3$, $\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -z \\ 0 \end{bmatrix}$ satisfies both inequalities and the value of the objective function at $\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -z \\ 0 \end{bmatrix}$ is z. Hence, there is no lower bound on the value of objective function.

4. Let x_i denote the amount of Product i to buy for $i = 1, 2, 3$. Then, the problem can be formulated as

$$
\begin{array}{llrcrcrcl}
\text{minimize} & 27x_1 & + & 31x_2 & + & 24x_3 \\
\text{subject to} & 0.16x_1 & + & 0.21x_2 & + & 0.11x_3 & \ge & 0.30 \\
& 0.19x_1 & + & 0.13x_2 & + & 0.15x_3 & \ge & 0.40 \\
& x_1 & , & x_2 & , & x_3 & \ge & 0.
\end{array}
$$

Remark. If one cannot buy fractional amounts of the products, the problem can be formulated as

$$
\begin{array}{llrcrcrcl}
\text{minimize} & 27x_1 & + & 31x_2 & + & 24x_3 \\
\text{subject to} & 0.16x_1 & + & 0.21x_2 & + & 0.11x_3 & \ge & 0.30 \\
& 0.19x_1 & + & 0.13x_2 & + & 0.15x_3 & \ge & 0.40 \\
& x_1 & , & x_2 & , & x_3 & \ge & 0. \\
& x_1 & , & x_2 & , & x_3 & \in & \mathbb{Z}.
\end{array}
$$
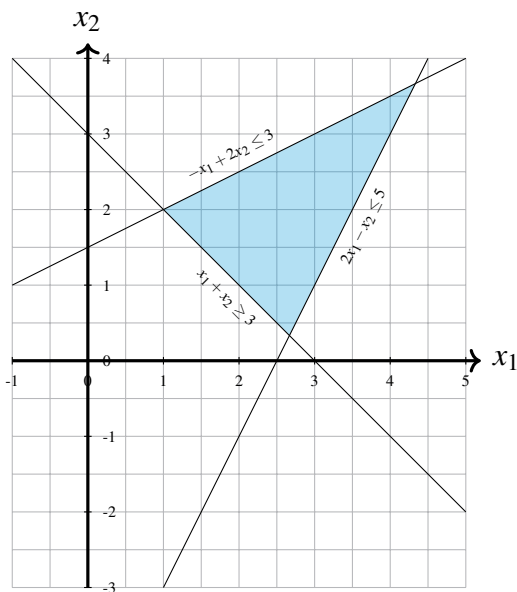
---

[2]https://tex.stackexchange.com/questions/75933/how-to-draw-the-region-of-inequality

5. Software - Excel

5.0.1. Excel Solver

5.0.2. Videos

Solving a linear program Optimal product mix Set Cover

Introduction to Designing Optimization Models Using Excel Solver

Traveling Salesman Problem

Also Travelin Salesman Problem

Multiple Traveling Salesman Problem

Shortest Path

5.0.3. Links

Loan Example

Several Examples including TSP

6. Software - Python

Outcomes

  - *Install and get python up and running in some form*
  - *Introduce basic python skills that will be helpful*

Resources

  - *A Byte of Python*
  - *Github - Byte of Python (CC-BY-SA)*

6.1 Installing and Managing Python

Installing and Managing Python

Lab Objective: *One of the great advantages of Python is its lack of overhead: it is relatively easy to download, install, start up, and execute. This appendix introduces tools for installing and updating specific packages and gives an overview of possible environments for working efficiently in Python.*

Installing Python via Anaconda

A *Python distribution* is a single download containing everything needed to install and run Python, together with some common packages. For this curriculum, we **strongly** recommend using the *Anaconda* distribution to install Python. Anaconda includes IPython, a few other tools for developing in Python, and a large selection of packages that are common in applied mathematics, numerical computing, and data science. Anaconda is free and available for Windows, Mac, and Linux.

Follow these steps to install Anaconda.

1. Go to `https://www.anaconda.com/download/`.

2. Download the **Python 3.6** graphical installer specific to your machine.

3. Open the downloaded file and proceed with the default configurations.

For help with installation, see `https://docs.anaconda.com/anaconda/install/`. This page contains links to detailed step-by-step installation instructions for each operating system, as well as information for updating and uninstalling Anaconda.

> **ACHTUNG!**
>
> This curriculum uses Python 3.6, **not** Python 2.7. With the wrong version of Python, some example code within the labs may not execute as intended or result in an error.

Managing Packages

A *Python package manager* is a tool for installing or updating Python packages, which involves downloading the right source code files, placing those files in the correct location on the machine, and linking the files to the Python interpreter. **Never** try to install a Python package without using a package manager (see `https://xkcd.com/349/`).

Conda

Many packages are not included in the default Anaconda download but can be installed via Anaconda's package manager, conda. See `https://docs.anaconda.com/anaconda/packages/pkg-docs` for the complete list of available packages. When you need to update or install a package, **always** try using conda first.

Command	Description
`conda install <package-name>`	Install the specified package.
`conda update <package-name>`	Update the specified package.
`conda update conda`	Update conda itself.
`conda update anaconda`	Update **all** packages included in Anaconda.
`conda --help`	Display the documentation for conda.

For example, the following terminal commands attempt to install and update matplotlib.

```
$ conda update conda            # Make sure that conda is up to date.
$ conda install matplotlib      # Attempt to install matplotlib.
$ conda update matplotlib       # Attempt to update matplotlib.
```

See `https://conda.io/docs/user-guide/tasks/manage-pkgs.html` for more examples.

> **NOTE**
>
> The best way to ensure a package has been installed correctly is to try importing it in IPython.
>
> ```
> # Start IPython from the command line.
> $ ipython
> IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.
>
> # Try to import matplotlib.
> In [1]: from matplotlib import pyplot as plt      # Success!
> ```

> **ACHTUNG!**
>
> Be careful not to attempt to update a Python package while it is in use. It is safest to update packages while the Python interpreter is not running.

Pip

The most generic Python package manager is called `pip`. While it has a larger package list, `conda` is the cleaner and safer option. Only use `pip` to manage packages that are not available through `conda`.

Command	Description
`pip install package-name`	Install the specified package.
`pip install --upgrade package-name`	Update the specified package.
`pip freeze`	Display the version number on all installed packages.
`pip --help`	Display the documentation for `pip`.

See `https://pip.pypa.io/en/stable/user_guide/` for more complete documentation.

Workflows

There are several different ways to write and execute programs in Python. Try a variety of workflows to find what works best for you.

Text Editor + Terminal

The most basic way of developing in Python is to write code in a text editor, then run it using either the Python or IPython interpreter in the terminal.

There are many different text editors available for code development. Many text editors are designed specifically for computer programming which contain features such as syntax highlighting and error detection, and are highly customizable. Try installing and using some of the popular text editors listed below.

- Atom: https://atom.io/

- Sublime Text: https://www.sublimetext.com/

- Notepad++ (Windows): https://notepad-plus-plus.org/

- Geany: https://www.geany.org/

- Vim: https://www.vim.org/

- Emacs: https://www.gnu.org/software/emacs/

Once Python code has been written in a text editor and saved to a file, that file can be executed in the terminal or command line.

```
$ ls                        # List the files in the current directory.
hello_world.py
$ cat hello_world.py        # Print the contents of the file to the terminal.
print("hello, world!")
$ python hello_world.py     # Execute the file.
hello, world!

# Alternatively, start IPython and run the file.
$ ipython
IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: %run hello_world.py
hello, world!
```

IPython is an enhanced version of Python that is more user-friendly and interactive. It has many features that cater to productivity such as tab completion and object introspection.

> **NOTE**
>
> While Mac and Linux computers come with a built-in bash terminal, Windows computers do not. Windows does come with *Powershell*, a terminal-like application, but some commands in Powershell are different than their bash analogs, and some bash commands are missing from Powershell

altogether. There are two good alternatives to the bash terminal for Windows:

- Windows subsystem for linux: `docs.microsoft.com/en-us/windows/wsl/`.

- Git bash: `https://gitforwindows.org/`.

Jupyter Notebook

The Jupyter Notebook (previously known as IPython Notebook) is a browser-based interface for Python that comes included as part of the Anaconda Python Distribution. It has an interface similar to the IPython interpreter, except that input is stored in cells and can be modified and re-evaluated as desired. See `https://github.com/jupyter/jupyter/wiki/` for some examples.

To begin using Jupyter Notebook, run the command `jupyter notebook` in the terminal. This will open your file system in a web browser in the Jupyter framework. To create a Jupyter Notebook, click the **New** drop down menu and choose **Python 3** under the **Notebooks** heading. A new tab will open with a new Jupyter Notebook.

Jupyter Notebooks differ from other forms of Python development in that notebook files contain not only the raw Python code, but also formatting information. As such, Juptyer Notebook files cannot be run in any other development environment. They also have the file extension `.ipynb` rather than the standard Python extension `.py`.

Jupyter Notebooks also support Markdown—a simple text formatting language—and LaTeX, and can embedded images, sound clips, videos, and more. This makes Jupyter Notebook the ideal platform for presenting code.

Integrated Development Environments

An *integrated development environment* (IDEs) is a program that provides a comprehensive environment with the tools necessary for development, all combined into a single application. Most IDEs have many tightly integrated tools that are easily accessible, but come with more overhead than a plain text editor. Consider trying out each of the following IDEs.

- JupyterLab: `http://jupyterlab.readthedocs.io/en/stable/`

- PyCharm: `https://www.jetbrains.com/pycharm/`

- Spyder: `http://code.google.com/p/spyderlib/`

- Eclipse with PyDev: `http://www.eclipse.org/`, `https://www.pydev.org/`

See `https://realpython.com/python-ides-code-editors-guide/` for a good overview of these (and other) workflow tools.

6.2 NumPy Visual Guide

NumPy Visual Guide **Lab Objective:** *NumPy operations can be difficult to visualize, but the concepts are straightforward. This appendix provides visual demonstrations of how NumPy arrays are used with slicing syntax, stacking, broadcasting, and axis-specific operations. Though these visualizations are for 1- or 2-dimensional arrays, the concepts can be extended to n-dimensional arrays.*

Data Access

The entries of a 2-D array are the rows of the matrix (as 1-D arrays). To access a single entry, enter the row index, a comma, and the column index. Remember that indexing begins with 0.

$$
\texttt{A[0]} =
\begin{bmatrix}
\times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times
\end{bmatrix}
\qquad
\texttt{A[2,1]} =
\begin{bmatrix}
\times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times
\end{bmatrix}
$$

Slicing

A lone colon extracts an entire row or column from a 2-D array. The syntax `[a:b]` can be read as "the *a*th entry up to (but not including) the *b*th entry." Similarly, `[a:]` means "the *a*th entry to the end" and `[:b]` means "everything up to (but not including) the *b*th entry."

$$
\texttt{A[1]} = \texttt{A[1,:]} =
\begin{bmatrix}
\times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times
\end{bmatrix}
\qquad
\texttt{A[:,2]} =
\begin{bmatrix}
\times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times
\end{bmatrix}
$$

$$
\texttt{A[1:,:2]} =
\begin{bmatrix}
\times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times
\end{bmatrix}
\qquad
\texttt{A[1:-1,1:-1]} =
\begin{bmatrix}
\times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times
\end{bmatrix}
$$

Stacking

`np.hstack()` stacks sequence of arrays horizontally and `np.vstack()` stacks a sequence of arrays vertically.

$$A = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix} \qquad\qquad B = \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix}$$

$$\texttt{np.hstack((A,B,A))} = \begin{bmatrix} \times & \times & \times & * & * & * & \times & \times & \times \\ \times & \times & \times & * & * & * & \times & \times & \times \\ \times & \times & \times & * & * & * & \times & \times & \times \end{bmatrix}$$

$$\texttt{np.vstack((A,B,A))} = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ * & * & * \\ * & * & * \\ * & * & * \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix}$$

Because 1-D arrays are flat, `np.hstack()` concatenates 1-D arrays and `np.vstack()` stacks them vertically. To make several 1-D arrays into the columns of a 2-D array, use `np.column_stack()`.

$$x = \begin{bmatrix} \times & \times & \times & \times \end{bmatrix} \qquad\qquad y = \begin{bmatrix} * & * & * & * \end{bmatrix}$$

$$\texttt{np.hstack((x,y,x))} = \begin{bmatrix} \times & \times & \times & \times & * & * & * & * & \times & \times & \times & \times \end{bmatrix}$$

$$\texttt{np.vstack((x,y,x))} = \begin{bmatrix} \times & \times & \times & \times \\ * & * & * & * \\ \times & \times & \times & \times \end{bmatrix} \qquad \texttt{np.column\_stack((x,y,x))} = \begin{bmatrix} \times & * & \times \\ \times & * & \times \\ \times & * & \times \\ \times & * & \times \end{bmatrix}$$

The functions `np.concatenate()` and `np.stack()` are more general versions of `np.hstack()` and `np.vstack()`, and `np.row_stack()` is an alias for `np.vstack()`.

Broadcasting

NumPy automatically aligns arrays for component-wise operations whenever possible. See `http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html` for more in-depth examples and broadcasting rules.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} \qquad\qquad x = \begin{bmatrix} 10 & 20 & 30 \end{bmatrix}$$

$$\texttt{A + x} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} + \begin{bmatrix} 10 & 20 & 30 \end{bmatrix} \qquad = \begin{bmatrix} 11 & 22 & 33 \\ 11 & 22 & 33 \\ 11 & 22 & 33 \end{bmatrix}$$

$$\texttt{A + x.reshape((1,-1))} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} + \begin{bmatrix} 10 \\ 20 \\ 30 \end{bmatrix} \qquad = \begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{bmatrix}$$

Operations along an Axis

Most array methods have an `axis` argument that allows an operation to be done along a given axis. To compute the sum of each column, use `axis=0`; to compute the sum of each row, use `axis=1`.

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

$$\texttt{A.sum(axis=0)} = \begin{bmatrix} \boxed{1} & \boxed{2} & \boxed{3} & \boxed{4} \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{bmatrix} = \begin{bmatrix} 4 & 8 & 12 & 16 \end{bmatrix}$$

$$\texttt{A.sum(axis=1)} = \begin{bmatrix} \boxed{1 \quad 2 \quad 3 \quad 4} \\ \boxed{1 \quad 2 \quad 3 \quad 4} \\ \boxed{1 \quad 2 \quad 3 \quad 4} \\ \boxed{1 \quad 2 \quad 3 \quad 4} \end{bmatrix} = \begin{bmatrix} 10 & 10 & 10 & 10 \end{bmatrix}$$

6.3 Plot Customization and Matplotlib Syntax Guide

Matplotlib Customization

Lab Objective: *The documentation for Matplotlib can be a little difficult to maneuver and basic information is sometimes difficult to find. This appendix condenses and demonstrates some of the more applicable and useful information on plot customizations. For an introduction to Matplotlib, see lab* **??**.

Colors

By default, every plot is assigned a different color specified by the "color cycle". It can be overwritten by specifying what color is desired in a few different ways.

•

Matplotlib recognizes some basic built-in colors.

Code	Color
`'b'`	blue
`'g'`	green
`'r'`	red
`'c'`	cyan
`'m'`	magenta
`'y'`	yellow
`'k'`	black
`'w'`	white

The following displays how these colors can be implemented. The result is displayed in Figure 6.1.

```python
import numpy as np
from matplotlib import pyplot as plt

colors = np.array(["k", "g", "b", "r", "c", "m", "y", "w"])
x = np.linspace(0, 5, 1000)
y = np.ones(1000)

for i in xrange(8):
    plt.plot(x, i*y, colors[i], linewidth=18)

plt.ylim([-1, 8])
plt.savefig("colors.pdf", format='pdf')
plt.clf()
```
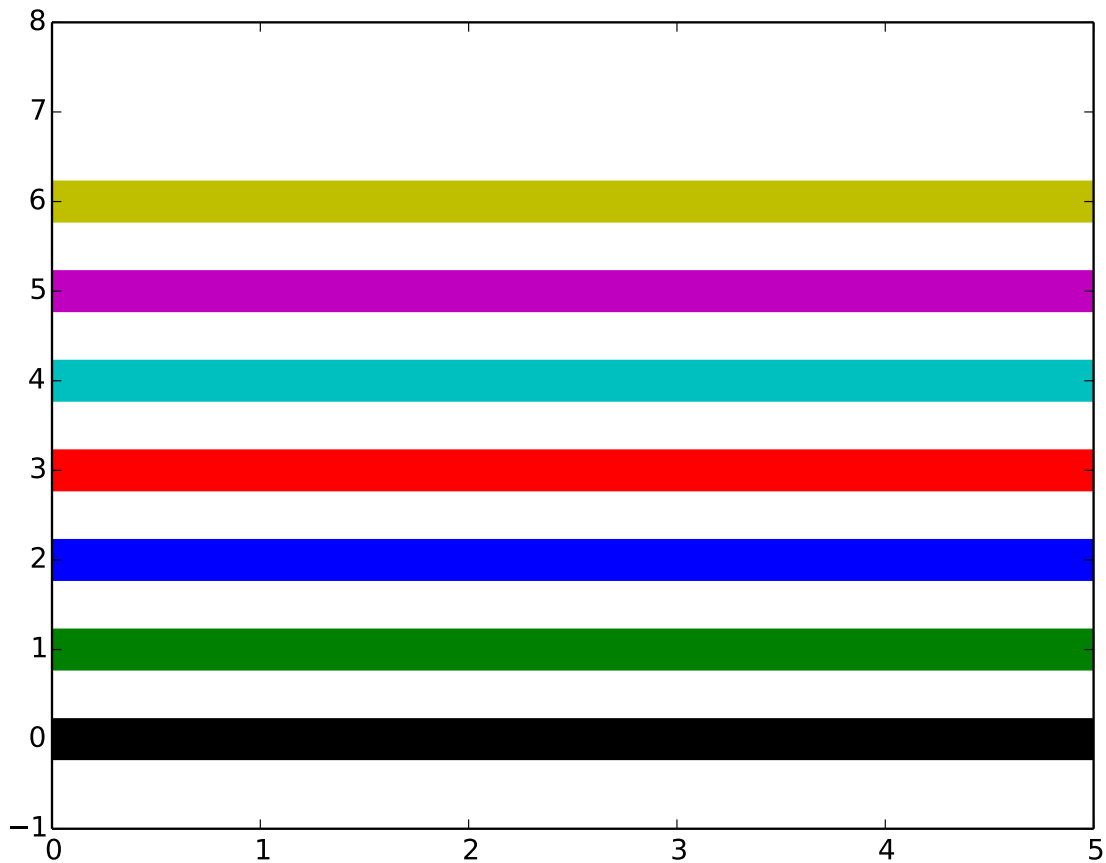
colors.py

Figure 6.1: A display of all the built-in colors.

There are many other ways to specific colors. A popular method to access colors that are not built-in is to use a RGB tuple. Colors can also be specified using an html hex string or its associated html color name like "DarkOliveGreen", "FireBrick", "LemonChiffon", "MidnightBlue", "PapayaWhip", or "SeaGreen".

Window Limits

You may have noticed the use of plt.ylim([ymin, ymax]) in the previous code. This explicitly sets the boundary of the y-axis. Similarily, plt.xlim([xmin, xmax]) can be used to set the boundary of the x-axis. Doing both commands simultaneously is possible with the plt.axis([xmin, xmax, ymin, ymax]). Remember that these commands must be executed after the plot.

Lines

Thickness

You may have noticed that the width of the lines above seemed thin considering we wanted to inspect the line color. linewidth is a keyword argument that is defaulted to be None but can be given any real number to adjust the line width.

The following displays how linewidth is implemented. It is displayed in Figure 6.2.

```
lw = np.linspace(.5, 15, 8)

for i in xrange(8):
    plt.plot(x, i*y, colors[i], linewidth=lw[i])

plt.ylim([-1, 8])
plt.show()
```
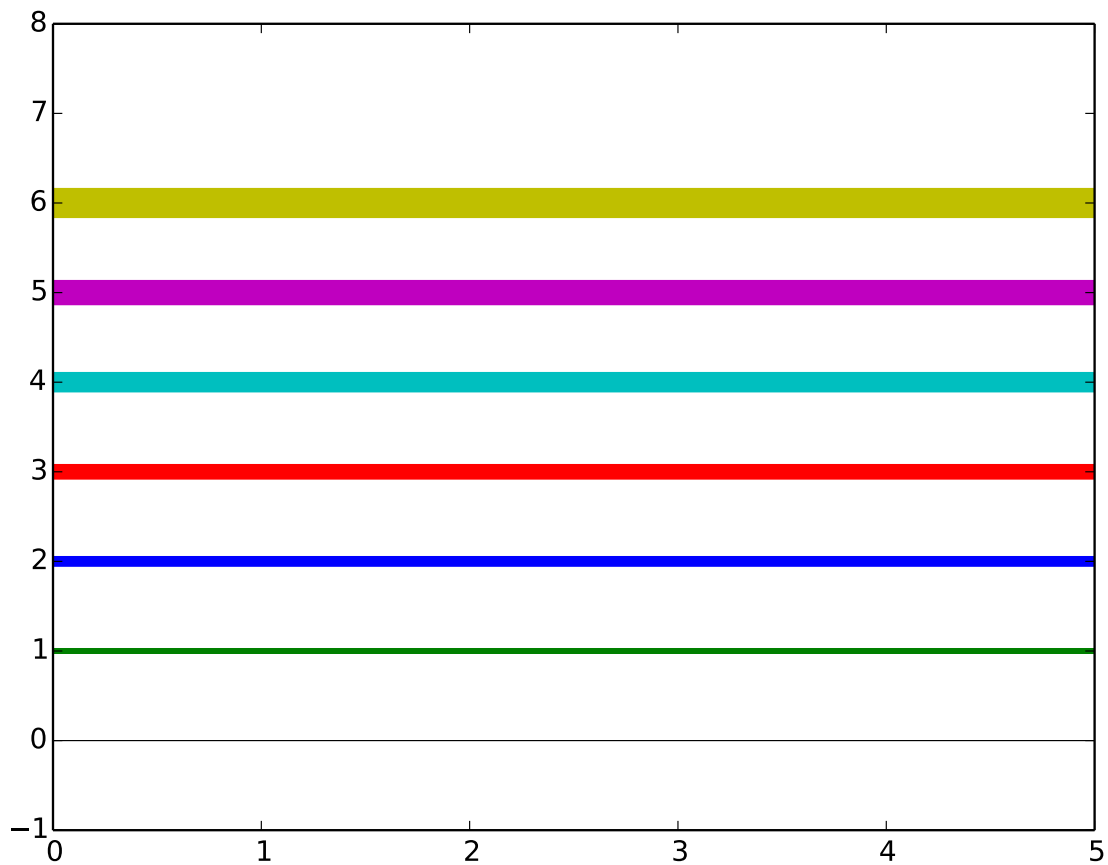
linewidth.py

Figure 6.2: plot of varying linewidths.

Style

By default, plots are drawn with a solid line. The following are accepted format string characters to indicate line style.

character	description
-	solid line style
--	dashed line style
-.	dash-dot line style
:	dotted line style
.	point marker
,	pixel marker
o	circle marker
v	triangle_down marker
^	triangle_up marker
<	triangle_left marker
>	triangle_right marker
1	tri_down marker
2	tri_up marker
3	tri_left marker
4	tri_right marker
s	square marker
p	pentagon marker
*	star marker
h	hexagon1 marker
H	hexagon2 marker
+	plus marker
x	x marker
D	diamond marker
d	thin_diamond marker
|	vline marker
_	hline marker

The following displays how `linestyle` can be implemented. It is displayed in Figure 6.3.

```
x = np.linspace(0, 5, 10)
y = np.ones(10)
ls = np.array(['-.', ':', 'o', 's', '*', 'H', 'x', 'D'])

for i in xrange(8):
    plt.plot(x, i*y, colors[i]+ls[i])

plt.axis([-1, 6, -1, 8])
plt.show()
```
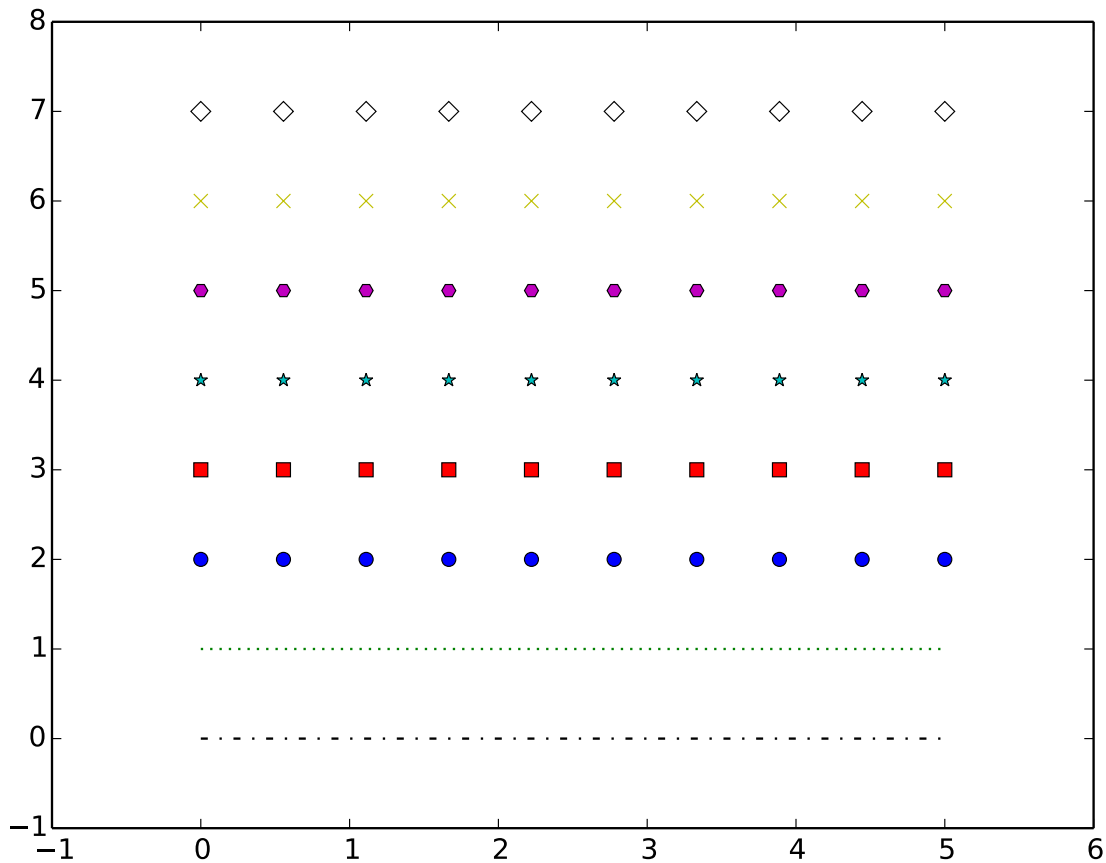
linestyle.py

Figure 6.3: plot of varying linestyles.

Text

It is also possible to add text to your plots. To label your axes, the `plt.xlabel()` and the `plt.ylabel()` can both be used. The function `plt.title()` will add a title to a plot. If you are working with subplots, this command will add a title to the subplot you are currently modifying. To add a title above the entire figure, use `plt.suptitle()`.

All of the `text()` commands can be customized with `fontsize` and `color` keyword arguments.

We can add these elements to our previous example. It is displayed in Figure 6.4.

```
for i in xrange(8):
    plt.plot(x, i*y, colors[i]+ls[i])


plt.title("My Plot of Varying Linestyles", fontsize = 20, color = "gold")
```

```
  plt.xlabel("x-axis", fontsize = 10, color = "darkcyan")
6 plt.ylabel("y-axis", fontsize = 10, color = "darkcyan")

8 plt.axis([-1, 6, -1, 8])
  plt.show()
```
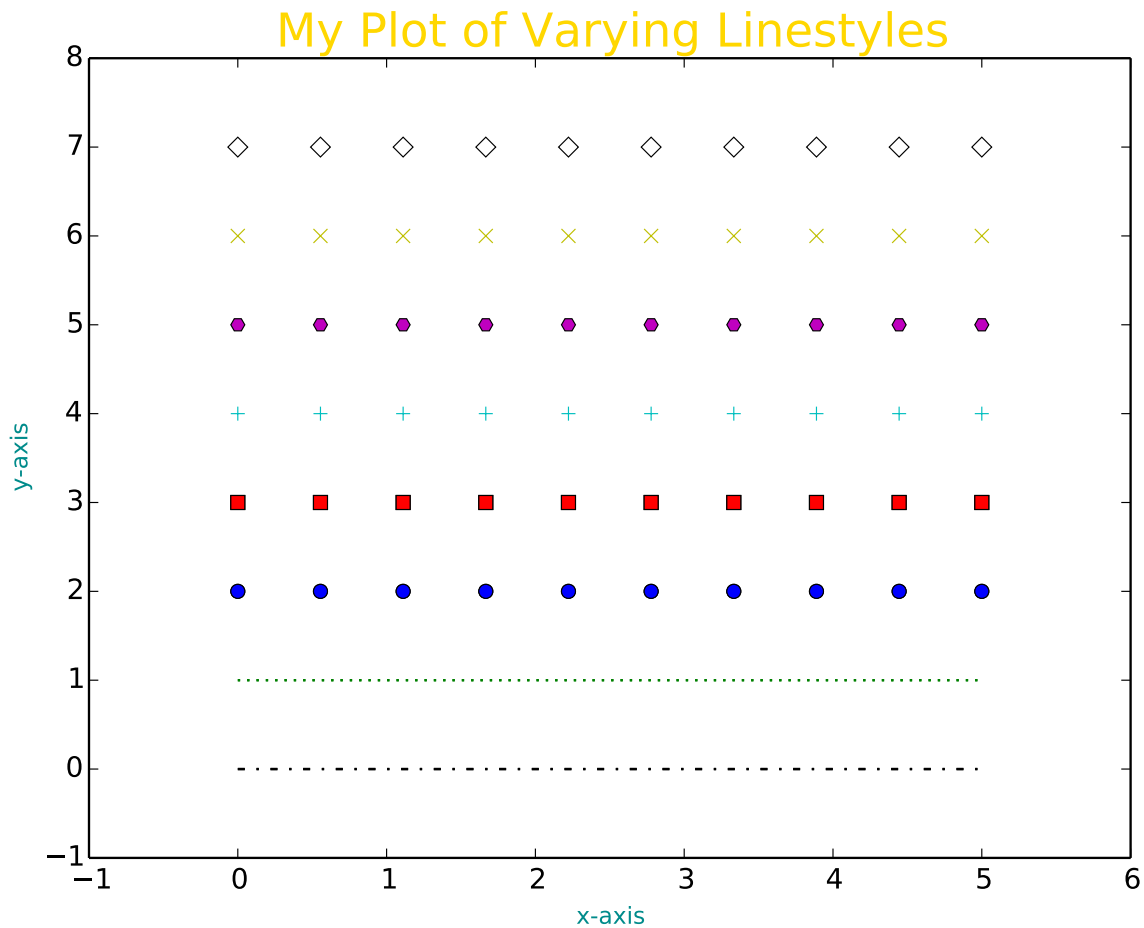
text.py

Figure 6.4: plot of varying linestyles using text labels.

See http://matplotlib.org for Matplotlib documentation.

6.4 Networkx - A Python Graph Algorithms Package

6.5 PuLP - An Optimization Modeling Tool for Python

> **Outcomes**
>
> - *Install and import PuLP*
> - *Run basic first PuLP model*
> - *Run "advanced" PuLP model using the algebraic modeling approach and importing data.*
> - *Explore PuLP objects and possibilities*
> - *Solve a Multi-Objective problem*

> **Resources**
>
> - *Documentation*
> - *PyPi installation*
> - *Examples*
> - *Blog with tutorial*

PuLP is an optimizaiton modeling language that is written for Python. It is free and open source. Yay! See Section ?? for a discussion of other options for implementing your optimization problem. PuLP is convenient for it's simple syntax and easy installation.

Key benefits of using an algebraic modeling language like PuLP over Excel

- Easily readable models

- Precompute parameters within Python

- Reuse of common optimization models without recreating the equations

We will follow the introduction to pulp Jupyter Notebook Tutorial and the following application with a cleaner implementation.

6.5.1. Installation

Open a Jupyter notebook. In one of the cells, run the following command, based on which system you are running. It will take a minute to load and download the package.

```
[ ]: ## Install pulp (on windows)
     !pip install pulp
```

```
[ ]: # on a mac
     pip install pulp
```

```
[ ]: # on the VT ARC servers
     import sys
     !{sys.executable} -m pip install pulp
```

Installation (Continued) Now restart the kernel of your notebook (find the tab labeled Kernel in your Jupyter notebook, and in the drop down, select restart).

6.5.2. Example Problem

6.5.2.1. Product Mix Problem

$$
\begin{array}{llll}
\text{maximize} & Z = 3X_1 + 2X_2 & \text{(Objective function)} & (1.1) \\
\text{subject to} & 10X_1 + 5X_2 \leq 300 & \text{(Constraint 1)} & (1.2) \\
& 4X_1 + 4X_2 \leq 160 & \text{(Constraint 2)} & (1.3) \\
& 2X_1 + 6X_2 \leq 180 & \text{(Constraint 3)} & (1.4) \\
\text{and} & X_1, X_2 \geq 0 & \text{(Non-negative)} & (1.5)
\end{array}
$$

OPTIMIZATION WITH PULP

```
[1]: from pulp import *

     # Define problem
     prob = LpProblem(name='Product_Mix_Problem', sense=LpMaximize)

     # Create decision variables and non-negative constraint
     x1 = LpVariable(name='X1', lowBound=0, upBound=None, cat='Continuous')
     x2 = LpVariable(name='X2', lowBound=0, upBound=None, cat='Continuous')
```

```python
# Set objective function
prob += 3*x1 + 2*x2

# Set constraints
prob += 10*x1 + 5*x2 <= 300
prob += 4*x1 + 4*x2 <= 160
prob += 2*x1 + 6*x2 <= 180

# Solving problem
prob.solve()
print('Status', LpStatus[prob.status])
```

```
Status Optimal
```

```python
[2]: print("Status:", LpStatus[prob.status])
     print("Objective value: ", prob.objective.value())

     for v in prob.variables():
         print(v.name,': ', v.value())
```

```
Status: Optimal
Objective value:  100.0
X1 :  20.0
X2 :  20.0
```

6.5.3. Things we can do

```python
[3]: # print the problem
     prob
```

```
[3]: Product_Mix_Problem:
     MAXIMIZE
     3*X1 + 2*X2 + 0
     SUBJECT TO
     _C1: 10 X1 + 5 X2 <= 300

     _C2: 4 X1 + 4 X2 <= 160

     _C3: 2 X1 + 6 X2 <= 180

     VARIABLES
```

```
X1 Continuous
X2 Continuous
```

```
[4]: # get the objective function
     prob.objective.value()
```

```
[4]: 100.0
```

```
[5]: # get list of the variables
     prob.variables()
```

```
[5]: [X1, X2]
```

```
[6]: for v in prob.variables():
         print(f'{v}: {v.varValue}')
```

```
X1: 20.0
X2: 20.0
```

6.5.3.1. Exploring the variables

```
[7]: v = prob.variables()[0]
```

```
[9]: v.name
```

```
[9]: 'X1'
```

```
[10]: v.value()
```

```
[10]: 20.0
```

```
[11]: v.varValue
```

```
[11]: 20.0
```

6.5.3.2. Other things you can do

```
[12]:  # get list of the constraints
       prob.constraints
```

```
[12]:  OrderedDict([('_C1', 10*X1 + 5*X2 + -300 <= 0),
                    ('_C2', 4*X1 + 4*X2 + -160 <= 0),
                    ('_C3', 2*X1 + 6*X2 + -180 <= 0)])
```

```
[13]:  prob.to_dict()
```

```
[13]:  {'objective': {'name': 'OBJ',
         'coefficients': [{'name': 'X1', 'value': 3}, {'name': 'X2', 'value': 2}]},
        'constraints': [{'sense': -1,
          'pi': 0.2,
          'constant': -300,
          'name': None,
          'coefficients': [{'name': 'X1', 'value': 10}, {'name': 'X2', 'value': 5}]},
         {'sense': -1,
          'pi': 0.25,
          'constant': -160,
          'name': None,
          'coefficients': [{'name': 'X1', 'value': 4}, {'name': 'X2', 'value': 4}]},
         {'sense': -1,
          'pi': -0.0,
          'constant': -180,
          'name': None,
          'coefficients': [{'name': 'X1', 'value': 2}, {'name': 'X2', 'value': 6}]}],
        'variables': [{'lowBound': 0,
          'upBound': None,
          'cat': 'Continuous',
          'varValue': 20.0,
          'dj': -0.0,
          'name': 'X1'},
         {'lowBound': 0,
          'upBound': None,
          'cat': 'Continuous',
          'varValue': 20.0,
          'dj': -0.0,
          'name': 'X2'}],
        'parameters': {'name': 'Product_Mix_Problem',
          'sense': -1,
```

```
 'status': 1,
 'sol_status': 1},
'sos1': [],
'sos2': []}
```

[15]:
```
# Store problem information in a json
prob.to_json('Product_Mix_Problem.json')
```

6.5.4. Common issue

If you forget the <=, ==, or >= when writing a constraint, you will silently overwrite the objective function instead of adding a constraint!

6.5.4.1. Transportation Problem

Transport programming is a special form of linear programming, and in general, the objective function is cost minimization. The formula form and applicable variables of the Transport Planning Act are as follows. When supply and demand match, the constraint becomes an equation, but when supply and demand do not match, the constraint becomes an inequality.

Sets: - J = set of demand nodes - I = set of supply nodes

Parameters:

- D_j: Demand at node j

- S_i: Supply from node i

- c_{ij}: cost per unit to send supply i to demand j

Variables:

- X_ij: Transport volume from supply i to demand j (units)

- Objective function:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij}$$

- Constraints:

$$\sum_{i=1}^{n} x_{ij} = S_i$$

$$\sum_{i=1}^{m} x_{ij} = D_j$$

$$x_{ij} \geq 0 \text{ for } i \in I, j \in J$$

6.5.4.2. Optimization with PuLP

Here we do a very basic implementation of the problem

```
[1]: from pulp import *

     prob = LpProblem('Transportation_Problem', LpMinimize)

     x11 = LpVariable('X11', lowBound=0)
     x12 = LpVariable('X12', lowBound=0)
     x13 = LpVariable('X13', lowBound=0)
     x14 = LpVariable('X14', lowBound=0)
     x21 = LpVariable('X21', lowBound=0)
     x22 = LpVariable('X22', lowBound=0)
     x23 = LpVariable('X23', lowBound=0)
     x24 = LpVariable('X24', lowBound=0)
     x31 = LpVariable('X31', lowBound=0)
     x32 = LpVariable('X32', lowBound=0)
     x33 = LpVariable('X33', lowBound=0)
     x34 = LpVariable('X34', lowBound=0)

     prob += 4*x11 + 5*x12 + 6*x13 + 8*x14 + 4*x21 + 7*x22 + 9*x23 + 2*x24 + 5*x31 +␣
      ↪8*x32 + 7*x33 + 6*x34

     prob += x11 + x12 + x13 + x14 == 120
     prob += x21 + x22 + x23 + x24 == 150
     prob += x31 + x32 + x33 + x34 == 200

     prob += x11 + x21 + x31 == 100
     prob += x12 + x22 + x32 == 60
     prob += x13 + x23 + x33 == 130
     prob += x14 + x24 + x34 == 180

     # Solving problem
     prob.solve();
```

```
[2]: print("Status:", LpStatus[prob.status])
     print("Objective value: ", prob.objective.value())

     for v in prob.variables():
         print(v.name,': ', v.value())
```

```
Status: Optimal
Objective value:   2130.0
X11 :   60.0
X12 :   60.0
X13 :   0.0
X14 :   0.0
X21 :   0.0
X22 :   0.0
X23 :   0.0
X24 :   150.0
X31 :   40.0
X32 :   0.0
X33 :   130.0
X34 :   30.0
```

6.5.4.3. Optimization with PuLP: Round 2!

We now use set notation for this implementation

```python
[3]: from pulp import *

     prob = LpProblem('Transportation_Problem', LpMinimize)


     # Sets
     n_suppliers = 3
     n_buyers = 4

     I = range(n_suppliers)
     J = range(n_buyers)

     routes = [(i, j) for i in I for j in J]


     # Parameters
     costs = [
         [4, 5, 6, 8],
         [4, 7, 9, 2],
         [5, 8, 7, 6]
     ]

     supply = [120, 150, 200]
     demand = [100, 60, 130, 180]
```

```python
# Variables
x = LpVariable.dicts('X', routes, lowBound=0)

# Objective
prob += lpSum([x[i, j] * costs[i][j] for i in I for j in J])


# Constraints

## Supply Constraints
for i in range(n_suppliers):
    prob += lpSum([x[i, j] for j in J]) == supply[i], f"Supply{i}"

## Demand Constraints
for j in range(n_buyers):
    prob += lpSum([x[i, j] for i in I]) == demand[j], f"Demand{j}"

# Solving problem
prob.solve();
```

```python
[4]: print("Status:", LpStatus[prob.status])
     print("Objective value: ", prob.objective.value())

     for v in prob.variables():
         print(v.name,': ', v.value())
```

```
Status: Optimal
Objective value:  2130.0
X_(0,_0) :  60.0
X_(0,_1) :  60.0
X_(0,_2) :  0.0
X_(0,_3) :  0.0
X_(1,_0) :  0.0
X_(1,_1) :  0.0
X_(1,_2) :  0.0
X_(1,_3) :  150.0
X_(2,_0) :  40.0
X_(2,_1) :  0.0
X_(2,_2) :  130.0
X_(2,_3) :  30.0
```

6.5.5. Changing details of the problem

```
[5]: original_obj = prob.objective
     val = prob.objective.value()
     r = 1.2
```

```
[6]: prob += original_obj <= r*val, "Objective bound"
```

```
[7]: prob
```

```
[7]: Transportation_Problem:
     MINIMIZE
     4*X_(0,_0) + 5*X_(0,_1) + 6*X_(0,_2) + 8*X_(0,_3) + 4*X_(1,_0) + 7*X_(1,_1) +
     9*X_(1,_2) + 2*X_(1,_3) + 5*X_(2,_0) + 8*X_(2,_1) + 7*X_(2,_2) + 6*X_(2,_3) + 0
     SUBJECT TO
     Supply0: X_(0,_0) + X_(0,_1) + X_(0,_2) + X_(0,_3) = 120

     Supply1: X_(1,_0) + X_(1,_1) + X_(1,_2) + X_(1,_3) = 150

     Supply2: X_(2,_0) + X_(2,_1) + X_(2,_2) + X_(2,_3) = 200

     Demand0: X_(0,_0) + X_(1,_0) + X_(2,_0) = 100

     Demand1: X_(0,_1) + X_(1,_1) + X_(2,_1) = 60

     Demand2: X_(0,_2) + X_(1,_2) + X_(2,_2) = 130

     Demand3: X_(0,_3) + X_(1,_3) + X_(2,_3) = 180

     Objective_bound: 4 X_(0,_0) + 5 X_(0,_1) + 6 X_(0,_2) + 8 X_(0,_3)
      + 4 X_(1,_0) + 7 X_(1,_1) + 9 X_(1,_2) + 2 X_(1,_3) + 5 X_(2,_0) + 8 X_(2,_1)
      + 7 X_(2,_2) + 6 X_(2,_3) <= 2556

     VARIABLES
     X_(0,_0) Continuous
     X_(0,_1) Continuous
     X_(0,_2) Continuous
     X_(0,_3) Continuous
     X_(1,_0) Continuous
     X_(1,_1) Continuous
     X_(1,_2) Continuous
     X_(1,_3) Continuous
```

```
X_(2,_0) Continuous
X_(2,_1) Continuous
X_(2,_2) Continuous
X_(2,_3) Continuous
```

[8]:
```python
# Change the objective
prob += x[0,0]    # minimize x[0,0]
```

```
/opt/anaconda3/envs/python377/lib/python3.7/site-packages/pulp/pulp.py:1544:
UserWarning: Overwriting previously set objective.
  warnings.warn("Overwriting previously set objective.")
```

[9]:
```python
prob.solve()
```

[9]: 1

[10]:
```python
LpStatus[prob.status]
```

[10]: 'Optimal'

[11]:
```python
print("Status:", LpStatus[prob.status])
print("Objective value: ", prob.objective.value())

for v in prob.variables():
    print(v.name,': ', v.value())
```

```
Status: Optimal
Objective value:  0.0
X_(0,_0) :  0.0
X_(0,_1) :  60.0
X_(0,_2) :  60.0
X_(0,_3) :  0.0
X_(1,_0) :  100.0
X_(1,_1) :  0.0
X_(1,_2) :  0.0
X_(1,_3) :  50.0
X_(2,_0) :  0.0
X_(2,_1) :  0.0
X_(2,_2) :  70.0
X_(2,_3) :  130.0
```

[12]:
```python
original_obj
```

```
[12]: 4*X_(0,_0) + 5*X_(0,_1) + 6*X_(0,_2) + 8*X_(0,_3) + 4*X_(1,_0) + 7*X_(1,_1) +
      9*X_(1,_2) + 2*X_(1,_3) + 5*X_(2,_0) + 8*X_(2,_1) + 7*X_(2,_2) + 6*X_(2,_3) + 0
```

```
[13]: original_obj.value()
```

```
[13]: 2430.0
```

6.5.6. Changing Constraint Coefficients

```
[14]: a = prob.constraints['Supply0']
```

```
[15]: a.changeRHS(500)
```

```
[16]: a
```

```
[16]: 1*X_(0,_0) + 1*X_(0,_1) + 1*X_(0,_2) + 1*X_(0,_3) + -500 = 0
```

```
[17]: prob.constraints['Supply0'].keys()
```

```
[17]: odict_keys([X_(0,_0), X_(0,_1), X_(0,_2), X_(0,_3)])
```

6.6 Multi Objective Optimization with PuLP

We consider two objectives and compute the pareto efficient frontier. We will implement the ε-constraint method. That is, we will add bounds based on an objective function and the optimize the alternate objective function.

6.6.0.1. Transportation Problem

Sets: - J = set of demand nodes - I = set of supply nodes

Parameters: - D_j: Demand at node j - S_i: Supply from node i - c_{ij}: cost per unit to send supply i to demand j

Variables: - x_{ij}: Transport volume from supply i to demand j (units)

- Objective function:

$$\min\left(obj1 = \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij}, \quad obj2 = x_{00} + x_{13} + x_{22} - x_{21} - x_{03}\right)$$

- Constraints:

$$\sum_{i=1}^{n} x_{ij} = S_i$$

$$\sum_{i=1}^{m} x_{ij} = D_j$$

- Decision variables:

$$x_{ij} \geq 0 \ \ i \in I, j \in J$$

6.6.0.2. Initial Optimization with PuLP

```python
[1]: from pulp import *

prob = LpProblem('Transportation_Problem', LpMinimize)


# Sets
n_suppliers = 3
n_buyers = 4

I = range(n_suppliers)
J = range(n_buyers)

routes = [(i, j) for i in I for j in J]


# Parameters
costs = [
    [4, 5, 6, 8],
    [4, 7, 9, 2],
    [5, 8, 7, 6]
]

supply = [120, 150, 200]
demand = [100, 60, 130, 180]


# Variables
x = LpVariable.dicts('X', routes, lowBound=0)

# Objectives
obj1 = lpSum([x[i, j] * costs[i][j] for i in I for j in J])
```

```
obj2 = x[0,0] + x[1,3] + x[2,2] - x[2,1] - x[0,3]

## start with first objective
prob += obj1


# Constraints

## Supply Constraints
for i in range(n_suppliers):
    prob += lpSum([x[i, j] for j in J]) == supply[i], f"Supply{i}"

## Demand Constraints
for j in range(n_buyers):
    prob += lpSum([x[i, j] for i in I]) == demand[j], f"Demand{j}"

# Solving problem
prob.solve();
```

```
[2]: print("Status:", LpStatus[prob.status])
     print("Objective value: ", prob.objective.value())

     for v in prob.variables():
         print(v.name,': ', v.value())
```

```
Status: Optimal
Objective value:  2130.0
X_(0,_0) :  60.0
X_(0,_1) :  60.0
X_(0,_2) :  0.0
X_(0,_3) :  0.0
X_(1,_0) :  0.0
X_(1,_1) :  0.0
X_(1,_2) :  0.0
X_(1,_3) :  150.0
X_(2,_0) :  40.0
X_(2,_1) :  0.0
X_(2,_2) :  130.0
X_(2,_3) :  30.0
```

```
[3]: # Record objective value
     obj1_opt = obj1.value()
     obj1_opt
```

[3]: 2130.0

[4]:
```python
# Add both objective values to a list and also the solution
obj1_vals = [obj1.value()]
obj2_vals = [obj2.value()]
feasible_points = [prob.variables()]
```

[5]:
```python
# Change objective functions and compute optimal objective value for obj2
prob += obj2
prob.solve()

obj2_opt = obj2.value()
obj2_opt
```

```
/opt/anaconda3/envs/python377/lib/python3.7/site-packages/pulp/pulp.py:1537:
UserWarning: Overwriting previously set objective.
  warnings.warn("Overwriting previously set objective.")
```

[5]: -180.0

[6]:
```python
# Append these values to the lists
obj1_vals.append(obj1.value())
obj2_vals.append(obj2.value())
feasible_points.append(prob.variables())
```

6.6.1. Creating the Pareto Efficient Frontier

[7]:
```python
import numpy as np

# Create an inequality for objective 1
prob += obj1 <= obj1_opt, "Objective_bound1"
obj_constraint = prob.constraints["Objective_bound1"]
```

[8]:
```python
# Set to optimize objective 2
prob += obj2
```

```
/opt/anaconda3/envs/python377/lib/python3.7/site-packages/pulp/pulp.py:1537:
UserWarning: Overwriting previously set objective.
  warnings.warn("Overwriting previously set objective.")
```

[9]:
```python
# Adjusting objective bound of objective 1
```

```
r_values = np.arange(1,2000,10)
for r in r_values:
    obj_constraint.changeRHS(r + obj1_opt)
    if 1 == prob.solve():
        obj1_vals.append(obj1.value())
        obj2_vals.append(obj2.value())
        feasible_points.append(prob.variables())

# Remove objective 1 constraint
obj_constraint.changeRHS(0)
obj_constraint.clear()
```

[10]:
```
# Create constaint for objective 2
prob += obj2 <= obj2_opt, "Objective_bound2"
obj2_constraint = prob.constraints["Objective_bound2"]

# set objective to objective 1
prob += obj1
```

[11]:
```
# Adjusting objective bound of objective 2

r_values = np.arange(1,400,5) # may need to adjust this
for r in r_values:
    obj2_constraint.changeRHS(r*obj2_opt)
    if 1 == prob.solve():
        obj1_vals.append(obj1.value())
        obj2_vals.append(obj2.value())
        feasible_points.append(prob.variables())

# Remove objective 2 constraint
obj2_constraint.changeRHS(0)
obj2_constraint.clear()
```
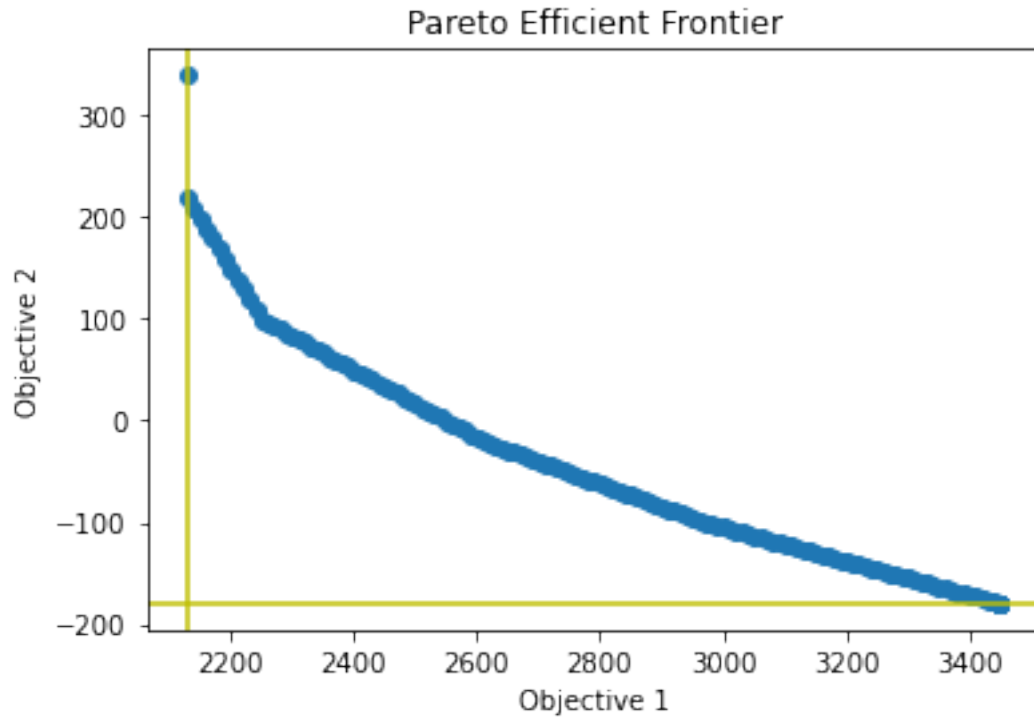
[12]:
```
import matplotlib.pyplot as plt
plt.scatter(obj1_vals, obj2_vals)
plt.axvline(x=obj1_opt, color = 'y')
plt.axhline(y=obj2_opt, color = 'y')
plt.title("Pareto Efficient Frontier")
plt.xlabel("Objective 1")
plt.ylabel("Objective 2")
```

[12]: Text(0, 0.5, 'Objective 2')

6.7 Comments

This code is a bit inefficient. It probably computes more pareto points than needed.

6.8 Jupyter Notebooks

Resources

- *https://github.com/mathinmse/mathinmse.github.io/blob/master/ Lecture-00B-Notebook-Basics.ipynb*
- *https://github.com/mathinmse/mathinmse.github.io/blob/master/ Lecture-00C-Writing-In-Jupyter.ipynb*

6.9 Reading and Writing

```
https://github.com/mathinmse/mathinmse.github.io/blob/master/
Lecture-10B-Reading-and-Writing-Data.ipynb
```

6.10 Python Crash Course

```
https://github.com/rpmuller/PythonCrashCourse
```

6.11 Gurobi

You can find lots great tutorial vidoes through Gurobi's youtube chanel.

6.11.1. Introductory Gurobi Examples

1. Gurobi-first-model.ipynb

2. Gurobi-knapsack1.ipynb

3. Gurobi-knapsack2.ipynb

4. Gurobi-knapsack3.ipynb

5. Gurobi-knapsack-SOS1.ipynb

6. Gurobi-SOS2-piecewise-linear.ipynb

Gurobi on GitHub

Gurobi Modeling Examples

Gurobi Log Tools

6.12 Plots, Pandas, and Geopandas

6.12.1. Matplotlib.pyplot

This is perhaps the most commonly used package for plotting. The functionality is a lot like MatLab's plotting functionality.

6.12.2. Pandas

Pandas is the most used package for manupulating data in python. You can load data from excel, csv, json, or other types of files, and manipulate data in a variety of ways.

6.12.3. Geopandas

Geopandas is a fantastic python package that allows you to interact with geospatial data, make plots, and analyze data. This typically involves loading shapefiles that describe a region (or many regions) into a geopandas dataframe. These function much like a regular pandas dataframe, but has a extra column for the geometry of the data and has other functionality for these types of dataframes.

Here are some great tutorials.

- tutorial
- tutorial github

6.13 Google OR Tools

Google has been building their own set of optimization tools and marketing them to solve operations research problems. They have a variety of type of solvers and have a number of example problems where they use the appropriate solver to solve a problem. This is a constantly evolving set of tool. See the link below examples on vehicle routing, scheduling, and more.

Google OR Tools

Youtube! Video!

7. CASE STUDY - Designing a campground - Simplex method

7.1 DESIGNING A CAMPGROUND - SIMPLEX METHOD

The Simplex method is probably the classic method of solving constraint optimization problems. We will use this solution approach, sometimes in modified form, over and over in this class, not just in this chapter.

Outcomes

You will learn
- *How to recognize linear programming (LP) problems*
- *Vocabulary of LP problems*
- *Graphical solution*
- *Algebraic solution*
- *Excel solution with the Excel Solver*

7.1.1. Case Study Description - Campground

You are opening a campground in the Florida Keys, and you are trying to make as much money as possible. You are planning a mix of RV sites, tent sites, and yurts. Let's assume you already own 10 acres, and that you can make \$80/ day profit on each RV, \$20/ day profit on each tent, and \$200 profit on each Yurt. However, there are restrictions:

1. Infrastructure takes up 20% of your site

2. You can have 20 RVs per acre, or

3. You can have 40 tents per acre, or

4. You can have 10 yurts per acre

5. You have a budget of \$100,000. It costs you \$1000 to develop an *RV* site, \$200 for a tent site, and \$8000 for a yurt.

6. Maintenance for the bath houses etc. is 15 min/ week/camper unit. You can afford 70hrs/week in maintenance help

7. Zoning ordinance requires you to have at least 20 tent sites What is the best layout for your campground, and how much profit can you make per day?

7.1.2. References

This case study was inspired by the Knights Key RV park in Florida. Read more about it here: http://www.knightskeyrvresortandmarina.com/news/ http://www.miaminewtimes.com/news/developers-plan-to-replace-rv-park-with-fivestar-resort-stirs-fears-hopes-in-keys-8038648

7.1.3. Solution Approach - Two Variables

We will again first look at a simpler problem by ignoring the yurts and only considering RV spaces and tents.

Let r be the number of RVs, t the number of tents, and $P(r,t)$ the profit. Your goal is to maximize $P(r,t) = 80r + 20t$. This function is called the objective function. The variables r and t are called decision

variables. You can see that in the current case $P(r,t)$ gets bigger if you increase r and/or t. If you picture a graph with r on the horizontal and t on the vertical axis, then the direction of increase for the objective function is to the top right.

Translating the relevant restrictions into equations gives

1. $r/20 + t/40 \leq 8$

2. $r \leq 160$

3. $t \leq 320$

4. $(r+t)/4 \leq 70$

5. $t \geq 20$

6. $r,t \geq 0$

7. $1,000r + 200t \leq 100,000$

Simplified

1. $2r + t \leq 320$

2. $r \leq 160$

3. $t \leq 320$

4. $5r + t \leq 500$

5. $r + t \leq 280$

6. $t \geq 20$

7. $r, t \geq 0$

These are called the functional constraints.

In addition, you can't have negative sites, so we have the non-negativity constraints

6. $r \geq 0$

7. $t \geq 0$.

A problem like the above with linear constraints and a linear objective function is called a linear programming problem.

7.1.4. Assumptions made about linear programming problem

PROPORTIONALITY   For both the objective function and the constraints, a change in a decision variable will result in a proportional change in the objective function or constraint. (Note that this rules out any exponents on the decision variables other than 1.)

ADDITIVITY   Both the objective function and the constraints are the sums of the respective changes in the decision variables (This means no multiplying different decision variables).

Basically, the proportionality and additivity assumptions are just fancy ways of saying that all functions in a linear programming problem are linear in the decision variables.

DIVISIBILITY   We are assuming that our decision variables can be non-integer, i.e. may take on fractional values. Problems with an integer constraint are called integer programming problems, we will only touch on them briefly later. Certainty We act as if the value assigned to each parameter is known, precise, and constant over time. This is rarely the case, so we need to compensate for that by performing sensitivity analysis. Basically, we need to investigate how much it affects our solution if the parameters change.

7.1.5. Graphical Simplex solution procedure

We will start with some vocabulary:

- Feasible solution: A solution for which all constraints are satisfied, not necessarily an optimal Solution.

- Infeasible solution: A solution that violates at least one constraint.

- Optimal solution: a solution that optimizes (could be a minimum or maximum, depending on your problem) the objective function. There may or may not be an optimal solution.
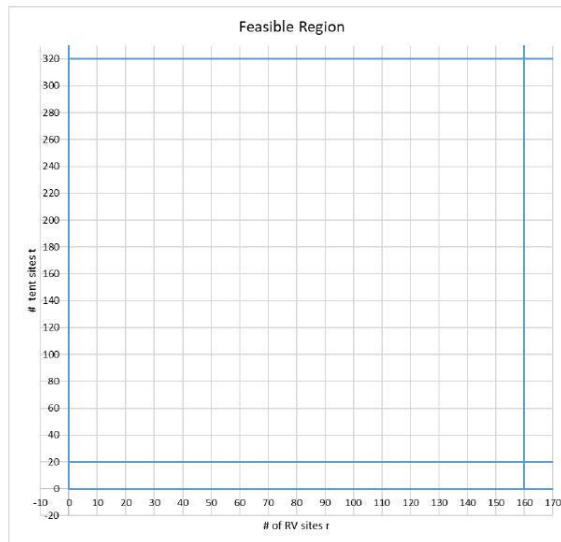
Feasible region: The set of all feasible solutions

- Corner point: the intersection of two or more constraints

- Corner point feasible solution CPF solution: A solution that occurs at a corner of the feasible region

The first step in the graphical solution procedure is to draw the feasible region (note that this gets really ugly if you have three or more variables).

The non-negativity constraints mean that we are looking for a solution in the first quadrant only. The constraints 2) $r \leq 160, 3) t \leq 320$, and 7) $t \geq 20$ mean you have to stay left of the line $r = 160$, below the line $t = 320$ and above the line $t = 20$. You see that the constraint $t \geq 20$ dominates the constraint $t \geq 0$, so the latter is redundant.
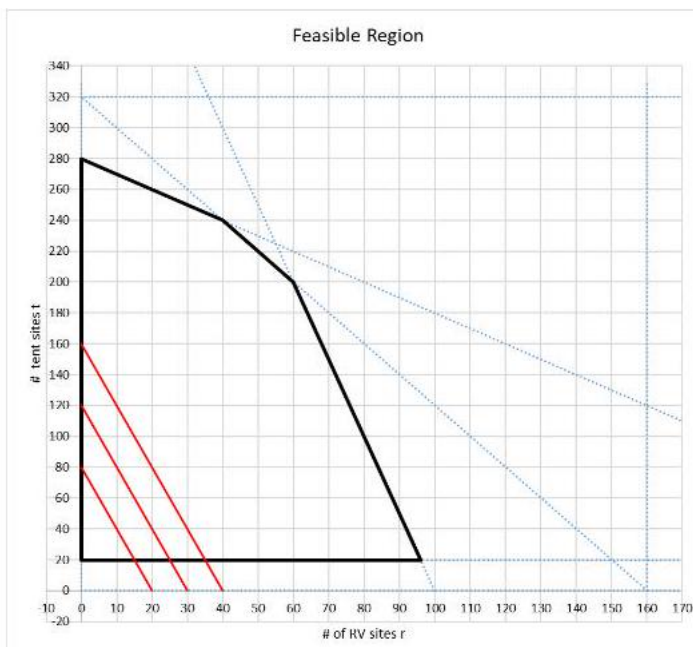

Feasible Region

Adding the remaining constraints yields this graph: Go ahead, shade the feasible region and identify all redundant constraints.
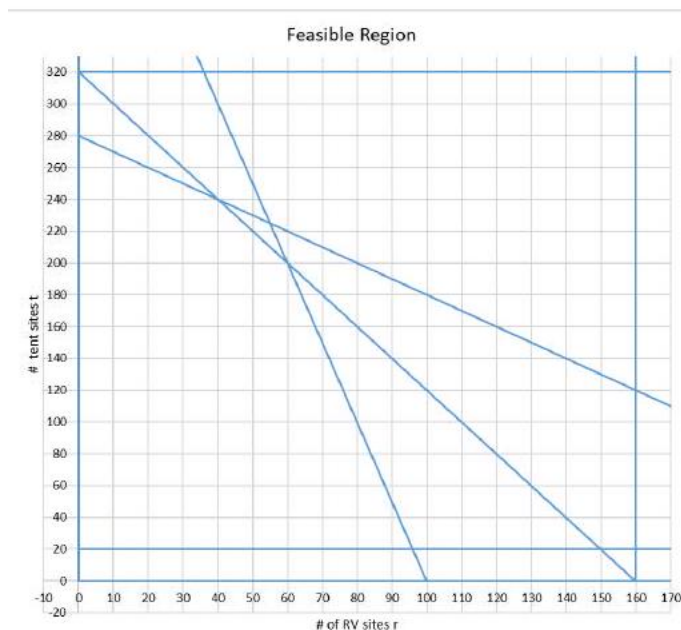

Feasible Region

All the points in the feasible region are possible solutions. Your job is to pick (one of) the best solutions. Note that there may not be a single best solution but rather several optimal solutions.

We have not yet used the objective function $P(r,t) = 80r + 20t$. Because we do not have a value for $P(r,t)$, so we will draw $P(r,t)$ for a few random values of $P(r,t)$ to get an idea of what it looks like. For $P(r,t) = 1600, 2400, 3200$ we get the lines shown in the next picture. Note that they are all parallel, and that the lines corresponding to the larger value of $P(r,t)$ move to the top left. The direction of increase is just as we expected, to the top right.

Feasible Region

As you move the line for the $P(r,t)$ to the right you increase your profit. But you also have to stay in the feasible region. Convince yourself that one of two cases will occur: either a unique optimal solution will be found at a corner point, or infinitely many optimal solutions returning the same maximum value for $P(r,t)$ will be found along a section of the boundary of the feasible region that includes two corner points. Therefore, we need to compute the corner points, i.e. the intersections of the constraints, and move $P(r,t)$ as far to the right as possible without leaving the feasible region.

Feasible Region
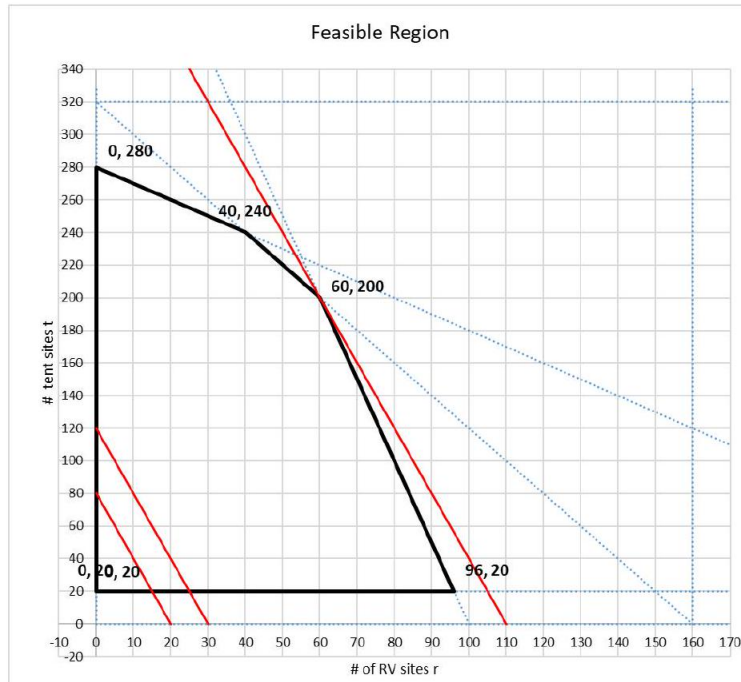
From the picture above, you can see that the optimal solution will be at the intersection of the lines corresponding to constraints 1) and 5).

$$r/20 + t/40 \leq 8 \text{ and } 1000t + 200t \leq 100,000$$

which is at the point $r = 60, t = 200$. (Now would be a good time to review how to solve systems of equations....). This gives us a profit $P(60, 200) = 60 \cdot \$80 + 200 \cdot \$20 = \$8800$.

7.1.6. Stating the Solution

In OR, typically someone hires you to work your problem, and then expects you to give the answer in the context of the problem. You can't just say "the solution is at $r = 60, t = 200, P(r,t) = 8800$". Give the answer like this:

"To maximize potential profit, the campground should have 60RV sites and 200 tent sites. In that case, the potential profit per day, assuming full occupancy, is $8800. You are limited by the available land and budget available, not by available labor or any zoning rules. You will have to hire 65 hours of help per week (260 camping units at 15 minutes/week/unit)."

7.1.7. Refinements to the graphical solution

One "brute force" approach to the graphical solution method would be to compute all intersections of all constraints, check if that corner is in the feasible region, and then compute the objective function at those points. However, the number of intersections increases quadratically with the number of lines ($\frac{n(n+1)}{2}$ for n non-parallel lines), so this approach quickly gets out of hand. Instead, the idea is to start at an easy to find corner of the feasible region. Often, the origin works. From that point, check the adjacent feasible corner points and move to the "best". Continue until there is no more improvement. Here is what that would look like in our case:

We start at a simple corner. Usually, people use the origin, which is not on the feasible region here, so we start at $(0, 20)$. Adjacent corners are $(0, 280)$ and $(96, 20)$ with $P(0, 20) = 400, P(0, 280) = 5600$, and $P(96, 20) = 8080$. $(96, 20)$ is best, so we move there. Next, check the adjacent corner $(60, 200)$. It gives $P(60, 200) = 8800$ which is an improvement, so move there. Check the next adjacent corner $(40, 240)$ It gives $P(40, 240) = 8000$. This is worse, so $(60, 200)$ is the optimal solution.
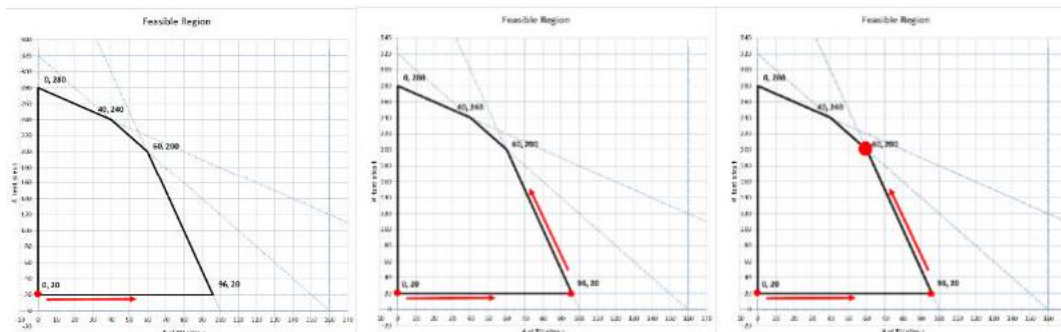
There is of course a big problem with this approach. It relies on us having a graph of the feasible region and being able to see the adjacent feasible corner points. If you are dealing with a large number of constraints and variables, this is not possible. We therefor take the idea of looking at adjacent feasible corners and moving towards the one that gives the best value for the objective function, and translate it into an algebraic method.

SOLUTION APPROACH - ALGEBRAIC SIMPLEX METHOD   As these are only three variables, we could still draw the feasible region, now a solid bound by planes. However, we need an approach that works for any number of variables. The key to this method is the fact that an optimal solution will occur at a corner point of the feasible region. While the n-dimensional proof is beyond the scope of this class, this fact should be intuitively clear in the 2-d and 3-d cases.

We will first demonstrate the algebraic method on the two-variable problem (RV and tent only) and then solve the full problem.

CORNER POINTS, INTERIOR CORNER POINTS, SLACK VARIABLES   We introduce slack variables to turn the inequalities into equalities. Basically, a slack variable lets you know how close you are to maxing out the constraint. In the current case, we have:

Constraints	Augmented constraints
1) $4r+t \le 320$	1) $2r+t+s_1 = 320$
2) $5r+t \le 500$	2) $5r+t+s_2 = 500$
3) $r+t \le 280$	3) $r+t+s_3 = 280$
4) $t \ge 20$	4) $t - s_4 = 20$
5) $r,t \ge 0$	5) $r,t,s_1,s_2,s_3,s_4 \ge 0$

A point is a corner point if it sits at the intersection of two or more constraints, i.e. if two or more slack variables are zero. A point is a feasible solution (i.e. inside the feasible region) if all slack variables are non-negative. A point is outside the feasible region if any slack variable is negative. We will from now on express each point as (r,t,s_1,s_2,s_3,s_4). Given r and t, one computes the values of the slack variables from the augmented constraints.

Examples: $r = 100, t = 20$ → $(100,20,100,-20,140,0)$ outside the feasible region

$r = 10, t = 30$ → $(10,30,270,420,$

$r = 150, t = 20$ → $(150,20,0,-270,110,0)$ corner outside fe

INITIALIZING   Because $(0,0)$ is not on the feasible region, we again start at $(r,t) = (0,20)$, which has the augmented form $(0,20,300,480,240,0)$. We are sitting on the intersection of the lines $r = 0$ and $t = 20$.

THE ADJUSTED OBJECTIVE FUNCTION   Note: If the origin is a feasible corner point and you start at the origin, you can skip this step.

We are sitting on the intersection of the lines $r = 0$ and $t = 20$. To reach the next adjacent corner, we have to move along one of those lines, but which one? If we move along the line $r = 0$, we move away from the line $t = 20$, which is the same as saying we are increasing the corresponding slack, s_4. If we move along the line $t = 20$, we increase r. We want to choose the direction of increase that gives us the fastest increase in P. The original objective function is $P(r,t) = 80r + 20t$, which does not let us see what happens if we increase s_4 We have to rewrite $P(r,t)$ in terms of r and s_4.

Using equation 4) we get $t = 20 + s_4$, and thus $P(r,s_4) = 80r + 20(20 + s_4) = 400 + 80r + 20s_4$

Determining which way to move

We want to choose the direction of increase that gives us the fastest increase in P. The objective function is $P(r,s_4) = 400 + 80r + 20s_4$. Because the variable r has the biggest coefficient, 80 , an increase in r should give the best return.

Determining how far to move - the next corner

We will leave s_4 at is current value, 0 , and increase r as much as possible without leaving the feasible region, i.e. without having t, s_1, s_2, s_3, s_4 become negative.

1. $2r + t + s_1 = 320$
   $s_1 = 320 - 20 - 2r \geq 0$, so $r \leq 150$
   $s_2 = 500 - 20 - 5r \geq 0$, so $r \leq 96$

2. $5r + t + s_2 = 500$
   $S_3 = 280 - 20 - r \geq 0$, so $r \leq 240$

3. $r + t + s_3 = 280$

4. $t - S_4 = 20$
   $t = 20$

So, r can be increased to 96 .

Augmented form of the next corner

Using $r = 96, s_4 = 0$, and substituting into the equations $1 - 4$, we have the augmented point $(96, 20, 108, 0, 164, 0)$. Note that this is the same corner $(96, 20)$ we used above.

The adjusted objective function

We are now sitting on the intersection of the lines $5r + t + s_2 = 500$ and $t = 20$. To reach the next adjacent corner, we have to move along one of those lines, which means either increasing s_2 or s_4. We have to rewrite $P(r,t)$ in terms of s_2 and s_4.

Using equations 2 and 4 , we find that $5r = 500 - t - s_2$ and $t = 20 + s_4$, which yields $5r = 480 - s_4 - s_2$. Substituting into P :

$P(s_2, s_4) = 400 + 80r + 20s_4 = 400 + 16(480 - s_4 - s_2) = 20s_4 = 8080 + 4s_4 - 16s_2$

Now we will repeat the above steps until the solution/objective function can no longer be improved upon.

DETERMINING WHICH WAY TO MOVE   Because the S_4 has the only positive coefficient, this is the only direction that will yield an increase in P.

DETERMINING HOW FAR TO MOVE - THE NEXT CORNER   We will leave s_2 at is current value, 0 , and increase s_4 as much as possible without leaving the feasible region.

1. $2r + t + s_1 = 320 \quad s_1 = 320 - t - 2r$

2. $5r + t + s_2 = 500$
   $s_2 = 500 - t - 5r$

3. $r + t + s_3 = 280$
   $s_3 = 280 - t - r$

4. $t - s_4 = 20 \, t = 20 + s_4$

We use that $s_2 = 0$ and $t = 20 + s_4$. This gives the set of equations:

1. $s_1 = 108 - 0.6s_4$
   ≥ 0, so $s_4 \leq 96$
   ≥ 0, so $S_4 \leq 480$

2. $r = 96 - 0.2s_4$
   ≥ 0, so $S_4 \leq 205$

3. $s_3 = 164 - 0.8 \, s_4$

4. $t = 20 + s_4$

≥ 0, so $s_4 \geq -20$ (as 20 is positive, this is true anyway) So S_4 can be increase up to 180 .

AUGMENTED FORM OF THE NEXT CORNER   Using $s_2 = 0, s_4 = 180$, and substituting into the equations $1 - 4$, we have the augmented point $(60, 200, 0, 0, 20, 180)$. Note that this is the second corner $(60, 200)$ we used above.

THE ADJUSTED OBJECTIVE FUNCTION   We again re-write the objective function, this time in terms of s_1 and s_2 : $P(s_1, s_2) = 8080 + 4s_4 - 16s_2 = 8080 + 4(180 - 1.6s_1) - 16s_2 = 8800 - 6.6s_1 - 16s_2$. Note that increasing either s_1 or s_2 will decrease the value of P, so we have reached the maximum. As s_1 and s_2 are non-negative, we can also see that the maximum for P occurs when s_1 and s_2 are zero, at $P = 8800$. Again, this is the same answer we arrived at earlier.

A nice side effect is that we can tell which constraints are holding us back, namely those associated with the zero slack variables s_1 and s_2. s_1 corresponds to the space limitations, and s_2 to the budget restrictions.

Solving the full problem

We are now ready to look at the original problem. We will assume you went to the bank and got a loan for $246,000 to supplement your original budget. Here are the constraints again:

1. Infrastructure takes up 20% of your site

2. You can have 20RV s per acre, or

3. You can have 40 tents per acre, or

4. You can have 10 yurts per acre

5. You have a budget of $346,000. It costs you $1000 to develop an RV site, $200 for a tent site, and $8000 for a yurt.

6. Maintenance for the bath houses etc. is 15 min/ week/camper unit, you can afford 70hrs/ week in maintenance help

7. Zoning ordinance requires you to have at least 20 tent sites

Functional Constraints	Simplified Constraints	Augmented Constraints
$r/20 + t/40 + y/10 \leq 8$	$2r + t + 4y \leq 320$	$2r + t + 4y + s_1 = 320$
$1000r + 200t + 8000y \leq 346,000$	$5r + t + 40y \leq 1730$	$5r + t + 40y + s_2 = 1750$
$(r + t + y)/4 \leq 70$	$r + t + y \leq 280$	$r + t + y + s_3 = 280$
$t \geq 20$	$t \geq 20$	$t - s_4 = 20$

NON-NEGATIVITY CONSTRAINTS: $r, t, y_1, s_1, s_2, s_3, s_4 \geq 0$

OBJECTIVE FUNCTION: Maximize $P(r,t,y) = 80r + 20t + 200y$

INITIALIZING   Because $(0,0,0)$ is not on the feasible region, we start at $(r,t,y) = (0,20,0)$, which has the augmented form $(0,20,0,300,480,240,0)$. We are sitting on the intersection of the planes $r = 0, t = 20, y = 0$. The augmented form of this corner is $(0,20,0,300,1710,260,0)$

THE   ADJUSTED   OBJECTIVE   FUNCTION   Rewriting P as $P(r,y,s_4)$ gives: $P(P(r,y,s_4) = 400 + 80r + 200y + 20s_4$

DETERMINING WHICH WAY TO MOVE   Looking at the coefficients of r, y, s_4 in the objective function, we find that we should increase y and leave r and $s_4 = 0$

DETERMINING HOW FAR TO MOVE - THE NEXT CORNER   Using $r = 0$ and $s_4 = 0$, the constraints become

$$
\begin{aligned}
t + 4y + s_1 &= 320 & 4y + s_1 &= 300 & s_1 &= 300 - 4y & \geq 0 & \rightarrow y \leq 75 \\
t + 40y + s_2 &= 1750 & 40y + s_2 &= 1730 & s_2 &= 1730 - 40y & \geq 0 & \rightarrow y \leq 42.75 \\
t + y + s_3 &= 280 & y + s_3 &= 260 & s_3 &= 260 - y & \geq 0 & \rightarrow y \leq 260
\end{aligned}
$$

$t = 20$

So, y can be increased up to 42.75

AUGMENTED FORM OF THE NEXT CORNER   With $r = 0$, $s_4 = 0$, and $y = 42.75$, we find the new augmented corner to be $(0, 20, 42.75, 129, 0, 217.25, 0)$.

THE ADJUSTED OBJECTIVE FUNCTION   We need to rewrite the objective function in terms of r, s_2, and s_4:

$$P(r, s_2, s_4) = 8950 + 55r + 15s_4 - 5s_2$$

The solution is not optimal yet, (there are still positive coefficients in the objective function), so we keep going.

DETERMINING WHICH WAY TO MOVE   We see that we should increase r and leave s_2 and $s_4 = 0$.

Determining how far to move - the next corner

With s_2 and $s_4 = 0$, we have

$$
\begin{aligned}
2r + t + 4y + s_1 &= 320 & 2r + 4y + s_1 &= 300 & s_1 &= 129 - 1.5r \geq 0 & \rightarrow r \leq 86 \\
5r + t + 40y &= 1750 & 5r + 40y &= 1730 & 40y &= 1710 - 5r \\
r + t + y + s_3 &= 280 & r + y + s_3 &= 260 & s_3 &= 236.25 - 0.875r \geq 0 & \rightarrow r \leq 342
\end{aligned}
$$

$t = 20$

So, y can be increased up to 86

AUGMENTED FORM OF THE NEXT CORNER   With $y = 86, s_2 = 2$, and $s_4 = 0$ we find the new augmented corner to be $(86, 20, 32, 0, 0, 142, 0)$

THE ADJUSTED OBJECTIVE FUNCTION   We need to rewrite the objective function in terms of s_1, s_2, and s_4 :

$$P(s_1, s_2, s_4) = 13680 - 36\tfrac{2}{3}s_1 - 1\tfrac{1}{3}s_2 - 18s_4$$

Note that now all variables have negative coefficients, so we cannot increase the value of P past 13680. The first, second, and fourth constraints are maxed out; we are limited in our ability to increase the profit by space, money, and zoning restrictions.

Stating the Solution

To maximize potential profit, the campground should have 86 RV sites, 20 tent sites, and 32 yurts. This will take an initial investment of $346,000. The potential profit per day, assuming full occupancy, is $13,680. You are limited by the available land and budget available and the zoning law requiring 20 tent sites. You will have to hire 34.5 hours of help per week (138 camping units at 15 minutes/week/unit)."

7.1.8. Solution Approach - Using Excel

Now that we know how the solution method works, we can use Excel to do the work for us. First, we must set up the work sheet. One way that works well is shown on the next page. The fields highlighted in green are necessary, the others serve to explain and label what we are doing.

	A	B	C	D	E	F	G
1	objective function	=B2*80+B3*20+B4*200					
2	# of RV sites, r	0		Slack variables			
3	# of tent sites, t	0		s1	s2	s3	s4
4	# of yurts, y	0		=F8-D8	=F9-D9	=F10-D10	=F11-D11
5							
6							
7	Original functional constraints	Simplified constraints	Constraints w/ slack vars				
8	r/20+t/40+y/10 ≤ 8	2r+t+4y=320	2r+t+4y+s1=320	=2*B2+B3+4*B4	<=	320	
9	1,000r+200t + 8000y ≤ 346,000	5r+t+40y=1730	5r+t+40y+s2=1730	=5*B2+B3+40*B4	<=	1730	
10	(r+t +y)/4 ≤ 70	r+t+y=280	r+t+y+s3=280	=B2+B3+B4	<=	280	
11	t ≥ 20	t=20	t-s4=20	=B3	>=	20	
12							

Excel has a built-in Solver under the Data tab (if you don't see it, you have to add it in. Go to file/options/Add-ins/Manage Excel Add-ins/Solver Add-in). If you choose "show iteration results in the options tab, the solver will stop at each iteration and show you the corner/solution it has arrived found at that step. You will see that the solver goes through the same steps and corner points as we did when we worked the problem "by hand".

	A	B	C	D	E	F	G
1	objective function	0			Slack variables		
2	# of RV sites, r	0					
3	# of tent sites, t	0		s1	s2	s3	s4
4	# of yurts, y	0		320	1730	280	20
5							

	Original functional constraints	Simplified constraints	Constraints w/ slack vars			
7	Original functional constraints	Simplified constraints	Constraints w/ slack vars			
8	r/20+t/40+y/10 ≤ 8	2r+t+4y=320	2r+t+4y+s1=320	0	<=	320
9	1,000r+200t + 8000y ≤ 346,000	5r+t+40y=1730	5r+t+40y+s2=1730	0	<=	1730
10	(r + t +y)/4 ≤ 70	r+t+y=280	r+t+y+s3=280	0	<=	280
11	t ≥ 20	t=20	t-s4=20	0	>=	20

$AllMethods|GRGNonlinear|Evolutionery|$

12	Solver Parameters				
13					
14					
15	Set Objective:	SBS1			↑
16					
17	To: ⦿ Max ◯ Min ◯ Value Of:	0			

Constrainterecisioni.   0.000001.

- Automatic scaling

Whe Automatic scoling

Whow lteration results

Solving with integer Constraints

Solving with integer constraints

18	By Changing Variable Cells:	$
19	SBS2:SBS4	
20	Subject to the Constraints:	SDS11 = SFs11
21		

☐ Ignoce integer constraints

By Changing Variable Cells:

 – T 2

1

(2.   Solving Limits

4

22	Subject to the Constraints:		
23	SDS11 >= SFS11		Add
	SDS8:SDS10 <= SFS8:SFS10		
24			Change
25			
26			Delete
27			
28			Reset All
29			
30			Load/Save
31			

Subbject to the Constraints:

Max Iime |Seconds|:

Iterations:

Evolutionary and Integer Constraints:

Evolutionary and inteE

Lox subprobiems:

Mox Ecosible Solutions:

Options

-
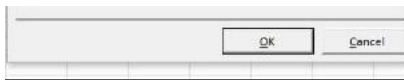
☐ Make Unconstrained Variables Non-Negative

Select a Solving
Method:

ORtions

Solving Method

Solving Method Select the GRG Nonlinear engine for Solver Problems that are smooth nonlinear. Select the
Simplex engine for linear Solver Problems, and select the Evolutionary engine for Solver problems that are non-smooth.

Help

Solve

Close

8. Simplex Method

Definition 8.1: Standard Form

A linear program is in standard form *if is written as*

$$\max \ c^\top x$$
$$\text{s.t.} \ Ax = b$$
$$x \geq 0.$$

Definition 8.2: Extreme Point

A point x in a convex set C is called an extreme point *if it cannot be written as a strict convex combination of other points in C.*

Theorem 8.3: Optimal Extreme Point - Bounded Case

Consider a linear optimization problem in standard form. Suppose that the feasible region is bounded and non-empty.
Then there exists an optimal solution at an extreme point of the feasible region.

Proof. [Proof Sketch]

♠

Definition 8.4: Basic solution

A basic solution to $Ax = b$ is obtained by setting $n - m$ variables equal to 0 and solving for the values of the remaining m variables. This assumes that the setting $n - m$ variables equal to 0 yields unique values for the remaining m variables or, equivalently, the columns of the remaining m variables are linearly independent.

Example 8.5

Consider the problem

$$\max \quad Z = -5X - 7Y$$
$$\text{s.t.} \quad X + 3Y \geq 6$$
$$5X + 2Y \geq 10$$
$$Y \leq 4$$
$$X, Y \geq 0$$
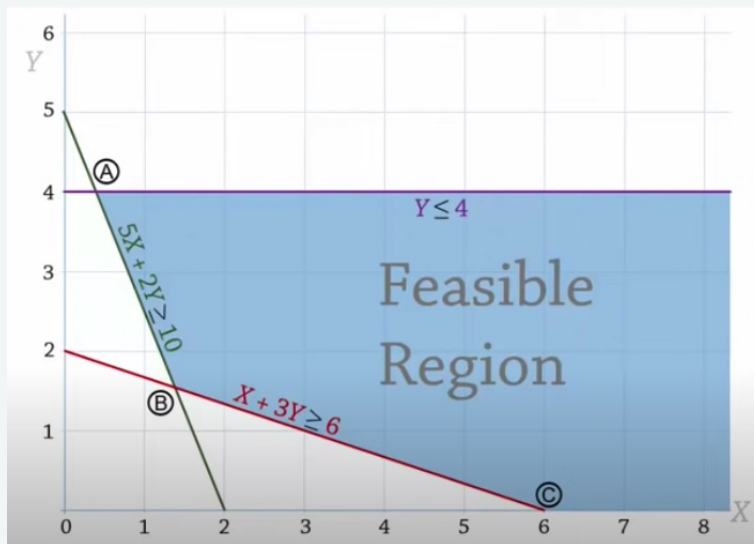
We begin by converting this problem to standard form.

$$\max \quad Z = -5X - 7Y + 0s_1 + 0s_2 + 0s_3$$
$$\text{s.t.} \quad X + 3Y - s_1 = 6$$
$$5X + 2Y - s_2 = 10$$
$$Y + s_3 = 4$$
$$X, Y, s_1, s_2, s_3 \geq 0$$

Thus, we can write this problem in matrix form with

$$\max \begin{bmatrix} -5 \\ -7 \\ 0 \\ 0 \\ 0 \end{bmatrix}^\top \begin{bmatrix} X \\ Y \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} \tag{8.1}$$

$$\begin{bmatrix} 1 & 3 & -1 & 0 & 0 \\ 5 & 2 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 10 \\ 4 \end{bmatrix} \tag{8.2}$$

$$(X, Y, s_1, s_2, s_3) \geq 0 \tag{8.3}$$

Definition 8.6: Basic feasible solution

Any basic solution in which all the variables are non-negative is a basic feasible solution.

Theorem 8.7: BFS iff extreme

A point in the feasible region of an LP is an extreme point if and only if it is a basic feasible solution to the LP.

To prove this theorem, we

Theorem 8.8: Representation

Consider an LP in standard form, having bfs $b_1, \ldots, b_k$. Any point x in the LP's feasible region may be written in the form

$$x = d + \sum_{i=1}^{k} \sigma_i b_i$$

where d is 0 or a direction of unboundedness and $\sum_{i=1}^{k} \sigma_i = 1$ and $\sigma_i \geq 0$.

Theorem 8.9: Optimal bfs

If an LP has an optimal solution, then it has an optimal bfs.

Proof. Let x be an optimal solution. Then

$$x = d + \sum_{i=1}^{k} \sigma_i b_i$$

where d is 0 or a direction of unboundeness.

- If $c^\top d > 0$, the $x' = \lambda d + \sum_{i=1}^k \sigma_i b_i$ has bigger objective value for $|lambda > 1$, which is a contradiction since x was optimal.

- If $c^\top d < 0$, the $x'' = \sum_{i=1}^k \sigma_i b_i$ has a bigger objective value, which is a contradiction since x was optimal.

Thus, we conclude that $c^\top d = 0$.

Since

$$c^\top x \geq c^\top b_i$$

for all $i = 1, \ldots, k$, we can conclude that

$$c^\top x = c^\top b_i$$

for all i such that $\sigma_i > 0$. Hence, there exists an optimal basic feasible solution. ♠

8.1 The Simplex Method

The Simplex Method **Lab Objective:** *The Simplex Method is a straightforward algorithm for finding optimal solutions to optimization problems with linear constraints and cost functions. Because of its simplicity and applicability, this algorithm has been named one of the most important algorithms invented within the last 100 years. In this lab we implement a standard Simplex solver for the primal problem.*

Standard Form

The Simplex Algorithm accepts a linear constrained optimization problem, also called a *linear program*, in the form given below:

$$
\begin{aligned}
\text{minimize} \quad & \mathbf{c}^\mathsf{T}\mathbf{x} \\
\text{subject to} \quad & A\mathbf{x} \leq \mathbf{b} \\
& \mathbf{x} \geq 0
\end{aligned}
$$

Note that any linear program can be converted to standard form, so there is no loss of generality in restricting our attention to this particular formulation.

Such an optimization problem defines a region in space called the *feasible region*, the set of points satisfying the constraints. Because the constraints are all linear, the feasible region forms a geometric object called a *polytope*, having flat faces and edges (see Figure 8.1). The Simplex Algorithm jumps among the vertices of the feasible region searching for an optimal point. It does this by moving along the edges of the feasible region in such a way that the objective function is always increased after each move.
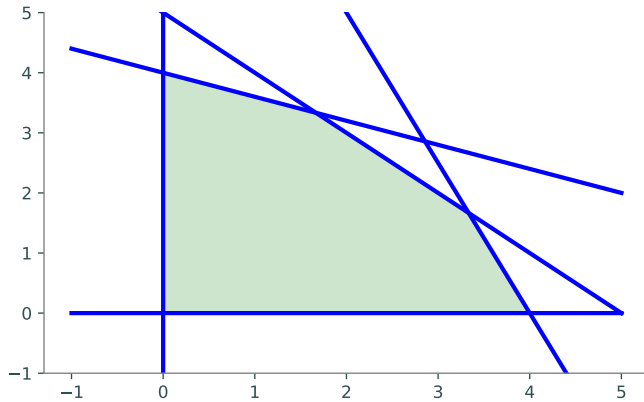
(a) The feasible region for a linear program with 2-dimensional constraints.
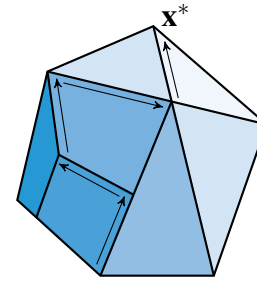
(b) The feasible region for a linear program with 3-dimensional constraints.

Figure 8.1: If an optimal point exists, it is one of the vertices of the polyhedron. The simplex algorithm searches for optimal points by moving between adjacent vertices in a direction that increases the value of the objective function until it finds an optimal vertex.

Implementing the Simplex Algorithm is straightforward, provided one carefully follows the procedure. We will break the algorithm into several small steps, and write a function to perform each one. To become familiar with the execution of the Simplex algorithm, it is helpful to work several examples by hand.

The Simplex Solver

Our program will be more lengthy than many other lab exercises and will consist of a collection of functions working together to produce a final result. It is important to clearly define the task of each function and how all the functions will work together. If this program is written haphazardly, it will be much longer and more difficult to read than it needs to be. We will walk you through the steps of implementing the Simplex Algorithm as a Python class.

For demonstration purposes, we will use the following linear program.

$$
\begin{aligned}
\text{minimize} \quad & -3x_0 - 2x_1 \\
\text{subject to} \quad & x_0 - x_1 \leq 2 \\
& 3x_0 + x_1 \leq 5 \\
& 4x_0 + 3x_1 \leq 7 \\
& x_0, x_1 \geq 0.
\end{aligned}
$$

Accepting a Linear Program

Our first task is to determine if we can even use the Simplex algorithm. Assuming that the problem is presented to us in standard form, we need to check that the feasible region includes the origin. For now, we only check for feasibility at the origin. A more robust solver sets up the auxiliary problem and solves it to find a starting point if the origin is infeasible.

> **Problem 8.10: Check feasibility at the origin.**
>
> *Write a class that accepts the arrays* $\mathbf{c}$, A, *and* $\mathbf{b}$ *of a linear optimization problem in standard form. In the constructor, check that the system is feasible at the origin. That is, check that* $A\mathbf{x} \preceq \mathbf{b}$ *when* $\mathbf{x} = 0$. *Raise a* `ValueError` *if the problem is not feasible at the origin.*

Adding Slack Variables

The next step is to convert the inequality constraints $A\mathbf{x} \leq \mathbf{b}$ into equality constraints by introducing a slack variable for each constraint equation. If the constraint matrix A is an $m \times n$ matrix, then there are m slack variables, one for each row of A. Grouping all of the slack variables into a vector $\mathbf{w}$ of length m, the constraints now take the form $A\mathbf{x} + \mathbf{w} = \mathbf{b}$. In our example, we have

$$\mathbf{w} = \begin{bmatrix} x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

When adding slack variables, it is useful to represent all of your variables, both the original primal variables and the additional slack variables, in a convenient manner. One effective way is to refer to a variable by its subscript. For example, we can use the integers 0 through $n - 1$ to refer to the original (non-slack) variables x_0 through x_{n-1}, and we can use the integers n through $n + m - 1$ to track the slack variables (where the slack variable corresponding to the ith row of the constraint matrix is represented by the index $n + i - 1$).

We also need some way to track which variables are *independent* (non-zero) and which variables are *dependent* (those that have value 0). This can be done using the objective function. At anytime during the optimization process, the non-zero variables in the objective function are *independent* and all other variables are *dependent*.

Creating a Dictionary

After we have determined that our program is feasible, we need to create the *dictionary* (sometimes called the *tableau*), a matrix to track the state of the algorithm.

There are many different ways to build your dictionary. One way is to mimic the dictionary that is often used when performing the Simplex Algorithm by hand. To do this we will set the corresponding dependent variable equations to 0. For example, if x_5 were a dependent variable we would expect to see a -1 in the column that represents x_5. Define

$$\bar{A} = \begin{bmatrix} A & I_m \end{bmatrix},$$

where I_m is the $m \times m$ identity matrix we will use to represent our slack variables, and define

$$\bar{c} = \begin{bmatrix} c \\ 0 \end{bmatrix}.$$

That is, $\bar{c} \in \mathbb{R}^{n+m}$ such that the first n entries are c and the final m entries are zeros. Then the initial dictionary has the form

$$D = \begin{bmatrix} 0 & \bar{c}^T \\ b & -\bar{A} \end{bmatrix} \tag{8.1}$$

The columns of the dictionary correspond to each of the variables (both primal and slack), and the rows of the dictionary correspond to the dependent variables.

For our example the initial dictionary is

$$D = \begin{bmatrix} 0 & -3 & -2 & 0 & 0 & 0 \\ 2 & -1 & 1 & -1 & 0 & 0 \\ 5 & -3 & -1 & 0 & -1 & 0 \\ 7 & -4 & -3 & 0 & 0 & -1 \end{bmatrix}.$$

The advantage of using this kind of dictionary is that it is easy to check the progress of your algorithm by hand.

Problem 8.11: Initialize the dictionary.

dd a method to your Simplex solver that takes in arrays c, A, and b to create the initial dictionary (D) as a NumPy array.

8.1.1. Pivoting

Pivoting is the mechanism that really makes Simplex useful. Pivoting refers to the act of swapping dependent and independent variables, and transforming the dictionary appropriately. This has the effect of moving from one vertex of the feasible polytope to another vertex in a way that increases the value of the objective function. Depending on how you store your variables, you may need to modify a few different parts of your solver to reflect this swapping.

When initiating a pivot, you need to determine which variables will be swapped. In the dictionary representation, you first find a specific element on which to pivot, and the row and column that contain the pivot element correspond to the variables that need to be swapped. Row operations are then performed on the dictionary so that the pivot column becomes a negative elementary vector.

Let's break it down, starting with the pivot selection. We need to use some care when choosing the pivot element. To find the pivot column, search from left to right along the top row of the dictionary (ignoring the first column), and stop once you encounter the first negative value. The index corresponding to this column will be designated the *entering index*, since after the full pivot operation, it will enter the basis and become a dependent variable.

Using our initial dictionary D in the example, we stop at the second column:

$$D = \begin{bmatrix} 0 & \boxed{-3} & -2 & 0 & 0 & 0 \\ 2 & -1 & 1 & -1 & 0 & 0 \\ 5 & -3 & -1 & 0 & -1 & 0 \\ 7 & -4 & -3 & 0 & 0 & -1 \end{bmatrix}$$

We now know that our pivot element will be found in the second column. The entering index is thus 1.

Next, we select the pivot element from among the negative entries in the pivot column (ignoring the entry in the first row). *If all entries in the pivot column are non-negative, the problem is unbounded and has no solution.* In this case, the algorithm should terminate. Otherwise, assuming our pivot column is the jth column of the dictionary and that the negative entries of this column are $D_{i_1,j}, D_{i_2,j}, \ldots, D_{i_k,j}$, we calculate the ratios

$$\frac{-D_{i_1,0}}{D_{i_1,j}}, \frac{-D_{i_2,0}}{D_{i_2,j}}, \ldots, \frac{-D_{i_k,0}}{D_{i_k,j}},$$

and we choose our pivot element to be one that minimizes this ratio. If multiple entries minimize the ratio, then we utilize *Bland's Rule*, which instructs us to choose the entry in the row corresponding to the smallest index (obeying this rule is important, as it prevents the possibility of the algorithm cycling back on itself infinitely). The index corresponding to the pivot row is designated as the *leaving index*, since after the full pivot operation, it will leave the basis and become a independent variable.

In our example, we see that all entries in the pivot column (ignoring the entry in the first row, of course) are negative, and hence they are all potential choices for the pivot element. We then calculate the ratios, and obtain

$$\frac{-2}{-1} = 2, \quad \frac{-5}{-3} = 1.66\ldots, \quad \frac{-7}{-4} = 1.75.$$

We see that the entry in the third row minimizes these ratios. Hence, the element in the second column (index 1), third row (index 2) is our designated pivot element.

$$
D = \begin{bmatrix}
0 & -3 & -2 & 0 & 0 & 0 \\
2 & -1 & 1 & -1 & 0 & 0 \\
5 & \boxed{-3} & -1 & 0 & -1 & 0 \\
7 & -4 & -3 & 0 & 0 & -1
\end{bmatrix}
$$

Definition 8.12: Bland's Rule

hoose the independent variable with the smallest index that has a negative coefficient in the objective function as the leaving variable. Choose the dependent variable with the smallest index among all the binding dependent variables.

Bland's Rule is important in avoiding cycles when performing pivots. This rule guarantees that a feasible Simplex problem will terminate in a finite number of pivots.

Finally, we perform row operations on our dictionary in the following way: divide the pivot row by the negative value of the pivot entry. Then use the pivot row to zero out all entries in the pivot column above and below the pivot entry. In our example, we first divide the pivot row by -3, and then zero out the two entries above the pivot element and the single entry below it:

$$
\begin{bmatrix}
0 & -3 & -2 & 0 & 0 & 0 \\
2 & -1 & 1 & -1 & 0 & 0 \\
5 & -3 & -1 & 0 & -1 & 0 \\
7 & -4 & -3 & 0 & 0 & -1
\end{bmatrix} \rightarrow
\begin{bmatrix}
0 & -3 & -2 & 0 & 0 & 0 \\
2 & -1 & 1 & -1 & 0 & 0 \\
5/3 & -1 & -1/3 & 0 & -1/3 & 0 \\
7 & -4 & -3 & 0 & 0 & -1
\end{bmatrix} \rightarrow
$$

$$
\begin{bmatrix}
-5 & 0 & -1 & 0 & 1 & 0 \\
2 & -1 & 1 & -1 & 0 & 0 \\
5/3 & -1 & -1/3 & 0 & -1/3 & 0 \\
7 & -4 & -3 & 0 & 0 & -1
\end{bmatrix} \rightarrow
\begin{bmatrix}
-5 & 0 & -1 & 0 & 1 & 0 \\
1/3 & 0 & -4/3 & 1 & -1/3 & 0 \\
5/3 & -1 & -1/3 & 0 & -1/3 & 0 \\
7 & -4 & -3 & 0 & 0 & -1
\end{bmatrix} \rightarrow
$$

$$
\begin{bmatrix}
-5 & 0 & -1 & 0 & 1 & 0 \\
1/3 & 0 & 4/3 & -1 & 1/3 & 0 \\
5/3 & -1 & -1/3 & 0 & -1/3 & 0 \\
1/3 & 0 & -5/3 & 0 & 4/3 & -1
\end{bmatrix}.
$$

The result of these row operations is our updated dictionary, and the pivot operation is complete.

Problem 8.13: Pivoting

dd a method to your solver that checks for unboundedness and performs a single pivot operation from start to completion. If the problem is unbounded, raise a `ValueError`.

8.1.2. Termination and Reading the Dictionary

Up to this point, our algorithm accepts a linear program, adds slack variables, and creates the initial dictionary. After carrying out these initial steps, it then performs the pivoting operation iteratively until the optimal point is found. But how do we determine when the optimal point is found? The answer is to look at the top row of the dictionary, which represents the objective function. More specifically, before each pivoting operation, check whether all of the entries in the top row of the dictionary (ignoring the entry in the first column) are nonnegative. If this is the case, then we have found an optimal solution, and so we terminate the algorithm.

The final step is to report the solution. The ending state of the dictionary and index list tells us everything we need to know. The minimal value attained by the objective function is found in the upper leftmost entry of the dictionary. The dependent variables all have the value 0 in the objective function or first row of our dictionary array. The independent variables have values given by the first column of the dictionary. Specifically, the independent variable whose index is located at the ith entry of the index list has the value $T_{i+1,0}$.

In our example, suppose that our algorithm terminates with the dictionary and index list in the following state:

$$D = \begin{bmatrix} -5.2 & 0 & 0 & 0 & 0.2 & 0.6 \\ 0.6 & 0 & 0 & -1 & 1.4 & -0.8 \\ 1.6 & -1 & 0 & 0 & -0.6 & 0.2 \\ 0.2 & 0 & -1 & 0 & 0.8 & -0.6 \end{bmatrix}$$

Then the minimal value of the objective function is -5.2. The independent variables have indices $4, 5$ and have the value 0. The dependent variables have indices $3, 1$, and 2, and have values $.6, 1.6$, and $.2$, respectively. In the notation of the original problem statement, the solution is given by

$$x_0 = 1.6$$
$$x_1 = .2.$$

Problem 8.14: SimplexSolver.solve()

rite an additional method in your solver called `solve()` that obtains the optimal solution, then returns the minimal value, the dependent variables, and the independent variables. The dependent and independent variables should be represented as two dictionaries that map the index of the variable to its corresponding value.

For our example, we would return the tuple

`(-5.2, {0: 1.6, 1: .2, 2: .6}, {3: 0, 4: 0}).`

At this point, you should have a Simplex solver that is ready to use. The following code demonstrates how your solver is expected to behave:

```
>>> import SimplexSolver

# Initialize objective function and constraints.
>>> c = np.array([-3., -2.])
>>> b = np.array([2., 5, 7])
>>> A = np.array([[1., -1], [3, 1], [4, 3]])

# Instantiate the simplex solver, then solve the problem.
>>> solver = SimplexSolver(c, A, b)
>>> sol = solver.solve()
>>> print(sol)
(-5.2,
 {0: 1.6, 1: 0.2, 2: 0.6},
 {3: 0, 4: 0})
```

If the linear program were infeasible at the origin or unbounded, we would expect the solver to alert the user by raising an error.

Note that this simplex solver is *not* fully operational. It can't handle the case of infeasibility at the origin. This can be fixed by adding methods to your class that solve the *auxiliary problem*, that of finding an initial feasible dictionary when the problem is not feasible at the origin. Solving the auxiliary problem involves pivoting operations identical to those you have already implemented, so adding this functionality is not overly difficult.

8.1.3. Exercises

EXERCISE 1.0 (LEARN LT$_E$X) Learn to use lt$_E$X for writing all of your homework solutions. Personally, I use MiKTEX, which is an implementation of ETEX for Windows. Specifically, within MiKTEX I am using pdfteTEX (it only matters for certain things like including graphics and also pdf into a document). I find it convenient to use the editor WinEdt, which is very LATEX friendly. A good book on ETTX is

In A.1 there is a template to get started. Also, there are plenty of tutorials and beginner's guides on the web.

EXERCISE 1.1 (CONVERT TO STANDARD FORM)   Give an original example (i.e., with actual numbers) to demonstrate that you know how to transform a general linear-optimization problem to one in standard form.

EXERCISE 1.2 (WEAK DUALITY EXAMPLE)   Give an original example to demonstrate the Weak Duality Theorem.

EXERCISE 1.3 (CONVERT TO $\leq$ FORM)   Describe a general recipe for transforming an arbitrary linear-optimization problem into one in which all of the linear constraints are of $\leq$ type.

EXERCISE 1.4 ($m+1$ INEQUALITIES)   Prove that the system of m equations in n variables $Ax = b$ is equivalent to the system $Ax \leq b$ augmented by only one additional linear inequality - that is, a total of only $m+1$ inequalities.

EXERCISE 1.5 (WEAK DUALITY FOR ANOTHER FORM)   Give and prove a Weak Duality Theorem for

$$
\begin{aligned}
\max \quad & c'x \\
& Ax \leq b; \\
& x \geq 0.
\end{aligned}
$$

HINT: Convert (P') to a standard-form problem, and then apply the ordinary Weak Duality Theorem for standard-form problems.

EXERCISE 1.6 (WEAK DUALITY FOR A COMPLICATED FORM)   Give and prove a Weak Duality Theorem for

$$
\begin{aligned}
\min \quad & c'x + f'w \\
& Ax + Bw \leq b; \\
& Dx = g; \\
& x \geq 0 \quad w \leq 0
\end{aligned}
$$

HINT: Convert (P') to a standard-form problem, and then apply the ordinary Weak Duality Theorem for standard-form problems.

EXERCISE 1.7 (WEAK DUALITY FOR A COMPLICATED FORM - WITH MATLAB)   The MATLAB code below makes and solves an instance of (P′) from Exercise 1.6. Study the code to see how it is works. Now, extend the code to solve the dual of (P′). Also, after converting (P′) to standard form (as indicated in the HINT for Exercise 1.6), use MATLAB to solve that problem and its dual. Make sure that you get the same optimal value for all of these problems.

8.1.4. 2.5 Exercises

EXERCISE 2.1 (DUAL IN AMPL)   Without changing the file production. dat, use AMPL to solve the dual of the Production Problem example, as described in Section 2.1. You will need to modify production.mod and production.run.

EXERCISE 2.2 (SPARSE SOLUTION FOR LINEAR EQUATIONS WITH AMPL)   In some application areas, it is interesting to find a "sparse solution" - that is, one with few non-zeros - to a system of equations $Ax = b$. It is well known that a 1-norm minimizing solution is a good heuristic for finding a sparse solution. Using AMPL, try this idea out on several large examples, and report on your results.

HINT: To get an interesting example, try generating a random $m \times n$ matrix A of zeros and ones, perhaps $m = 50$ equations and $n = 500$ variables, maybe with probability $1/2$ of an entry being equal to one. Then choose maybe $m/2$ columns from A and add them up to get b. In this way, you will know that there is a solution with only $m/2$ non-zeros (which is already pretty sparse). Your 1-norm minimizing solution might in fact recover this solution ($\odot$), or it may be sparser ($\odot\odot$), or perhaps less sparse ($\odot$).

EXERCISE 2.3 (BLOODY AMPL)   A transportation problem is a special kind of (single-commodity min-cost) networkflow problem. There are certain nodes v called supply nodes which have net supply $b_v > 0$. The other nodes v are called demand nodes, and they have net supply $b_v < 0$. There are no nodes with $b_v = 0$, and all arcs point from supply nodes to demand nodes.

A simplified example is for matching available supply and demand of blood, in types A, B, AB and O. Suppose that we have s_v units of blood available, in types $v \in \{A, B, AB, O\}$. Also, we have requirements d_v by patients of different types $v \in \{A, B, AB, O\}$. It is very important to understand that a patient of a certain type can accept blood not just from their own type. Do some research to find out the compatible blood types for a patient; don't make a mistake - lives depend on this! In this spirit, if your model misallocates any blood in an incompatible fashion, you will receive a grade of F on this problem.

Describe a linear-optimization problem that satisfies all of the patient demand with compatible blood. You will find that type O is the most versatile blood, then both A and B, followed by AB. Factor in this point when you formulate your objective function, with the idea of having the left-over supply of blood being as versatile as possible.

Using AMPL, set up and solve an example of a blood-distribution problem.

EXERCISE 2.4 (MIX IT UP)   "I might sing a gospel song in Arabic or do something in Hebrew. I want to mix it up and do it differently than one might imagine." - Stevie Wonder

We are given a set of ingredients $1, 2, \ldots, m$ with availabilities b_i and per unit costs c_i. We are given a set of products $j, 2, \ldots, m$ with minimum production requirements d_j and per unit revenues e_j. It is required that product j have at least a fraction of l_{ij} of ingredient i and at most a fraction of u_{ij} of ingredient i. The goal is to devise a plan to maximize net profit.

Formulate, mathematically, as a linear-optimization problem. Then, model with AMPL, make up some data, try some computations, and report on your results. Exercise 2.5 (Task scheduling)

We are given a set of tasks, numbered $1, 2, \ldots, n$ that should be completed in the minimum amount of time. For convenience, task 0 is a "start task" and task $n + 1$ is an "end task". Each task, except for the start and end task, has a known duration d_i. For convenience, let $d_0 := 0$. There are precedences between tasks. Specifically, Ψ_i is the set of tasks that must be completed before task i can be started. Let $t_0 := 0$, and for all other tasks i, let t_i be a decision variable representing its start time.

Formulate the problem, mathematically, as a linear-optimization problem. The objective should be to minimize the start time t_{n+1} of the end task. Then, model the problem with AMPL, make up some data, try some computations, and report on your results.

EXERCISE 2.6 (INVESTING WISELY)   Almost certainly, Albert Einstein did not say that "compound interest is the most powerful force in the universe."

A company wants to maximize their cash holdings after T time periods. They have an external inflow of p_t dollars at the start of time period t, for $t = 1, 2, \ldots, T$. At the start of each time period, available cash can be allocated to any of K different investment vehicles (in any available non-negative amounts). Money allocated to investmentvehicle k at the start of period t must be held in that investment k for all remaining time periods, and it generates income $v_{t,t}^k, v_{t,t+1}^k, \ldots, v_{tT}^k$, per dollar invested. It should be assumed that money obtained from cashing out the investment at the end of the planning horizon (that is, at the end of period T) is part of $v_{t,T}^k$. Note that at the start of time period t, the cash available is the external inflow of p_t, plus cash accumulated from all investment vehicles in prior periods that was not reinvested. Finally, assume that cash held over for one time period earns interest of q percent.

Formulate the problem, mathematically, as a linear-optimization problem. Then, model the problem with AMPL, make up some data, try some computations, and report on your results.

8.2 Finding Feasible Basis

Finding an Initial BFS When a basic feasible solution is not apparent, we an produce one using *artificial variables*. This *artificial* basis is undesirable from the perspective of the original problem, we do not want the artificial variables in our solution, so we penalize them in the objective function, and allow the simplex algorithm to drive them to zero (if possible) and out of the basis. There are two such methods, the **Big M method** and the **Two-phase method**, which we illustrate below:

Solve the following LP using the Big M Method and the simplex algorithm:

$$max \ z = 9x_1 + 6x_2$$
$$s.t. \ 3x_1 + 3x_2 \leq 9$$
$$2x_1 - 2x_2 \geq 3$$
$$2x_1 + 2x_2 \geq 4$$
$$x_1, x_2 \geq 0.$$

Here is the LP is transformed into standard form by using slack variables x_3, x_4, and x_5, with the required artificial variables x_6 and x_7, which allow us to easily find an initial basic feasible solution (to the artificial problem).

$$max \ z_a = 9x_1 + 6x_2 - Mx_6 - Mx_7$$
$$s.t. \ 3x_1 + 3x_2 + x_3 = 9$$
$$2x_1 - 2x_2 - x_4 + x_6 = 3$$
$$2x_1 + 2x_2 - x_5 + x_7 = 4$$
$$x_i \geq 0, \ i = 1, \cdots, 7.$$

z	x_1	x_2	x_3	x_4	x_5	x_6	x_7	RHS	ratio
1	-9	-6	0	0	0	M	M	0	
0	3	3	1	0	0	0	0	9	
0	2	-2	0	-1	0	1	0	3	
0	2	2	0	0	-1	0	1	4	

This tableau is not in the correct form, it does not represent a basis, the columns for the artificial variables need to be adjusted.

z	x_1	x_2	x_3	x_4	x_5	x_6	x_7	RHS	ratio
1	-9 - 4M	-6	0	M	M	0	0	-7M	
0	3	3	1	0	0	0	0	9	3
0	2	-2	0	-1	0	1	0	3	3/2
0	2	2	0	0	-1	0	1	4	2

The current solution is not optimal, so x_1 enters the basis, and by the ratio test, x_6 (an artificial variable) leaves the basis.

z	x_1	x_2	x_3	x_4	x_5	x_6	x_7	RHS	ratio
1	0	-15 -4M	0	-9/2 -M	M	9/2 + 2M	0	27/2 -M	
0	0	6	1	3/2	0	-3/2	0	3/2	3/4
0	1	-1	0	-1/2	0	1/2	0	3/2	-
0	0	4	0	1	-1	-1	1	1	1/4

The current solution is not optimal, so x_2 enters the basis, and by the ratio test, x_7 (an artificial variable) leaves the basis.

z	x_1	x_2	x_3	x_4	x_5	x_6	x_7	RHS	ratio
1	0	0	0	-3/4	-15/4	-	-	17 1/4	
0	0	0	1	0	3/2	0	-3/2	3	-
0	1	0	0	-1/4	-1/4	1/2	1/4	7/4	-
0	0	1	0	1/4	-1/4	-1/4	1/4	1/4	1

The current solution is not optimal, so x_4 enters the basis, and by the ratio test, x_2 leaves the basis.

z	x_1	x_2	x_3	x_4	x_5	x_6	x_7	RHS	ratio
1	0	3	0	0	-9/2	-	-	18	
0	0	0	1	0	3/2	0	-3/2	3	-
0	1	1	0	0	-1/2	0	1/2	2	-
0	0	4	0	1	-1	-1	1	1	1

The current solution is not optimal, so x_5 enters the basis, and by the ratio test, x_3 leaves the basis.

z	x_1	x_2	x_3	x_4	x_5	x_6	x_7	RHS	ratio
1	0	3	3	0	0	-	-	27	
0	0	0	2/3	0	1	0	-1	2	
0	1	1	1/3	0	0	0	0	3	
0	0	4	2/3	1	0	-1	0	3	

The current solution is optimal!

Solve the following LP using the Two-phase Method and Simplex Algorithm.

$$\max \ z = 2x_1 + 3x_2$$
$$s.t. \ 3x_1 + 3x_2 \geq 6$$
$$2x_1 - 2x_2 \leq 2$$
$$-3x_1 + 3x_2 \leq 6$$
$$x_1, x_2 \geq 0.$$

Here is first phase LP (in standard form), where x_3, x_4, and x_5 are slack variables, and x_6 is an artificial variable.

$$\min \ z_a = x_6$$
$$s.t. \ 3x_1 + 3x_2 - x_3 + x_6 = 6$$
$$2x_1 - 2x_2 + x_4 = 2$$
$$-3x_1 + 3x_2 + x_5 = 6$$
$$x_i \geq 0, \ i = 1, \cdots, 6.$$

Next, we put the LP into a tableau, which, still is not in the right form for our basic variables (x_6, x_4, and x_5).

z	x_1	x_2	x_3	x_4	x_5	x_6	RHS	ratio
1	0	0	0	0	0	-1	0	
0	3	3	-1	0	0	1	6	
0	2	-2	0	1	0	0	2	
0	-3	3	0	0	1	0	6	

To remedy this, we use row operation to modify the row 0 coefficients, yielding the following:

z	x_1	x_2	x_3	x_4	x_5	x_6	RHS	ratio
1	3	3	-1	0	0	0	6	
0	3	3	-1	0	0	1	6	2
0	2	-2	0	1	0	0	2	-
0	-3	3	0	0	1	0	6	2

The current solution is not optimal, either x_1 or x_2 can enter the basis, let's choose x_2. Then by the ratio test, either x_6 (an artificial variable) or x_5 (a slack variable) can leaves the basis. Let's choose x_6.

z	x_1	x_2	x_3	x_4	x_5	x_6	RHS	ratio
1	0	0	0	0	0	-1	0	
0	1	1	-1/3	0	0	1/3	2	
0	4	0	-2/3	1	0	2/3	6	
0	-6	0	1	0	1	-1	0	

The current solution is optimal, so we end the first phase with a basic feasible solution to the original problem, with x_2, x_4, and x_5 as the basic variables. Now we provide a new row zero that corresponds to the original problem.

z	x_1	x_2	x_3	x_4	x_5	x_6	RHS	ratio
1	1	0	-1	0	0	0	6	
0	1	1	-1/3	0	0	1/3	2	
0	4	0	-2/3	1	0	2/3	6	
0	-6	0	1	0	1	-1	0	

z	x_1	x_2	x_3	x_4	x_5	x_6	RHS	ratio
1	-5	0	0	0	1	-1	6	
0	-1	1	0	0	1/3	0	2	
0	0	0	0	1	2/3	0	6	
0	-6	0	1	0	1	-1	0	

From this tableau we can see that the LP is unbounded and an extreme point is [0, 2, 0, 6,0] and an extreme direction is [1, 1, 6, 0, 0].

Degeneracy and the Simplex Algorithm

Degeneracy must be considered in the simplex algorithm, as it causes some trouble. For instance, it might mislead us into thinking there are multiple optimal solutions, or provide faulty insight. Further, the algorithm as described can *cycle*, that is, remain on a degenerate extreme point repeatedly cycling through a subset of bases that represent that point, never leaving.

min	z	x_1	x_2	x_3	x_4	x_5	x_6	x_7	rhs
	1	0	0	0	3/4	-20	1/2	-6	0
	0	1	0	0	1/4	-8	-1	9	0
	0	0	1	0	1/2	-12	-1/2	3	0
	0	0	0	1	0	0	1	0	1

Solve the following LP using the Simplex Algorithm:

$$\max \quad z = 40x_1 + 30x_2$$
$$s.t. \quad 6x_1 + 4x_2 \leq 40$$
$$4x_1 + 2x_2 \leq 20$$
$$x_1, x_2 \geq 0.$$

By adding slack variables, we have the following tableau. Luckily, this tableau represents a basis, where

z	x_1	x_2	s_1	s_2	RHS
1	-40	-30	0	0	0
0	6	4	1	0	40
0	4	2	0	1	20

BV=$\{s_1, s_2\}$, but by inspecting the row 0 (objective function row) coefficients, we can see that this is not optimal. By Dantzig's Rule, we enter x_1 into the basis, and by the ratio test we see that s_2 leaves the basis. By performing elementary row operations, we obtain the following tableau for the new basis BV=$\{s_1, x_1\}$.

z	x_1	x_2	s_1	s_2	RHS
1	0	-10	0	10	200
0	0	1	1	-3/2	10
0	1	1/2	0	1/4	5

This tableau is not optimal, entering x_2 into the basis can improve the objective function value. The basic variables s_1 and x_1 tie in the ration test. If we have x_1 leave the basis, we get the following tableau (BV=$\{s_1, x_2\}$).

z	x_1	x_2	s_1	s_2	RHS
1	20	0	0	15	300
0	-2	0	1	-2	0
0	2	1	0	1/2	10

This is an optimal tableau, with an objective function value of 300, If instead of x_1 leaving the basis, suppose s_1 left, this would lead to the following tableau (BV=$\{x_2, x_1\}$).

z	x_1	x_2	s_1	s_2	RHS
1	0	0	10	-5	300
0	0	1	1	-3/2	10
0	1	0	-1/2	1	0

This tableau does not look optimal, yet the objective function value is the same as the optimal solution's. This occurs because the optimal extreme point is a degenerate.

9. Duality

Before I prove the stronger duality theorem, let me first provide some intuition about where this duality thing comes from in the first place. [6] Consider the following linear programming problem:

$$\text{maximize } 4x_1 + x_2 + 3x_3$$
$$\text{subject to} \quad x_1 + 4x_2 \leq 2$$
$$3x_1 - x_2 + x_3 \qquad\qquad\qquad \leq 4$$
$$x_1, x_2, x_3 \qquad\qquad\qquad \geq 0$$

Let σ^* denote the optimum objective value for this LP. The feasible solution $x = (1,0,0)$ gives us a lower bound $\sigma^* \geq 4$. A different feasible solution $x = (0,0,3)$ gives us a better lower bound $\sigma^* \geq 9$. We could play this game all day, finding different feasible solutions and getting ever larger lower bounds. How do we know when we're done? Is there a way to prove an upper bound on σ^* ?

In fact, there is. Let's multiply each of the constraints in our LP by a new non-negative scalar value y_i :

$$\text{maximize } 4x_1 + x_2 + 3x_3$$
$$\text{subject to } y_1 (x_1 + 4x_2) \quad \leq 2y_1$$
$$y_2 (3x_1 - x_2 + x_3) \leq 4y_2$$
$$x_1, x_2, x_3 \geq 0$$

Because each y_i is non-negative, we do not reverse any of the inequalities. Any feasible solution (x_1, x_2, x_3) must satisfy both of these inequalities, so it must also satisfy their sum:

$$(y_1 + 3y_2)x_1 + (4y_1 - y_2)x_2 + y_2x_3 \leq 2y_1 + 4y_2.$$

Now suppose that each y_i is larger than the i th coefficient of the objective function:

$$y_1 + 3y_2 \geq 4, \quad 4y_1 - y_2 \geq 1, \quad y_2 \geq 3.$$

This assumption lets us derive an upper bound on the objective value of any feasible solution:

$$4x_1 + x_2 + 3x_3 \leq (y_1 + 3y_2)x_1 + (4y_1 - y_2)x_2 + y_2x_3 \leq 2y_1 + 4y_2.$$

In particular, by plugging in the optimal solution (x_1^*, x_2^*, x_3^*) for the original LP, we obtain the following upper bound on σ^* :
$$\sigma^* = 4x_1^* + x_2^* + 3x_3^* \leq 2y_1 + 4y_2.$$

Now it's natural to ask how tight we can make this upper bound. How small can we make the expression $2y_1 + 4y_2$ without violating any of the inequalities we used to prove the upper bound? This is just another

linear programming problem.

$$
\begin{aligned}
\text{minimize} \quad & 2y_1 + 4y_2 \\
\text{subject to} \quad & y_1 + 3y_2 \geq 4 \\
& 4y_1 - y_2 \geq 1 \\
& y_2 \geq 3 \\
& y_1, y_2 \geq 0
\end{aligned}
$$

"This example is taken from Robert Vanderbei's excellent textbook Linear Programming: Foundations and Extensions [Springer, 2001], but the idea appears earlier in Jens Clausen's 1997 paper 'Teaching Duality in Linear Programming: The Multiplier Approach'.

`https://www.cs.purdue.edu/homes/egrigore/580FT15/26-lp-jefferickson.pdf`

In which we introduce the theory of duality in linear programming.

9.1 The Dual of Linear Program

Suppose that we have the following linear program in maximization standard form:

$$
\begin{aligned}
\text{maximize} \quad & x_1 + 2x_2 + x_3 + x_4 \\
\text{subject to} \quad & \\
& x_1 + 2x_2 + x_3 \leq 2 \\
& x_2 + x_4 \leq 1 \\
& x_1 + 2x_3 \leq 1 \\
& x_1 \geq 0 \\
& x_2 \geq 0 \\
& x_3 \geq 0
\end{aligned}
$$

and that an LP-solver has found for us the solution $x_1 := 1, x_2 := \frac{1}{2}, x_3 := 0, x_4 := \frac{1}{2}$ of cost 2.5. How can we convince ourselves, or another user, that the solution is indeed optimal, without having to trace the steps of the computation of the algorithm?

Observe that if we have two valid inequalities

$$a \leq b \text{ and } c \leq d$$

then we can deduce that the inequality

$$a + c \leq b + d$$

(derived by "summing the left hand sides and the right hand sides" of our original inequalities) is also true. In fact, we can also scale the inequalities by a positive multiplicative factor before adding them up, so for every non-negative values $y_1, y_2 \geq 0$ we also have

$$y_1 a + y_2 c \leq y_1 b + y_2 d$$

Going back to our linear program (1), we see that if we scale the first inequality by $\frac{1}{2}$, add the second inequality, and then add the third inequality scaled by $\frac{1}{2}$, we get that, for every (x_1, x_2, x_3, x_4) that is feasible for (1),

$$x_1 + 2x_2 + 1.5x_3 + x_4 \leq 2.5$$

And so, for every feasible (x_1, x_2, x_3, x_4), its cost is

$$x_1 + 2x_2 + x_3 + x_4 \leq x_1 + 2x_2 + 1.5x_3 + x_4 \leq 2.5$$

meaning that a solution of cost 2.5 is indeed optimal.

In general, how do we find a good choice of scaling factors for the inequalities, and what kind of upper bounds can we prove to the optimum?

Suppose that we have a maximization linear program in standard form.

$$
\begin{aligned}
\text{maximize} \quad & c_1 x_1 + \ldots c_n x_n \\
\text{subject to} \quad & \\
& a_{1,1} x_1 + \ldots + a_{1,n} x_n \leq b_1 \\
& \vdots \\
& a_{m,1} x_1 + \ldots + a_{m,n} x_n \leq b_m \\
& x_1 \geq 0 \\
& \vdots \\
& x_n \geq 0
\end{aligned}
$$

For every choice of non-negative scaling factors $y_1, \ldots, y_m$, we can derive the inequality

$$
\begin{aligned}
y_1 \cdot (a_{1,1} x_1 + \ldots + a_{1,n} x_n) \\
+ \cdots \\
+ y_n \cdot (a_{m,1} x_1 + \ldots + a_{m,n} x_n) \\
\leq y_1 b_1 + \cdots y_m b_m
\end{aligned}
$$

which is true for every feasible solution $(x_1, \ldots, x_n)$ to the linear program (2). We can rewrite the inequality as

$$
\begin{aligned}
(a_{1,1} y_1 + \cdots a_{m,1} y_m) \cdot x_1 \\
+ \cdots
\end{aligned}
$$

$$+ (a_{1,n}y_1 \cdots a_{m,n}y_m) \cdot x_n$$
$$\leq y_1 b_1 + \cdots y_m b_m$$

So we get that a certain linear function of the x_i is always at most a certain value, for every feasible $(x_1, \ldots, x_n)$. The trick is now to choose the y_i so that the linear function of the x_i for which we get an upper bound is, in turn, an upper bound to the cost function of $(x_1, \ldots, x_n)$. We can achieve this if we choose the y_i such that

$$c_1 \leq a_{1,1}y_1 + \cdots a_{m,1}y_m$$
$$\vdots$$
$$c_n \leq a_{1,n}y_1 \cdots a_{m,n}y_m$$

Now we see that for every non-negative $(y_1, \ldots, y_m)$ that satisfies (3), and for every $(x_1, \ldots, x_n)$ that is feasible for (2),

$$c_1 x_1 + \ldots c_n x_n$$
$$\leq (a_{1,1}y_1 + \cdots a_{m,1}y_m) \cdot x_1$$
$$+ \cdots$$
$$+ (a_{1,n}y_1 \cdots a_{m,n}y_m) \cdot x_n$$
$$\leq y_1 b_1 + \cdots y_m b_m$$

Clearly, we want to find the non-negative values $y_1, \ldots, y_m$ such that the above upper bound is as strong as possible, that is we want to

$$
\begin{aligned}
\text{minimize} \quad & b_1 y_1 + \cdots b_m y_m \\
\text{subject to} \quad & \\
& a_{1,1}y_1 + \ldots + a_{m,1}y_m \geq c_1 \\
& \vdots \\
& a_{n,1}y_1 + \ldots + a_{m,n}y_m \geq c_n \\
& y_1 \geq 0 \\
& \vdots \\
& y_m \geq 0
\end{aligned}
$$

So we find out that if we want to find the scaling factors that give us the best possible upper bound to the optimum of a linear program in standard maximization form, we end up with a new linear program, in standard minimization form. Definition 1 If

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

is a linear program in maximization standard form, then its dual is the minimization linear program

$$\text{minimize} \quad \mathbf{b}^T \mathbf{y}$$
$$\text{subject to}$$
$$A^T \mathbf{y} \geq \mathbf{c}$$
$$\mathbf{y} \geq 0$$

So if we have a linear program in maximization linear form, which we are going to call the primal linear program, its dual is formed by having one variable for each constraint of the primal (not counting the non-negativity constraints of the primal variables), and having one constraint for each variable of the primal (plus the nonnegative constraints of the dual variables); we change maximization to minimization, we switch the roles of the coefficients of the objective function and of the right-hand sides of the inequalities, and we take the transpose of the matrix of coefficients of the left-hand side of the inequalities.

The optimum of the dual is now an upper bound to the optimum of the primal. How do we do the same thing but starting from a minimization linear program? We can rewrite

$$\underset{l}{\text{minimize}} \quad \mathbf{c}^T \mathbf{y}$$
$$\text{subject to}$$
$$A\mathbf{y} \geq \mathbf{b}$$
$$\mathbf{y} \geq 0$$

in an equivalent way as

$$\underset{\text{maximize}}{\text{mubject to}} \quad -\mathbf{c}^T \mathbf{y}$$
$$-A\mathbf{y} \leq -\mathbf{b}$$
$$\mathbf{y} \geq 0$$

If we compute the dual of the above program we get

$$\underset{\text{minimize}}{\text{mubject to}} \quad -\mathbf{b}^T \mathbf{z}$$
$$-A^T \mathbf{z} \geq -\mathbf{c}$$
$$\mathbf{z} \geq 0$$

that is,

$$\text{maximize} \quad \mathbf{b}^T \mathbf{z}$$
$$\text{subject to}$$
$$A^T \mathbf{z} \leq \mathbf{c}$$
$$\mathbf{y} \geq 0$$

So we can form the dual of a linear program in minimization normal form in the same way in which we formed the dual in the maximization case:

- switch the type of optimization,

- introduce as many dual variables as the number of primal constraints (not counting the non-negativity constraints),

- define as many dual constraints (not counting the non-negativity constraints) as the number of primal variables.

- take the transpose of the matrix of coefficients of the left-hand side of the inequality,

- switch the roles of the vector of coefficients in the objective function and the vector of right-hand sides in the inequalities.

Note that:

Fact 2 The dual of the dual of a linear program is the linear program itself.

We have already proved the following:

Fact 3 If the primal (in maximization standard form) and the dual (in minimization standard form) are both feasible, then

$$\text{opt(primal)} \leq \text{opt(dual)}$$

Which we can generalize a little

Theorem 4 (Weak Duality Theorem) If LP_1 is a linear program in maximization standard form, LP_2 is a linear program in minimization standard form, and LP_1 and LP_2 are duals of each other then:

- If LP_1 is unbounded, then LP_2 is infeasible; - If LP_2 is unbounded, then LP_1 is infeasible;

- If LP_1 and LP_2 are both feasible and bounded, then

$$\text{opt}(LP_1) \leq \text{opt}(LP_2)$$

ProOF: We have proved the third statement already. Now observe that the third statement is also saying that if LP_1 and LP_2 are both feasible, then they have to both be bounded, because every feasible solution to LP_2 gives a finite upper bound to the optimum of LP_1 (which then cannot be $+\infty$) and every feasible solution to LP_1 gives a finite lower bound to the optimum of LP_2 (which then cannot be $-\infty$).

What is surprising is that, for bounded and feasible linear programs, there is always a dual solution that certifies the exact value of the optimum.

Theorem 5 (Strong Duality) If either LP_1 or LP_2 is feasible and bounded, then so is the other, and

$$\text{opt}(LP_1) = \text{opt}(LP_2)$$

To summarize, the following cases can arise:

- If one of LP_1 or LP_2 is feasible and bounded, then so is the other;

- If one of LP_1 or LP_2 is unbounded, then the other is infeasible;

- If one of LP_1 or LP_2 is infeasible, then the other cannot be feasible and bounded, that is, the other is going to be either infeasible or unbounded. Either case can happen.

9.2 Linear programming duality

Consider the following problem:

$$
\begin{aligned}
\min \quad & \mathbf{c}^{\mathsf{T}}\mathbf{x} \\
\text{s.t.} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b}.
\end{aligned}
\tag{9.1}
$$

In the remark at the end of Chapter **??**, we saw that if (9.1) has an optimal solution, then there exists $\mathbf{y}^* \in \mathbb{R}^m$ such that $\mathbf{y}^* \geq 0$, $\mathbf{y}^{*\mathsf{T}}\mathbf{A} = \mathbf{c}^{\mathsf{T}}$, and $\mathbf{y}^{*\mathsf{T}}\mathbf{b} = \gamma$ where γ denotes the optimal value of (9.1).

Take any $\mathbf{y} \in \mathbb{R}^m$ satisfying $\mathbf{y} \geq 0$ and $\mathbf{y}^{\mathsf{T}}\mathbf{A} = \mathbf{c}^{\mathsf{T}}$. Then we can infer from $\mathbf{A}\mathbf{x} \geq \mathbf{b}$ the inequality $\mathbf{y}^{\mathsf{T}}\mathbf{A}\mathbf{x} \geq \mathbf{y}^{\mathsf{T}}\mathbf{b}$, or more simply, $\mathbf{c}^{\mathsf{T}}\mathbf{x} \geq \mathbf{y}^{\mathsf{T}}\mathbf{b}$. Thus, for any such $\mathbf{y}$, $\mathbf{y}^{\mathsf{T}}\mathbf{b}$ gives a lower bound for the objective function value of any feasible solution to (9.1). Since γ is the optimal value of (P), we must have $\gamma \geq \mathbf{y}^{\mathsf{T}}\mathbf{b}$.

As $\mathbf{y}^{*\mathsf{T}}\mathbf{b} = \gamma$, we see that γ is the optimal value of

$$
\begin{aligned}
\max \quad & \mathbf{y}^{\mathsf{T}}\mathbf{b} \\
\text{s.t.} \quad & \mathbf{y}^{\mathsf{T}}\mathbf{A} = \mathbf{c}^{\mathsf{T}} \\
& \mathbf{y} \geq 0.
\end{aligned}
\tag{9.2}
$$

Note that (9.2) is a linear programming problem! We call it the **dual problem** of the **primal problem** (9.1). We say that the dual variable y_i is **associated** with the constraint $\mathbf{a}^{(i)\mathsf{T}}\mathbf{x} \geq b_i$ where $\mathbf{a}^{(i)\mathsf{T}}$ denotes the ith row of $\mathbf{A}$.

In other words, we define the dual problem of (9.1) to be the linear programming problem (9.2). In the discussion above, we saw that if the primal problem has an optimal solution, then so does the dual problem and the optimal values of the two problems are equal. Thus, we have the following result:

> **Theorem 9.1: strong-duality-special**
>
> *Suppose that (9.1) has an optimal solution. Then (9.2) also has an optimal solution and the optimal values of the two problems are equal.*

At first glance, requiring all the constraints to be $\geq$-inequalities as in (9.1) before forming the dual problem seems a bit restrictive. We now see how the dual problem of a primal problem in general form can be defined. We first make two observations that motivate the definition.

Observation 1

Suppose that our primal problem contains a mixture of all types of linear constraints:

$$\begin{aligned} \min \quad & \mathbf{c}^{\mathsf{T}}\mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b} \\ & \mathbf{A}'\mathbf{x} \leq \mathbf{b}' \\ & \mathbf{A}''\mathbf{x} = \mathbf{b}'' \end{aligned} \tag{9.3}$$

where $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^{m}$, $\mathbf{A}' \in \mathbb{R}^{m' \times n}$, $\mathbf{b}' \in \mathbb{R}^{m'}$, $\mathbf{A}'' \in \mathbb{R}^{m'' \times n}$, and $\mathbf{b}'' \in \mathbb{R}^{m''}$.

We can of course convert this into an equivalent problem in the form of (9.1) and form its dual. However, if we take the point of view that the function of the dual is to infer from the constraints of (9.3) an inequality of the form $\mathbf{c}^{\mathsf{T}}\mathbf{x} \geq \gamma$ with γ as large as possible by taking an appropriate linear combination of the constraints, we are effectively looking for $\mathbf{y} \in \mathbb{R}^{m}$, $\mathbf{y} \geq 0$, $\mathbf{y}' \in \mathbb{R}^{m'}$, $\mathbf{y}' \leq 0$, and $\mathbf{y}'' \in \mathbb{R}^{m''}$, such that

$$\mathbf{y}^{\mathsf{T}}\mathbf{A} + \mathbf{y}'^{\mathsf{T}}\mathbf{A}' + \mathbf{y}''^{\mathsf{T}}\mathbf{A}'' = \mathbf{c}^{\mathsf{T}}$$

with $\mathbf{y}^{\mathsf{T}}\mathbf{b} + \mathbf{y}'^{\mathsf{T}}\mathbf{b}' + \mathbf{y}''^{\mathsf{T}}\mathbf{b}''$ to be maximized.

(The reason why we need $\mathbf{y}' \leq 0$ is because inferring a $\geq$-inequality from $\mathbf{A}'\mathbf{x} \leq \mathbf{b}'$ requires nonpositive multipliers. There is no restriction on $\mathbf{y}''$ because the constraints $\mathbf{A}''\mathbf{x} = \mathbf{b}''$ are equalities.)

This leads to the dual problem:

$$\begin{aligned} \max \quad & \mathbf{y}^{\mathsf{T}}\mathbf{b} + \mathbf{y}'^{\mathsf{T}}\mathbf{b}' + \mathbf{y}''^{\mathsf{T}}\mathbf{b}'' \\ \text{s.t.} \quad & \mathbf{y}^{\mathsf{T}}\mathbf{A} + \mathbf{y}'^{\mathsf{T}}\mathbf{A}' + \mathbf{y}''^{\mathsf{T}}\mathbf{A}'' = \mathbf{c}^{\mathsf{T}} \\ & \mathbf{y} \geq 0 \\ & \mathbf{y}' \leq 0. \end{aligned} \tag{9.4}$$

In fact, we could have derived this dual by applying the definition of the dual problem to

$$\begin{aligned} \min \quad & \mathbf{c}^{\mathsf{T}}\mathbf{x} \\ \text{s.t.} \quad & \begin{bmatrix} \mathbf{A} \\ -\mathbf{A}' \\ \mathbf{A}'' \\ -\mathbf{A}'' \end{bmatrix} \mathbf{x} \geq \begin{bmatrix} \mathbf{b} \\ -\mathbf{b}' \\ \mathbf{b}'' \\ -\mathbf{b}'' \end{bmatrix}, \end{aligned}$$

which is equivalent to (9.3). The details are left as an exercise.

Observation 2

Consider the primal problem of the following form:

$$\begin{aligned} \min \quad & \mathbf{c}^{\mathsf{T}}\mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b} \\ & x_i \geq 0 \quad i \in P \\ & x_i \leq 0 \quad i \in N \end{aligned} \tag{9.5}$$

where P and N are disjoint subsets of $\{1,\ldots,n\}$. In other words, constraints of the form $x_i \geq 0$ or $x_i \leq 0$ are separated out from the rest of the inequalities.

Forming the dual of (9.5) as defined under Observation 1, we obtain the dual problem

$$
\begin{aligned}
\max \quad & \mathbf{y}^{\mathsf{T}}\mathbf{b} \\
\text{s.t.} \quad & \mathbf{y}^{\mathsf{T}}\mathbf{a}^{(i)} = c_i && i \in \{1,\ldots,n\}\backslash(P \cup N) \\
& \mathbf{y}^{\mathsf{T}}\mathbf{a}^{(i)} + p_i = c_i && i \in P \\
& \mathbf{y}^{\mathsf{T}}\mathbf{a}^{(i)} + q_i = c_i && i \in N \\
& p_i \geq 0 && i \in P \\
& q_i \leq 0 && i \in N
\end{aligned}
\tag{9.6}
$$

where $\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$. Note that this problem is equivalent to the following without the variables p_i, $i \in P$ and q_i, $i \in N$:

$$
\begin{aligned}
\max \quad & \mathbf{y}^{\mathsf{T}}\mathbf{b} \\
\text{s.t.} \quad & \mathbf{y}^{\mathsf{T}}\mathbf{a}^{(i)} = c_i && i \in \{1,\ldots,n\}\backslash(P \cup N) \\
& \mathbf{y}^{\mathsf{T}}\mathbf{a}^{(i)} \leq c_i && i \in P \\
& \mathbf{y}^{\mathsf{T}}\mathbf{a}^{(i)} \geq c_i && i \in N,
\end{aligned}
\tag{9.7}
$$

which can be taken as the dual problem of (9.5) instead of (9.6). The advantage here is that it has fewer variables than (9.6).

Hence, the dual problem of

$$
\begin{aligned}
\min \quad & \mathbf{c}^{\mathsf{T}}\mathbf{x} \\
\text{s.t.} \quad & \mathbf{A}\mathbf{x} \geq \mathbf{b} \\
& \mathbf{x} \geq 0
\end{aligned}
$$

is simply

$$
\begin{aligned}
\max \quad & \mathbf{y}^{\mathsf{T}}\mathbf{b} \\
\text{s.t.} \quad & \mathbf{y}^{\mathsf{T}}\mathbf{A} \leq \mathbf{c}^{\mathsf{T}} \\
& \mathbf{y} \geq 0.
\end{aligned}
$$

As we can see from bove, there is no need to associate dual variables to constraints of the form $x_i \geq 0$ or $x_i \leq 0$ provided we have the appropriate types of constraints in the dual problem. Combining all the observations lead to the definition of the dual problem for a primal problem in general form as discussed next.

9.2.1. The dual problem

Let $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$, $\mathbf{c} \in \mathbb{R}^n$. Let $\mathbf{a}^{(i)^\mathsf{T}}$ denote the ith row of $\mathbf{A}$. Let $\mathbf{A}_j$ denote the jth column of $\mathbf{A}$.

Let (P) denote the minimization problem with variables in the tuple $\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$ given as follows:

- The objective function to be minimized is $\mathbf{c}^\mathsf{T}\mathbf{x}$

- The constraints are

$$\mathbf{a}^{(i)^\mathsf{T}}\mathbf{x} \sqcup_i b_i$$

where $\sqcup_i$ is $\leq$, $\geq$, or $=$ for $i = 1, \ldots, m$.

- For each $j \in \{1, \ldots, n\}$, x_j is constrained to be nonnegative, nonpositive, or free (i.e. not constrained to be nonnegative or nonpositive.)

Then the **dual problem** is defined to be the maximization problem with variables in the tuple $\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$ given as follows:

- The objective function to be maximized is $\mathbf{y}^\mathsf{T}\mathbf{b}$

- For $j = 1, \ldots, n$, the jth constraint is

$$\begin{cases} \mathbf{y}^\mathsf{T}\mathbf{A}_j \leq c_j & \text{if } x_j \text{ is constrained to be nonnegative} \\ \mathbf{y}^\mathsf{T}\mathbf{A}_j \geq c_j & \text{if } x_j \text{ is constrained to be nonpositive} \\ \mathbf{y}^\mathsf{T}\mathbf{A}_j = c_j & \text{if } x_j \text{ is free.} \end{cases}$$

- For each $i \in \{1, \ldots, m\}$, y_i is constrained to be nonnegative if $\sqcup_i$ is $\geq$; y_i is constrained to be nonpositive if $\sqcup_i$ is $\leq$; y_i is free if $\sqcup_i$ is $=$.

The following table can help remember the above.

Primal (min)	Dual (max)
$\geq$ constraint	≥ 0 variable
$\leq$ constraint	≤ 0 variable
$=$ constraint	free variable
≥ 0 variable	$\leq$ constraint
≤ 0 variable	$\geq$ constraint
free variable	$=$ constraint

Below is an example of a primal-dual pair of problems based on the above definition:

Consider the primal problem:

$$
\begin{array}{rrcrcrcl}
\min & x_1 & - & 2x_2 & + & 3x_3 & & \\
\text{s.t.} & -x_1 & & & + & 4x_3 & = & 5 \\
& 2x_1 & + & 3x_2 & - & 5x_3 & \geq & 6 \\
& & & 7x_2 & & & \leq & 8 \\
& x_1 & & & & & \geq & 0 \\
& & & x_2 & & & & \text{free} \\
& & & & & x_3 & \leq & 0.
\end{array}
$$

Here, $\mathbf{A} = \begin{bmatrix} -1 & 0 & 4 \\ 2 & 3 & -5 \\ 0 & 7 & 0 \end{bmatrix}$, $\mathbf{b} = \begin{bmatrix} 5 \\ 6 \\ 8 \end{bmatrix}$, and $\mathbf{c} = \begin{bmatrix} 1 \\ -2 \\ 3 \end{bmatrix}$.

The primal problem has three constraints. So the dual problem has three variables. As the first constraint in the primal is an equation, the corresponding variable in the dual is free. As the second constraint in the primal is a $\geq$-inequality, the corresponding variable in the dual is nonnegative. As the third constraint in the primal is a $\leq$-inequality, the corresponding variable in the dual is nonpositive. Now, the primal problem has three variables. So the dual problem has three constraints. As the first variable in the primal is nonnegative, the corresponding constraint in the dual is a $\leq$-inequality. As the second variable in the primal is free, the corresponding constraint in the dual is an equation. As the third variable in the primal is nonpositive, the corresponding constraint in the dual is a $\geq$-inequality. Hence, the dual problem is:

$$
\begin{array}{rrcrcrcl}
\max & 5y_1 & + & 6y_2 & + & 8y_3 & & \\
\text{s.t.} & -y_1 & + & 2y_2 & & & \leq & 1 \\
& & & 3y_2 & + & 7y_3 & = & -2 \\
& 4y_1 & - & 5y_2 & & & \geq & 3 \\
& y_1 & & & & & & \text{free} \\
& & & y_2 & & & \geq & 0 \\
& & & & & y_3 & \leq & 0.
\end{array}
$$

Remarks. Note that in some books, the primal problem is always a maximization problem. In that case, what is our primal problem is their dual problem and what is our dual problem is their primal problem.

One can now prove a more general version of Theorem 9.2 as stated below. The details are left as an exercise.

> **Theorem 9.2: Duality Theorem for Linear Programming**
>
> *Let (P) and (D) denote a primal-dual pair of linear programming problems. If either (P) or (D) has an optimal solution, then so does the other. Furthermore, the optimal values of the two problems are equal.*

Theorem 9.2.1 is also known informally as **strong duality**.

Exercises

1. Write down the dual problem of

$$
\begin{array}{rrcrcl}
\min & 4x_1 & - & 2x_2 & & \\
\text{s.t.} & x_1 & + & 2x_2 & \geq & 3 \\
& 3x_1 & - & 4x_2 & = & 0 \\
& & & x_2 & \geq & 0.
\end{array}
$$

2. Write down the dual problem of the following:

$$
\begin{array}{rrcrcrcl}
\min & & & 3x_2 & + & x_3 & & \\
\text{s.t.} & x_1 & + & x_2 & + & 2x_3 & = & 1 \\
& x_1 & & & - & 3x_3 & \leq & 0 \\
& x_1 & , & x_2 & , & x_3 & \geq & 0.
\end{array}
$$

3. Write down the dual problem of the following:

$$
\begin{array}{rrcrcrcl}
\min & x_1 & & & - & 9x_3 & & \\
\text{s.t.} & x_1 & - & 3x_2 & + & 2x_3 & = & 1 \\
& x_1 & & & & & \leq & 0 \\
& & & x_2 & & & & \text{free} \\
& & & & & x_3 & \geq & 0.
\end{array}
$$

4. Determine all values c_1, c_2 such that the linear programming problem

$$
\begin{array}{rl}
\min & c_1 x_1 + c_2 x_2 \\
\text{s.t.} & 2x_1 + x_2 \geq 2 \\
& x_1 + 3x_2 \geq 1.
\end{array}
$$

has an optimal solution. Justify your answer

Solutions

1. The dual is

$$
\begin{array}{rrcrcl}
\max & 3y_1 & & & & \\
\text{s.t.} & y_1 & + & 3y_2 & = & 4 \\
& 2y_1 & - & 4y_2 & \leq & -2 \\
& y_1 & & & \geq & 0.
\end{array}
$$

2. The dual is

$$
\begin{array}{rrcrcl}
\max & y_1 & & & & \\
\text{s.t.} & y_1 & + & y_2 & \leq & 0 \\
& y_1 & & & \leq & 3 \\
& 2y_1 & - & 3y_2 & \leq & 1 \\
& y_1 & & & & \text{free} \\
& & & y_2 & \leq & 0.
\end{array}
$$

3. The dual is

$$
\begin{array}{rrcl}
\max & y_1 & & \\
\text{s.t.} & y_1 & \geq & 1 \\
& -3y_1 & = & 0 \\
& 2y_1 & \leq & -9 \\
& y_1 & & \text{free.}
\end{array}
$$

4. Let (P) denote the given linear programming problem.

Note that $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ is a feasible solution to (P). Therefore, by Theorem **??**, it suffices to find all values c_1, c_2 such that

(P) is not unbounded. This amounts to finding all values c_1, c_2 such that the dual problem of (P) has a feasible solution.

The dual problem of (P) is

$$
\begin{array}{rl}
\max & 2y_1 + y_2 \\
& 2y_1 + y_2 = c_1 \\
& y_1 + 3y_2 = c_2 \\
& y_1, y_2 \geq 0.
\end{array}
$$

The two equality constraints gives $\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \frac{3}{5}c_1 - \frac{1}{5}c_2 \\ -\frac{1}{5}c_1 + \frac{2}{5}c_2 \end{bmatrix}$. Thus, the dual problem is feasible if and only if c_1 and c_2 are real numbers satisfying

$$
\begin{array}{rl}
\frac{3}{5}c_1 - \frac{1}{5}c_2 \geq & 0 \\
-\frac{1}{5}c_1 + \frac{2}{5}c_2 \geq & 0,
\end{array}
$$

or more simply,

$$
\frac{1}{3}c_2 \leq c_1 \leq 2c_2.
$$

10. Sensitivity Analysis

Sensitivity analysis is an important tool for understanding the behavior of linear programs. It allows us to analyze how the optimal solution of a linear program changes as the coefficients of the constraints or the objective function are varied within certain bounds. In this section, we will present the basic concepts of sensitivity analysis and provide two examples to illustrate its use.

The first step in sensitivity analysis is to solve the linear program using a method such as the simplex method. Once the optimal solution has been obtained, we can then analyze how the solution changes as the coefficients of the constraints or the objective function are varied. For example, consider the following linear program:

$$\text{Maximize} \quad 3x_1 + 2x_2 \text{ Subject to} \qquad x_1 + x_2 \le 4 \; 2x_1 + x_2 \le 5 \qquad x_1, x_2 \ge 0$$

The optimal solution to this linear program is $x_1 = 2$, $x_2 = 2$, with an optimal objective value of $3x_1 + 2x_2 = 10$.

To perform sensitivity analysis, we can consider how the optimal solution changes as the right-hand side (RHS) of the constraints is varied. For example, suppose we increase the RHS of the first constraint by 1, to 5. This corresponds to the modified constraint $x_1 + x_2 \le 5$. The optimal solution to this modified linear program is $x_1 = 2$, $x_2 = 3$, with an optimal objective value of $3x_1 + 2x_2 = 12$. Thus, we can see that increasing the RHS of the first constraint by 1 has led to an increase in the optimal objective value.

Another example of sensitivity analysis is consider the effect of changing the coefficient of the objective function. For example, suppose we multiply the coefficient of x_1 by 2, to obtain the modified objective function $6x_1 + 2x_2$. The optimal solution to this modified linear program is $x_1 = 1$, $x_2 = 2$, with an optimal objective value of $6x_1 + 2x_2 = 8$. Thus, we can see that multiplying the coefficient of x_1 by 2 has led to a decrease in the optimal objective value.

In summary, sensitivity analysis is a useful tool for understanding how the optimal solution of a linear program changes as the coefficients of the constraints or the objective function are varied. It allows us to determine how robust the solution is to changes in the input data, and to identify the most critical factors affecting the solution.

11. Multi-Objective Optimization

11.1 Multi Objective Optimization and The Pareto Frontier

On Dealing with Ties and Multiple Objectives in Linear Programming

Consider a high end furniture manufacturer which builds dining tables and chairs out of expensive bocote and rosewood.

The manufacturer has an ongoing deal with a foreign sawmill which supplies them with 960 and 200 board-feet (bdft) of bocote and rosewood respectively each month.

A single table requires 80 bdft of bocote and 20 bdft of rosewood.

Each chair requires only 20 bdft of bocote but 10 bdft of rosewood.

$$P = \{(x,y) \in \mathbb{R}^2 :$$
$$80x + 20y \le 960$$
$$12x + 10y \le 200$$
$$x, y \ge 0\}$$
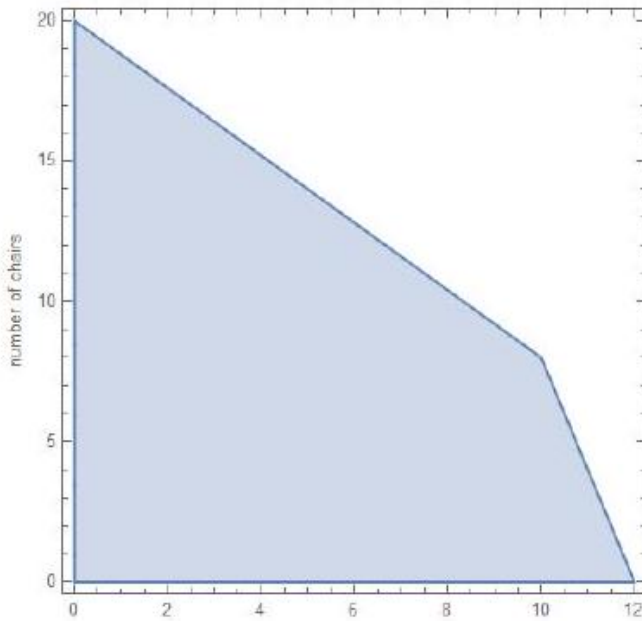
Suppose that each table will sells for \$7000 while a chair goes for \$1500. To increase profit we want to maximize

$$F(x,y) = 8000x + 2000y$$

over P. Having taken a linear programming class, the manager knows his way around these problems and begins the simplex method:

Maximize $8000x + 2000y$

s.t. $80x + 20y \leq 960$

$$12x + 10y \leq 200$$

$$x, y \geq 0$$

-4	-1	0	0	0
4	1	1	0	48
6	5	0	1	100

Maximize $4x + y$

s.t. $4x + y + s_1 = 48$

$$\frac{f}{2000} \quad \begin{matrix} 6x + 5y + s_2 = 100 \\ x, y \geq 0 \end{matrix}$$

Having found an optimal solution, the manager is quick to set up production. The best thing to do is produce 12 tables a month and no chairs!

But there are actually multiple optima!

How could we have noticed this this from the tableau? From the original formulation?

Is the manager's solution really the best?
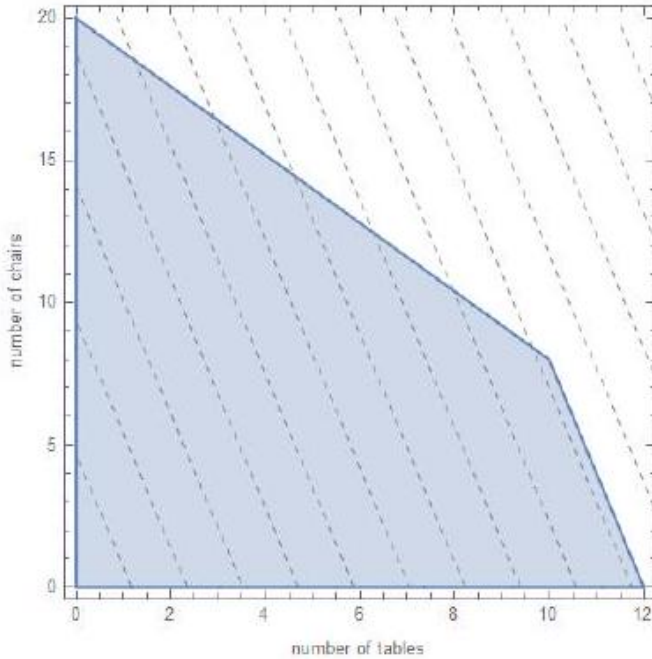
number of tables Having fired the prior manager for producing no chairs, a new and more competent manager is hired. This one knows that Dalbergia stevensonii (the tree which produces their preferred rosewood) is a threatened species and decides that she doesn't want to waste any more rosewood than is necessary.

After some investigation, she finds that table production wastes nearly 10 bdft of rosewood per table while chairs are dramatically more efficient wasting only 2 bdft per chair. She comes up with a new, secondary objective function that she would like to minimize:

$$w(x, y) = 10x + 2y.$$

Having noticed that there are multiple profit-maximizers, she formulates a new problem to break the tie:

$$\text{Minimize } 10x + 2y$$
$$\text{s.t.} \quad 80x + 20y = 960$$
$$x \in [10, 12]$$
$$y \in [0, 8].$$

This is easy in this case because the set of profit-optimal solutions is simple.

Because this is an LP, the optimal solution will be at an extreme point; there are only two here, so the problem reduces to

$$\arg\min\{10x + 2y : (x, y) \in \{(12, 0), (10, 8)\}\}$$

Therefore, swapping out some tables for chairs reduces waste and without affecting revenue!

What the manager just did is called the Ordered Criteria or Lexicographic method for Multi-Objective Optimization. After a few months, the manager convinces the owners that reducing waste is worth a small

loss in profit. The owners concede to a 30% loss in revenue and our manager gets to work on a new model:

$$\text{Minimize } 10x + 2y$$
$$\text{s.t.} \quad 8000x + 2000y \geq (\alpha)96000$$
$$80x + 20y \leq 960$$
$$12x + 10y \leq 200$$
$$x, y \geq 0$$

where $\alpha = 0.7$ This new constraint limits us to solutions which offered at least 70% of maximum possible revenue.
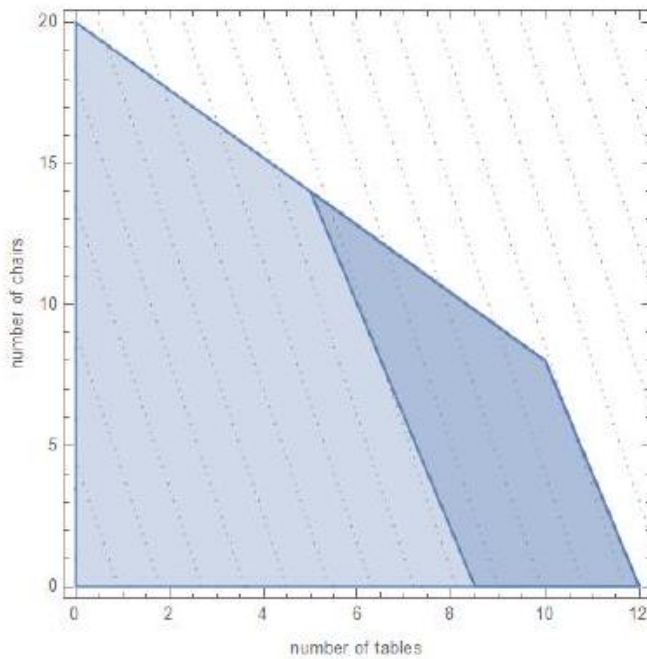
The strategy is the called the Benchmark or Rollover method because we choose a benchmark for one of our objectives (revenue in this case), roll that benchmark into the constraints, and optimize for the second objective (waste).

Notice that if we set α to 1 , the rollover problem is equivalent to the lexicographic problem. Either approach requires a known optimal value to the first objective function.

Interestingly, our rollover solution is NOT an extreme point to the ORIGINAL feasible region. Given a set P and some number of functions $f_i : P \to \mathbb{R}$ that we seek to maximize, we call a point $\mathbf{x} \in P$ Pareto Optimal or Efficient if there does not exist another point $\bar{\mathbf{x}} \in P$ such that

- $f_i(\bar{\mathbf{x}}) > f_i(\mathbf{x})$ for some i and

$\to f_j(\bar{\mathbf{x}}) \geq f_j(\mathbf{x})$ for all $j \neq i$.

That is, we cannot make any objective better without making some other objective worse.

The Pareto Frontier is the set of all Pareto optimal points for some problem. Which of these points is Pareto optimal?

What is the frontier of this problem?
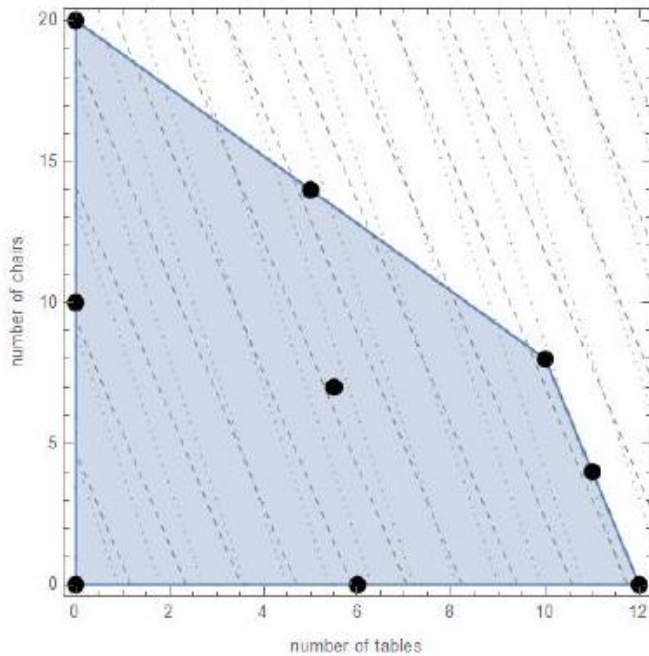
number of tables The rollover method is generalized in Goal Programming

By varying α, it is possible to generate many distinct efficient solutions.

However, this method can generate inefficient solutions if the underlying model is poorly constructed.
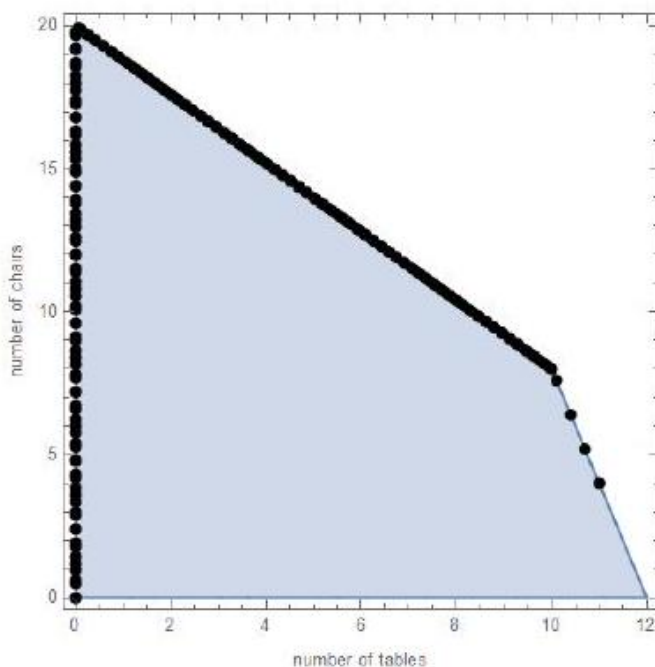
number of tables It is more common to see a Pareto frontier plotted with respect to its objectives.
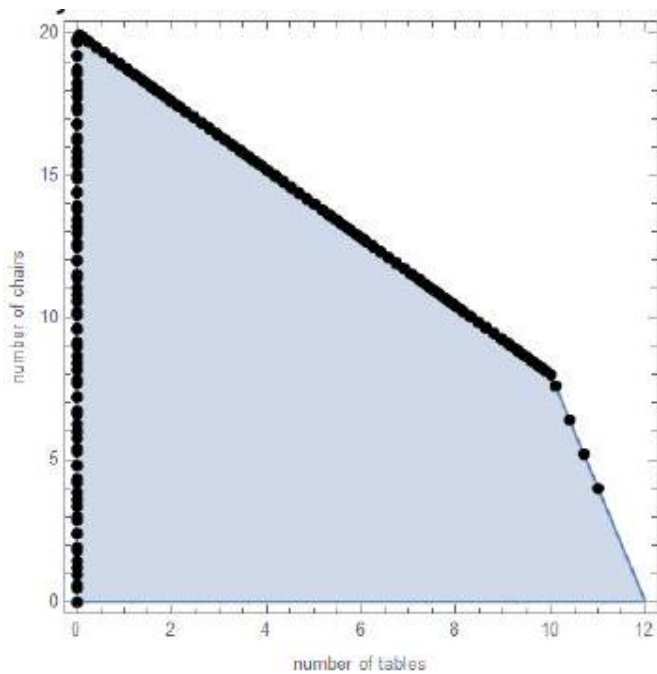
number of table
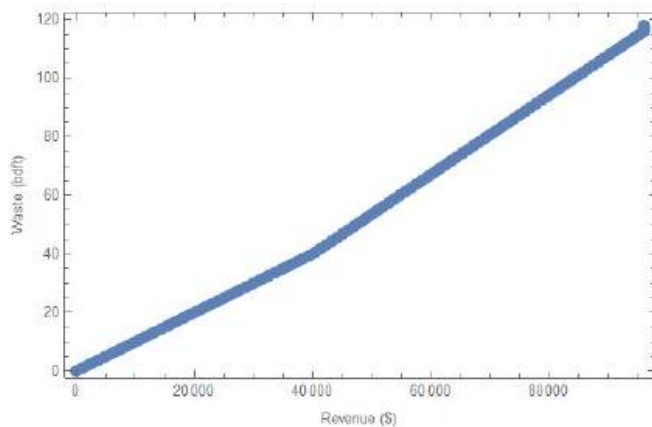
One of the owners of our manufactory decides to explore possible planning himself; he implements the multi-objective method that he remembers, Scalarization by picking some arbitrary constant $\lambda \in [0,1]$ and combining his two objectives like so:

$$\begin{aligned} \text{Minimize} \quad & \lambda(8000x + 2000y) + (1-\lambda)(10x + 2y) \\ \text{s.t.} \quad & 80x + 20y \leq 960 \\ & 12x + 10y \leq 200 \\ & x, y \geq 0 \end{aligned}$$

What is the benefit of this method?

Where does it fall short?

11.2 What points will the Scalarization method find if we vary λ?
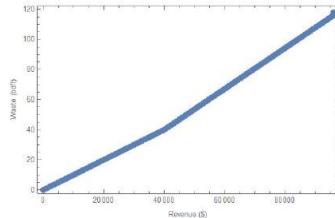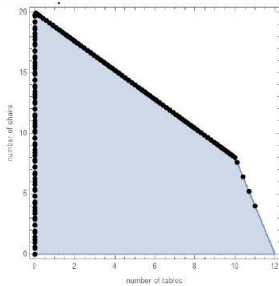
These are all nice ideas, but the problem presented above is neither difficult nor practical.

What are some areas that a Pareto frontier would be actually useful?

11.3 Political Redistricting [3]
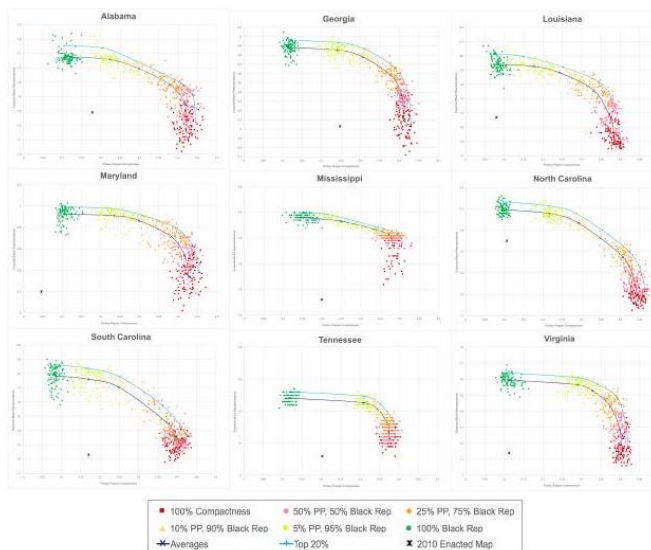
11.4 Portfolio Optimization [5]

11.5 Simulated Portfolio Optimization based on Efficient Frontier
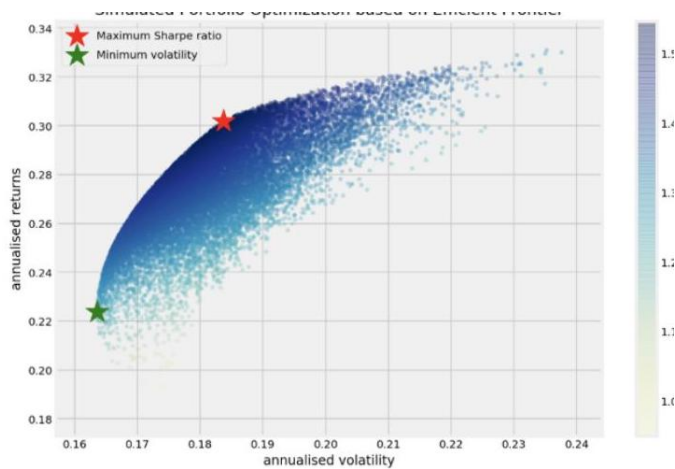
11.6 Aircraft Design [1]
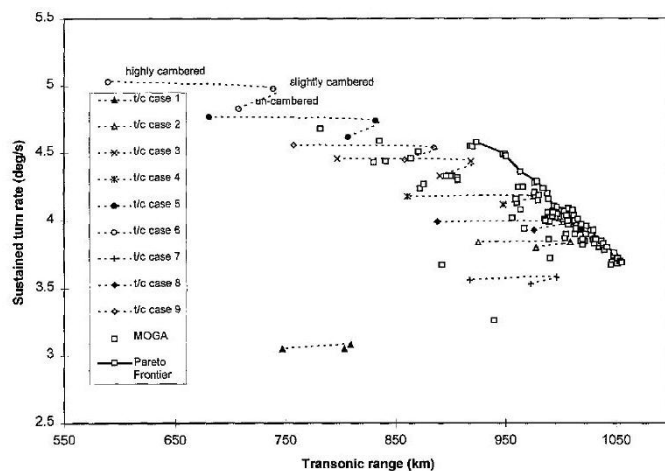
11.7 Vehicle Dynamics [4]
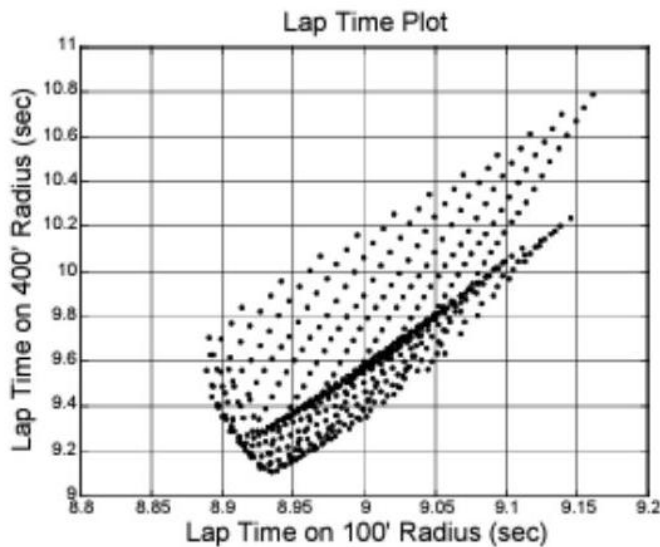
Figure 7: Grid Search Results in the Performance Space

11.8 Sustainable Constriction [2]
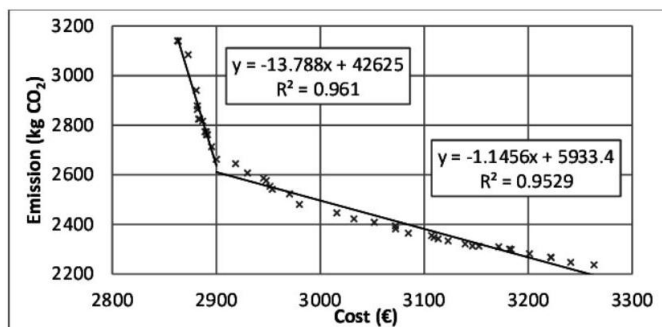
11.9 References

S. Fenwick and John C. Harris. the application of pareto frontier methods in the multidisciplinary wing design of a generic modern military delta aircraft: Semantic scholar, Jan 1999.

URL: https://WWW. semanticscholar. org/paper/

The-application-of-Pareto-frontier-methods-in-the-a-Fenwick-Harris/ fced 00a59 d200c2c74 ed 655 a 457344 bcleea 6 ff 5.

T García-Segura, V Yepes, and J Alcalá.

Sustainable design using multiobjective optimization of high-strength concrete i-beams. In The 2014 International Conference on High Performance and Optimum Design of Structures and Materials HPSM/OPTI, volume 137, pages $347 - 358, 2014$.

URL: https://www. researchgate.net/publication/271439836_Sustainable_design_using_ multiobjective_optimization_of_high-strength_concrete_l-beams.

Nicholas Goedert, Robert Hildebrand, Laurel Travis, Matthew Pierson, and Jamie Fravel. Black representation and district compactness in southern congressional districts. not yet published, ask Dr. Hildebrand for it.

Edward M Kasprzak and Kemper E Lewis.

Pareto analysis in multiobjective optimization using the collinearity theorem and scaling method. Structural and Multidisciplinary Optimization.

Ricky Kim.

Efficient frontier portfolio optimisation in python, Jun 2021.

URL: https://towardsdatascience. com/

efficient-frontier-portfolio-optimisation-in-python-e7844051e7f.

Part II

Discrete Algorithms

12. Graph Algorithms

12.1 Graph Theory and Network Flows

In the modern world, planning efficient routes is essential for business and industry, with applications as varied as product distribution, laying new fiber optic lines for broadband internet, and suggesting new friends within social network websites like Facebook.

This field of mathematics started nearly 300 years ago as a look into a mathematical puzzle (we'll look at it in a bit). The field has exploded in importance in the last century, both because of the growing complexity of business in a global economy and because of the computational power that computers have provided us.

12.2 Graphs

12.2.1. Drawing Graphs

Example 12.1

Here is a portion of a housing development from Missoula, Montana[a]. As part of her job, the development's lawn inspector has to walk down every street in the development making sure homeowners' landscaping conforms to the community requirements.

Naturally, she wants to minimize the amount of walking she has to do. Is it possible for her to walk down every street in this development without having to do any backtracking? While you might be able to answer that question just by looking at the picture for a while, it would be ideal to be able to answer the question for any picture regardless of its complexity.

To do that, we first need to simplify the picture into a form that is easier to work with. We can do that by drawing a simple line for each street. Where streets intersect, we will place a dot.

[a]Same Beebe. http://www.flickr.com/photos/sbeebe/2850476641/
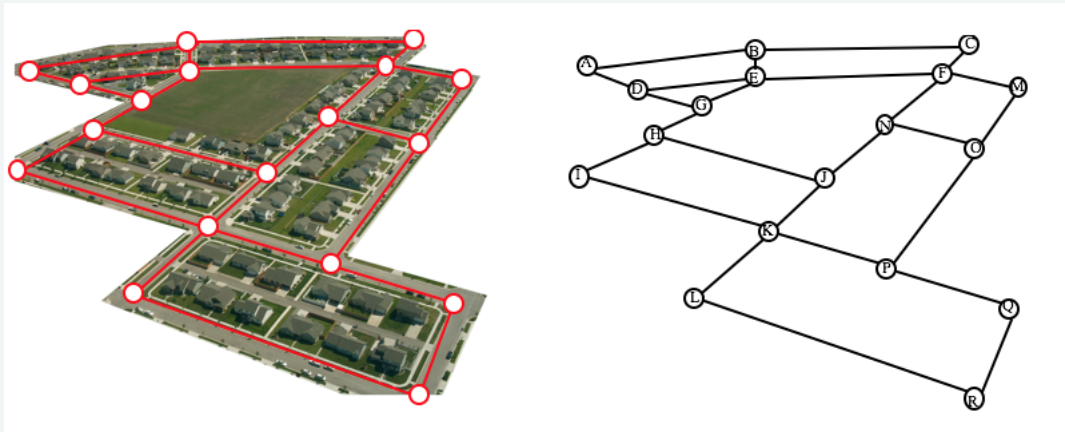
This type of simplified picture is called a **graph**.

Definition 12.2: Graphs, Vertices, and Edges

A graph consists of a set of dots, called vertices, and a set of edges connecting pairs of vertices.

While we drew our original graph to correspond with the picture we had, there is nothing particularly important about the layout when we analyze a graph. Both of the graphs below are equivalent to the one drawn above since they show the same edge connections between the same vertices as the original graph.
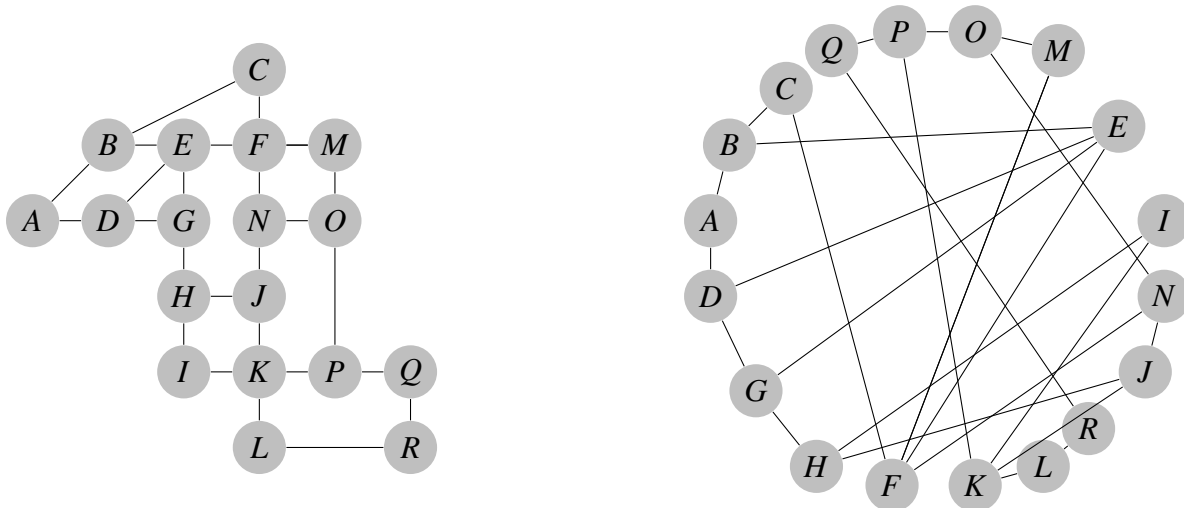
You probably already noticed that we are using the term graph differently than you may have used the term in the past to describe the graph of a mathematical function.

Example 12.3

Back in the 18th century in the Prussian city of Königsberg, a river ran through the city and seven bridges crossed the forks of the river. The river and the bridges are highlighted in the picture to the right

Picture

As a weekend amusement, townsfolk would see if they could find a route that would take them across every bridge once and return them to where they started.

Leonard Euler (pronounced OY-lur), one of the most prolific mathematicians ever, looked at this problem in 1735, laying the foundation for graph theory as a field in mathematics. To analyze this problem, Euler introduced edges representing the bridges:
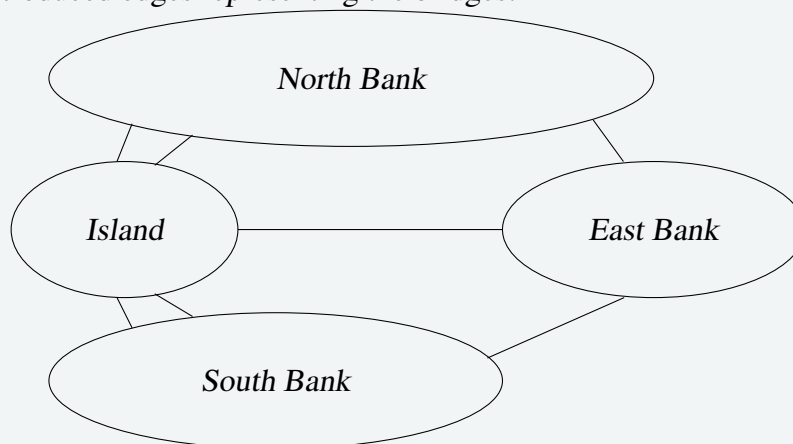
Since the size of each land mass it is not relevant to the question of bridge crossings, each can be shrunk down to a vertex representing the location:
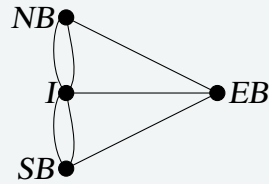
Notice that in this graph there are *two edges* connecting the north bank and island, corresponding to the two bridges in the original drawing. Depending upon the interpretation of edges and vertices appropriate to a scenario, it is entirely possible and reasonable to have more than one edge connecting two vertices.

While we haven't answered the actual question yet of whether or not there is a route which crosses every bridge once and returns to the starting location, the graph provides the foundation for exploring this question.

12.3 Definitions

While we loosely defined some terminology earlier, we now will try to be more specific.

Definition 12.4: Vertex

A vertex is a dot in the graph that could represent an intersection of streets, a land mass, or a general location, like "work? or "school". Vertices are often connected by edges. Note that vertices only occur when a dot is explicitly placed, not whenever two edges cross. Imagine a freeway overpass – the freeway and side street cross, but it is not possible to change from the side street to the freeway at that point, so there is no intersection and no vertex would be placed.

Definition 12.5: Edges

Edges connect pairs of vertices. An edge can represent a physical connection between locations, like a street, or simply that a route connecting the two locations exists, like an airline flight.

Definition 12.6: Loop

A loop is a special type of edge that connects a vertex to itself. Loops are not used much in street network graphs.
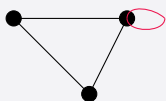
Definition 12.7: Degree of a vertex

The degree of a vertex is the number of edges meeting at that vertex. It is possible for a vertex to have a degree of zero or larger.

Degree 0	Degree 1	Degree 2	Degree 3	Degree 4

Definition 12.8: Path

A path is a sequence of vertices using the edges. Usually we are interested in a path between two vertices. For example, a path from vertex A to vertex M is shown below. It is one of many possible paths in this graph.
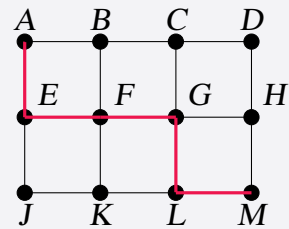
Definition 12.9: Circuit (a.k.a. cycle)

A circuit (a.k.a. cycle) is a path that begins and ends at the same vertex. A circuit (a.k.a. cycle) starting and ending at vertex A is shown below.
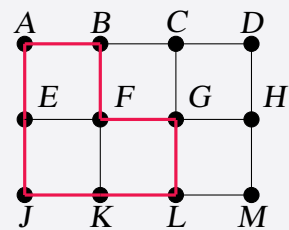
Definition 12.10: Connected

*A graph is connected if there is a path from any vertex to any other vertex. Every graph drawn so far has been connected. The graph below is **disconnected**; there is no way to get from the vertices on the left to the vertices on the right.*
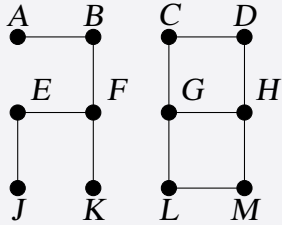
Definition 12.11: Weights

Depending upon the problem being solved, sometimes weights are assigned to the edges. The weights could represent the distance between two locations, the travel time, or the travel cost. It is important to note that the distance between vertices in a graph does not necessarily correspond to the weight of an edge.

Exercise 12.12

The graph below shows 5 cities. The weights on the edges represent the airfare for a one-way flight between the cities.

  a. *How many vertices and edges does the graph have?*
  b. *Is the graph connected?*
  c. *What is the degree of the vertex representing LA?*
  d. *If you fly from Seattle to Dallas to Atlanta, is that a path or a circuit?*
  e. *If you fly from LA to Chicago to Dallas to LA, is that a path or a circuit?*
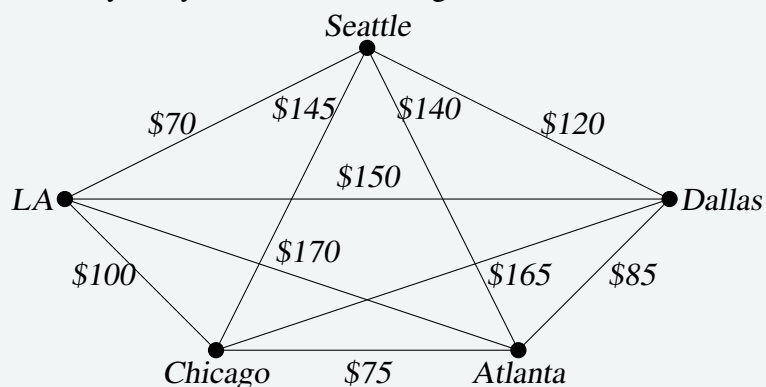
12.4 Shortest Path

Outcomes

- *What is the problem statement?*
- *How to use Dijkstra's algorihtm*
- *Software solutions*

Resources

- *How Dijkstra's algorithm works*
- *YouTube Video of Dijkstra's Algorithm*
- *Python Example using Networkx and also showing Dijkstra's algorithm*

When you visit a website like Google Maps or use your Smartphone to ask for directions from home to your Aunt's house in Pasadena, you are usually looking for a shortest path between the two locations. These computer applications use representations of the street maps as graphs, with estimated driving times as edge weights.

While often it is possible to find a shortest path on a small graph by guess-and-check, our goal in this chapter is to develop methods to solve complex problems in a systematic way by following **algorithms**. An algorithm is a step-by-step procedure for solving a problem. Dijkstra's (pronounced dike-stra) algorithm will find the shortest path between two vertices.

Dijkstra's Algorithm

1. Mark the ending vertex with a distance of zero. Designate this vertex as current.

2. Find all vertices leading to the current vertex. Calculate their distances to the end. Since we already know the distance the current vertex is from the end, this will just require adding the most recent edge. Don't record this distance if it is longer than a previously recorded distance.

3. Mark the current vertex as visited. We will never look at this vertex again.

4. Mark the vertex with the smallest distance as current, and repeat from step 2.

Example 12.13

Suppose you need to travel from Yakima, WA (vertex Y) to Tacoma, WA (vertex T) Looking at a map, it looks like driving through Auburn (A) then Mount Rainier (MR) might be shortest, but it's not totally clear since that road is probably slower than taking the major highway through North Bend (NB). A graph with travel times in minutes is shown below. An alternate route through Eatonville (E) and Packwood (P) is also shown.
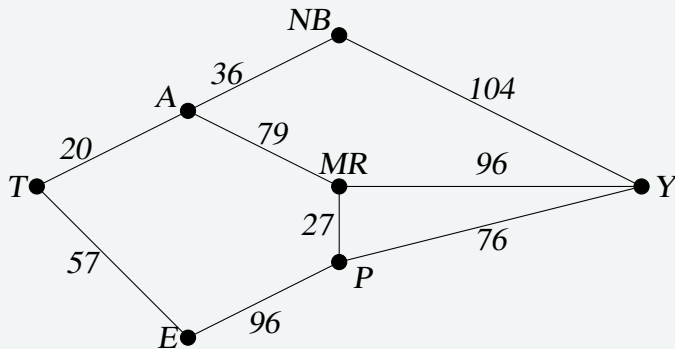
Step 1: Mark the ending vertex with a distance of zero. The distances will be recorded in [brackets] after the vertex name.
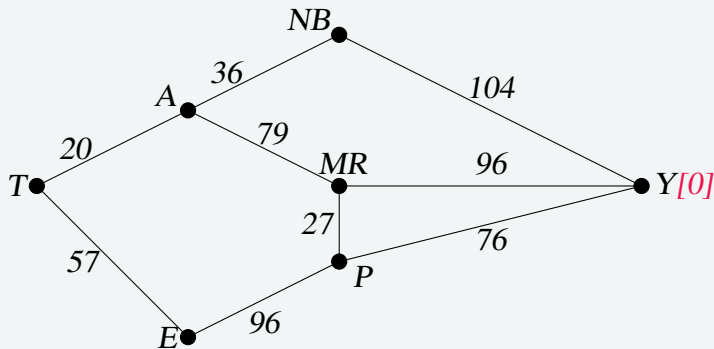
Step 2: For each vertex leading to Y, we calculate the distance to the end. For example, NB is a distance of 104 from the end, and MR is 96 from the end. Remember that distances in this case refer to the travel time in minutes.
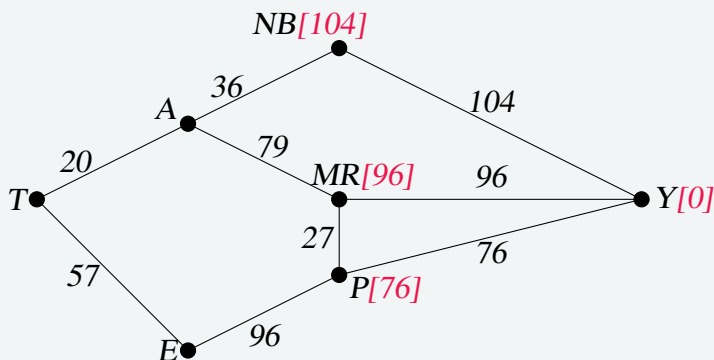
Step 3 & 4: We mark Y as visited, and mark the vertex with the smallest recorded distance as current. At this point, P will be designated current. Back to step 2.

Step 2 (#2): For each vertex leading to P (and not leading to a visited vertex) we find the distance from the end. Since E is 96 minutes from P, and we've already calculated P is 76 minutes from Y, we can compute that E is $96 + 76 = 172$ minutes from Y.

If we make the same computation for MR, we'd calculate $76 + 27 = 103$. Since this is larger than the previously recorded distance from Y to MR, we will not replace it.
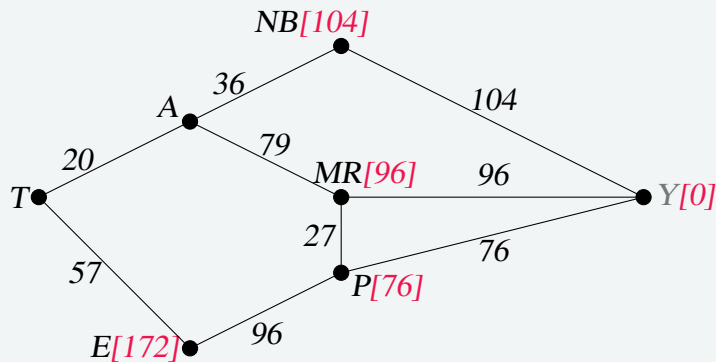
Step 3 & 4 (#2): We mark P as visited, and designate the vertex with the smallest recorded distance as current: MR. Back to step 2.

Step 2 (#3): For each vertex leading to MR (and not leading to a visited vertex) we find the distance to the end. The only vertex to be considered is A, since we've already visited Y and P. Adding MR's distance 96 to the length from A to MR gives the distance $96 + 79 = 175$ minutes from A to Y.
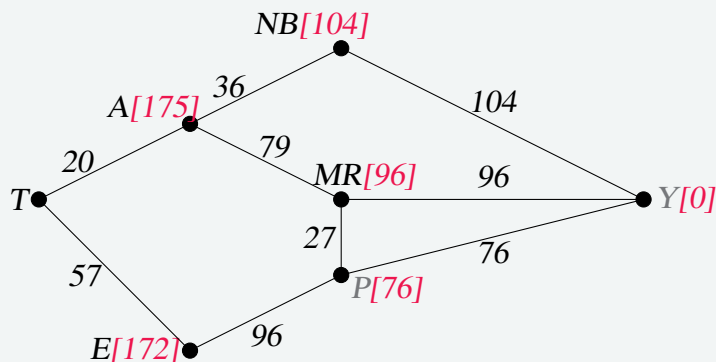
Step 3 & 4 (#3): We mark MR as visited, and designate the vertex with smallest recorded distance as current: NB. Back to step 2.

Step 2 (#4): For each vertex leading to NB, we find the distance to the end. We know the shortest distance from NB to Y is 104 and the distance from A to NB is 36, so the distance from A to Y through NB is $104 + 36 = 140$. Since this distance is shorter than the previously calculated distance from Y to A through MR, we replace it.
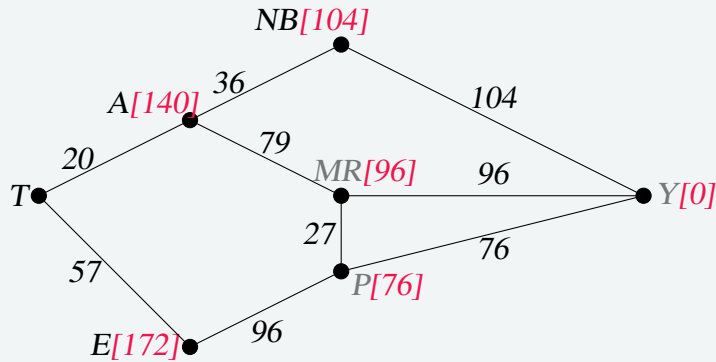
Step 3 & 4 (#4): We mark NB as visited, and designate A as current, since it now has the shortest distance.

Step 2 (#5): T is the only non-visited vertex leading to A, so we calculate the distance from T to Y through A: $20 + 140 = 160$ minutes.
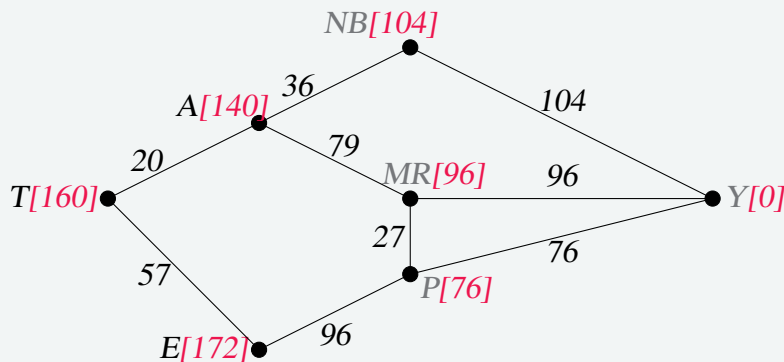
Step 3 & 4 (#5): We mark A as visited, and designate E as current.

Step 2 (#6): The only non-visited vertex leading to E is T. Calculating the distance from T to Y through E, we compute $172 + 57 = 229$ minutes. Since this is longer than the existing marked time, we do not replace it.

Step 3 (#6): We mark E as visited. Since all vertices have been visited, we are done.

From this, we know that the shortest path from Yakima to Tacp,a will take 160 minutes. Tracking which sequence of edges yielded 160 minutes, we see the shortest path is Y-NB-A-T.

Dijkstra's algorithm is an **optimal algorithm**, meaning that it always produces the actual shortest path, not just a path that is pretty short, provided one exists. This algorithm is also **efficient**, meaning that it can be implemented in a reasonable amount of time. Dijkstra's algorithm takes around V^2 calculations, where V is the number of vertices in a graph[1]. A graph with 100 vertices would take around 10,000 calculations. While that would be a lot to do by hand, it is not a lot for computer to handle. It is because of this efficiency that your car's GPS unit can compute driving directions in only a few seconds.

---

[1]It can be made to run faster through various optimizations to the implementation.

In contrast, an **inefficient** algorithm might try to list all possible paths then compute the length of each path. Trying to list all possible paths could easily take 10^{25} calculations to compute the shortest path with only 25 vertices; that's a 1 with 25 zeros after it! To put that in perspective, the fastest computer in the world would still spend over 1000 years analyzing all those paths.

Example 12.14: Dijkstra's algorithm example

We would like to find a shortest path in the graph from node a to node g. See Code for python code to solve this problem and create these graphics.
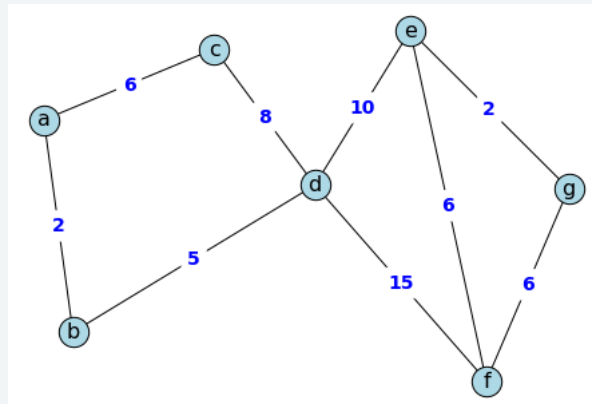
We will initialize our algorithm at node 'a'.
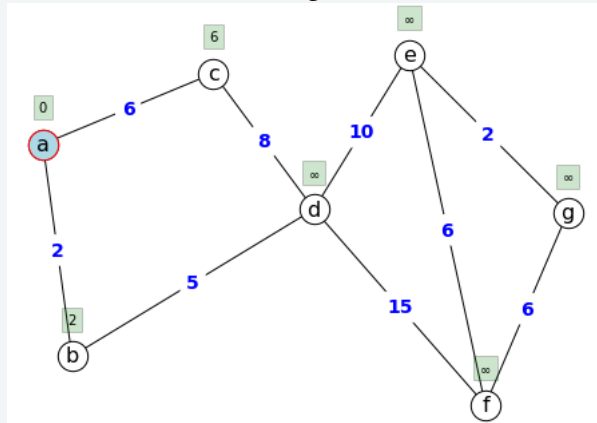
current	a	b	c	d	e	f	g
a	0	2	6	∞	∞	∞	∞
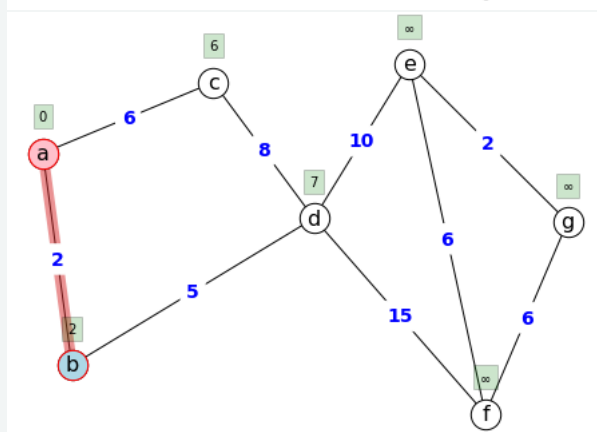
current	a	b	c	d	e	f	g
b	0	2	6	7	∞	∞	∞
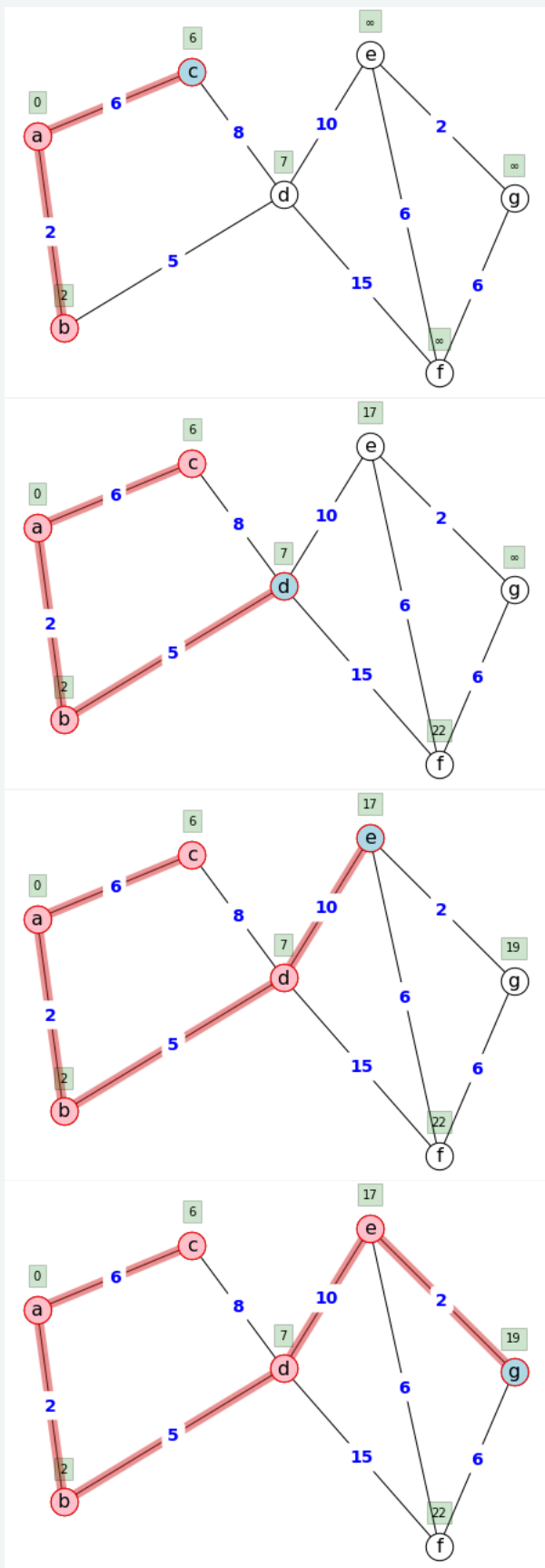
current	a	b	c	d	e	f	g
c	0	2	6	7	∞	∞	∞

current	a	b	c	d	e	f	g
d	0	2	6	7	17	22	∞

current	a	b	c	d	e	f	g
e	0	2	6	7	17	22	19

current	a	b	c	d	e	f	g
g	0	2	6	7	17	22	19
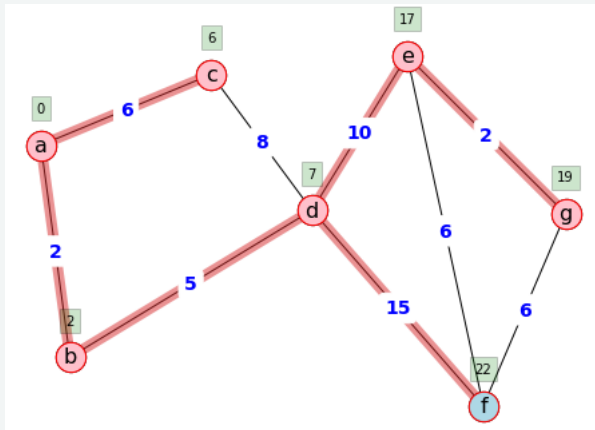
current	a	b	c	d	e	f	g
f	0	2	6	7	17	22	19

We can now summarize our calculations that followed Dijkstra's algorithm.
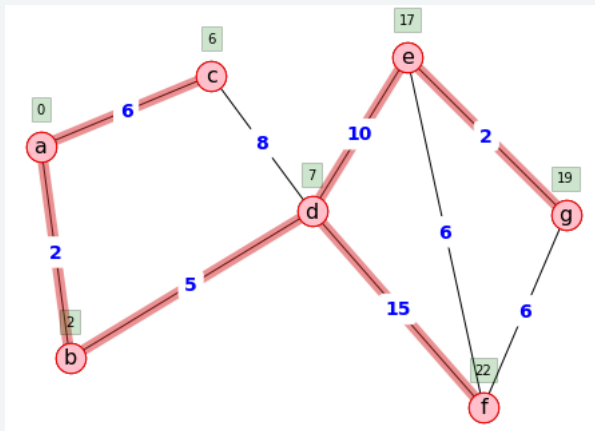
current	a	b	c	d	e	f	g
a	0	2	6	∞	∞	∞	∞
b	0	2	6	7	∞	∞	∞
c	0	2	6	7	∞	∞	∞
d	0	2	6	7	17	22	∞
e	0	2	6	7	17	22	19
g	0	2	6	7	17	22	19
f	0	2	6	7	17	22	19

FINAL SOLUTION   *The shortest path from a to g is the path a - b - d - e - g,*
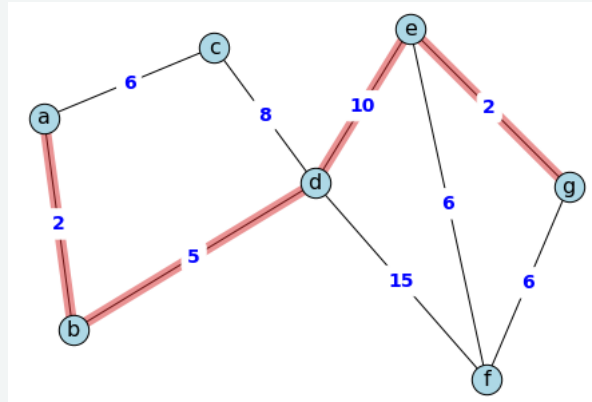
and has length

$$2 + 5 + 10 + 2 = 19.$$

Example 12.15

A shipping company needs to route a package from Washington, D.C. to San Diego, CA. To minimize costs, the package will first be sent to their processing center in Baltimore, MD then sent as part of mass shipments between their various processing centers, ending up in their processing center in Bakersfield, CA. From there it will be delivered in a small truck to San Diego.

The travel times, in hours, between their processing centers are shown in the table below. Three hours has been added to each travel time for processing. Find the shortest path from Baltimore to Bakersfield.

	Baltimore	Denver	Dallas	Chicago	Atlanta	Bakersfield
Baltimore	*			15	14	
Denver		*		18	24	19
Dallas			*	18	15	25
Chicago	15	18	18	*	14	
Atlanta	14	24	15	14	8	
Bakersfield		19	25			*

While we could draw a graph, we can also work directly from the table.

Step 1: The ending vertex, Bakersfield, is marked as current.

Step 2: All cities connected to Bakersfield, in this case Denver and Dallas, have their distances calculated; we'll mark those distances in the column headers.

Step 3 & 4: Mark Bakersfield as visited. Here, we are doing it by shading the corresponding row and column of the table. We mark Denver as current, shown in bold, since it is the vertex with the shortest distance.

	Baltimore	**Denver** [19]	Dallas [25]	Chicago	Atlanta	Bakersfield [0]
Baltimore	*			15	14	
Denver		*		18	24	19
Dallas			*	18	15	25
Chicago	15	18	18	*	14	
Atlanta	14	24	15	14	8	
Bakersfield		19	25			*

Step 2 (#2): For cities connected to Denver, calculate distance to the end. For example, Chicago is 18 hours from Denver, and Denver is 19 hours from the end, the distance for Chicago to the end is $18 + 19 = 37$ (Chicago to Denver to Bakersfield). Atlanta is 24 hours from Denver, so the distance to the end is $24 + 19 = 43$ (Atlanta to Denver to Bakersfield).

Step 3 & 4 (#2): We mark Denver as visited and mark Dallas as current.

	Baltimore	Denver [19]	**Dallas [25]**	Chicago [37]	Atlanta [43]	Bakersfield [0]
Baltimore	*			15	14	
Denver		*		18	24	19
Dallas			*	18	15	25
Chicago	15	18	18	*	14	
Atlanta	14	24	15	14	8	
Bakersfield		19	25			*

Step 2 (#3): For cities connected to Dallas, calculate the distance to the end. For Chicago, the distance from Chicago to Dallas is 18 and from Dallas to the end is 25, so the distance from Chicago to the end through Dallas would be $18 + 25 = 43$. Since this is longer than the currently marked distance for Chicago, we do not replace it. For Atlanta, we calculate $15 + 25 = 40$. Since this is shorter than the currently marked distance for Atlanta, we replace the existing distance.

Step 3 & 4 (#3): We mark Dallas as visited, and mark Chicago as current.

	Baltimore	Denver [19]	Dallas [25]	**Chicago [37]**	Atlanta [40]	Bakersfield [0]
Baltimore	*			15	14	
Denver		*		18	24	19
Dallas			*	18	15	25
Chicago	15	18	18	*	14	
Atlanta	14	24	15	14	8	
Bakersfield		19	25			*

Step 2 (#4): Baltimore and Atlanta are the only non-visited cities connected to Chicago. For Baltimore, we calculate $15 + 37 = 52$ and mark that distance. For Atlanta, we calculate $14 + 37 = 51$. Since this is longer than the existing distance of 40 for Atlanta, we do not replace that distance.

Step 3 & 4 (#4): Mark Chicago as visited and Atlanta as current.

	Baltimore [52]	Denver [19]	Dallas [25]	Chicago [37]	**Atlanta [40]**	Bakersfield [0]
Baltimore	*			15	14	
Denver		*		18	24	19
Dallas			*	18	15	25
Chicago	15	18	18	*	14	
Atlanta	14	24	15	14	8	
Bakersfield		19	25			*

Step 2 (#5): The distance from Atlanta to Baltimore is 14. Adding that to the distance already calculated for Atlanta gives a total distance of $14 + 40 = 54$ hours from Baltimore to Bakersfield through Atlanta. Since this is larger than the currently calculated distance, we do not replace the distance for Baltimore.

Step 3 & 4 (#5): We mark Atlanta as visited. All cities have been visited and we are done.

> The shortest route from Baltimore to Bakersfield will take 52 hours, and will route through Chicago and Denver.

Exercise 12.16

Find the shortest path between vertices A and G in the graph below.
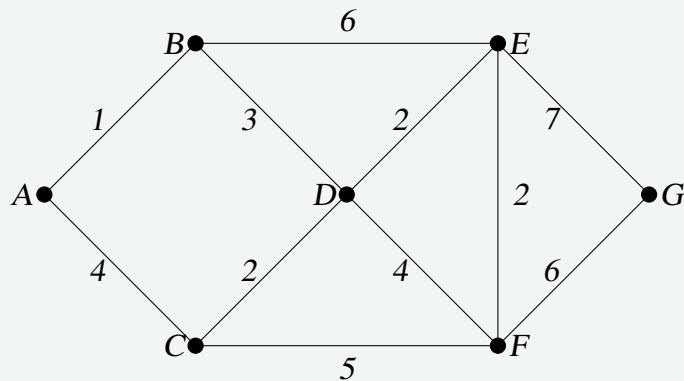
12.5 Spanning Trees

A company requires reliable internet and phone connectivity between their five offices (named A, B, C, D, and E for simplicity) in New York, so they decide to lease dedicated lines from the phone company. The phone company will charge for each link made. The costs, in thousands of dollars per year, are shown in the graph.
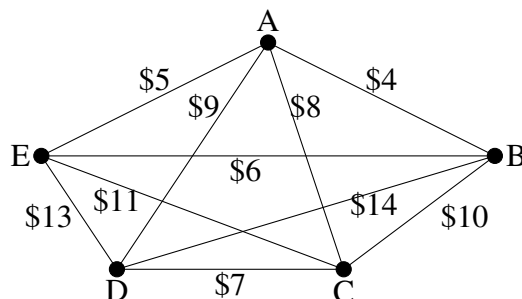
In this case, we don't need to find a circuit, or even a specific path; all we need to do is make sure we can make a call from any office to any other. In other words, we need to be sure there is a path from any vertex to any other vertex.

> **Definition 12.17: Spanning Tree**
>
> *A spanning tree is a connected graph using all vertices in which there are no circuits. In other words, there is a path from any vertex to any other vertex, but no circuits.*

Some examples of spanning trees are shown below. Notice there are no circuits in the trees, and it is fine to have vertices with degree higher than two.
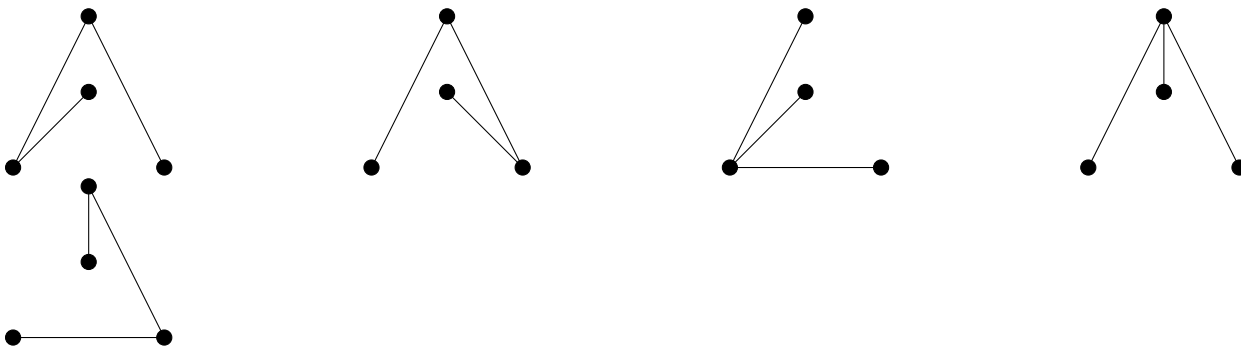
Usually we have a starting graph to work from, like in the phone example above. In this case, we form our spanning tree by finding a **subgraph** – a new graph formed using all the vertices but only some of the edges from the original graph. No edges will be created where they didn't already exist.

Of course, any random spanning tree isn't really what we want. We want the **minimum cost spanning tree (MCST)**.

> **Definition 12.18: Minimum Cost Spanning Tree (MCST)**
>
> *The minimum cost spanning tree is the spanning tree with the smallest total edge weight.*

A nearest neighbor style approach doesn't make as much sense here since we don't need a circuit, so instead we will take an approach similar to sorted edges.

Kruskal's Algorithm

1. Select the cheapest unused edge in the graph.

2. Repeat step 1, adding the cheapest unused edge, unless:

   - adding the edge would create a circuit.

3. Repeat until a spanning tree is formed.

Example 12.19

Using our phone line graph from above, begin adding edges:

AB $4   OK
AE $5   OK
BE $6   reject – closes circuit ABEA
DC $7   OK
AC $8   OK
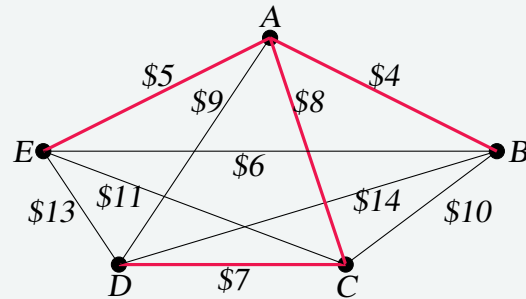
At this point we stop – every vertex is now connected, so we have formed a spanning tree with cost $24 thousand a year.

Remarkably, Kruskal's algorithm is both optimal and efficient; we are guaranteed to always produce the optimal MCST.

Example 12.20

The power company needs to lay updated distribution lines connecting the ten Oregon cities below to the power grid. How can they minimize the amount of new line to lay?

	Ashland	Astoria	Bend	Corvallis	Crater Lake	Eugene	Newport	Portland	Salem	Seaside
Ashland	–	374	200	223	108	178	252	285	240	356
Astoria	374	–	255	166	433	199	135	95	136	17
Bend	200	255	–	128	277	128	180	160	131	247
Corvalis	223	166	128	–	430	47	52	84	40	155
Crater Lake	108	433	277	430	–	453	478	344	389	423
Eugene	178	199	128	47	453	–	91	110	64	181
Newport	252	135	180	52	478	91	–	114	83	117
Portland	285	95	160	84	344	110	114	–	47	78
Salem	240	136	131	40	389	64	83	47	–	118
Seaside	356	17	247	155	423	181	117	78	118	–

Using Kruskal's algorithm, we add edges from cheapest to most expensive, rejecting any that close a circuit. We stop when the graph is connected.

Seaside to Astoria	17 miles	
Corvallis to Salem	40 miles	
Portland to Salem	47 miles	
Corvallis to Eugene	47 miles	
Corvallis to Newport	52 miles	
Salem to Eugene	reject – closes circuit	
Portland to Seaside	78 miles	

The graph up to this point is shown to the right.
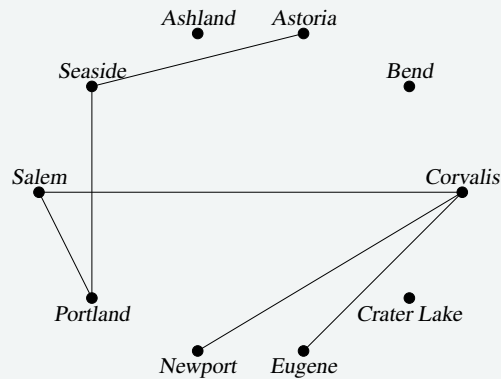
Continuing,

Newport to Salem	reject
Corvallis to Portland	reject
Eugene to Newport	reject
Portland to Astoria	reject
Ashland to Crater Lake	108 miles
Eugene to Portland	reject
Newport to Portland	reject
Newport to Seaside	reject
Salem to Seaside	reject
Bend to Eugene	128 miles
Bend to Salem	reject
Astoria to Newport	reject
Salem to Astoria	reject
Corvallis to Seaside	reject
Portland to Bend	reject
Astoria to Corvallis	reject
Eugene to Ashland	178 miles
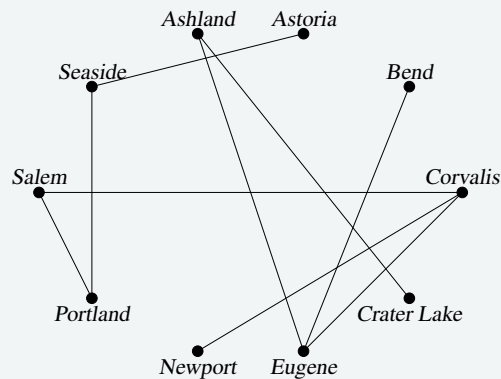
This connects the graph. The total length of cable to lay would be 695 miles.

Exercise 12.21: Min Cost Spanning Tree

Find a minimum cost spanning tree on the graph below using Kruskal's algorithm.
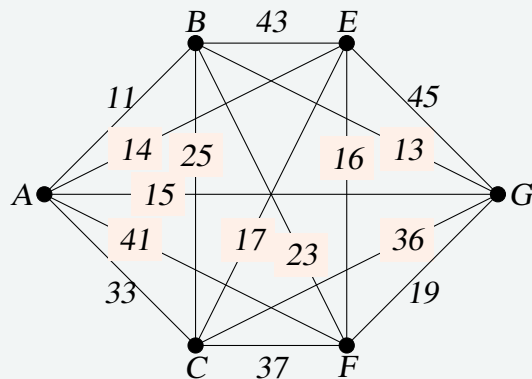
12.6 Exercise Answers

1. (a) 5 vertices, 10 edges

   (b) Yes, it is connected.

   (c) The vertex is degree 4.

   (d) A path

   (e) A circuit

2. The shortest path is ABDEG, with length 13.

3. Yes, all vertices have even degree so this graph has an Euler Circuit. There are several possibilities. One is: ABEGFCDFEDBCA

4.

   This graph can be eulerized by duplicating the edge BC, as shown. One possible Euler circuit on the eulerized graph is ACDBCBA.
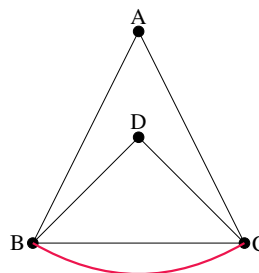
5. At each step, we look for the nearest location we haven't already visited. From B the nearest computer is E with time 24.

From E, the nearest computer is D with time 11.
From D the nearest is A with time 12.
From A the nearest is C with time 34.
From C, the only computer we haven't visited is F with time 27.
From F, we return back to B with time 50.

The NNA circuit from B is BEDACFB with time 158 milliseconds.

Using NNA again from other starting vertices:
Starting at A: ADEBCFA: time 146
Starting at C: CDEBAFC: time 167
Starting at D: DEBCFAD: time 146
Starting at E: EDACFBE: time 158
Starting at F: FDEBCAF: time 158

The RNNA found a circuit with time 146 milliseconds: ADEBCFA. We could also write this same circuit starting at B if we wanted: BCFADEB or BEDAFCB.

6.

AB: Add, cost 11
BG: Add, cost 13
AE: Add, cost 14
EF: Add, cost 15
EC: Skip (degree 3 at E)
FG: Skip (would create a circuit not including C)
BF, BC, AG, AC: Skip (would cause a vertex to have degree 3)
GC: Add, cost 36
CF: Add, cost 37, completes the circuit

Final circuit: ABGCFEA
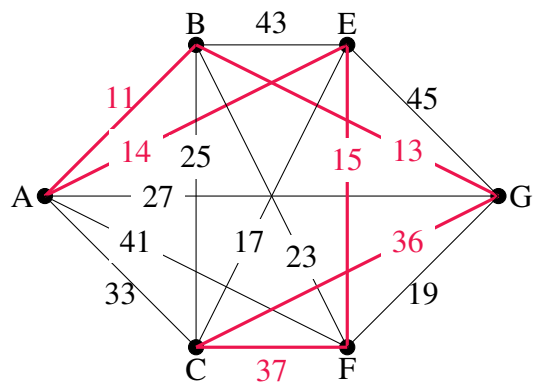
7. (**??**)

AB: Add, cost 11
BG: Add, cost 13
AE: Add, cost 14
AG: Skip, would create circuit ABGA
EF: Add, cost 16
EC: Add, cost 17

This completes the spanning tree.
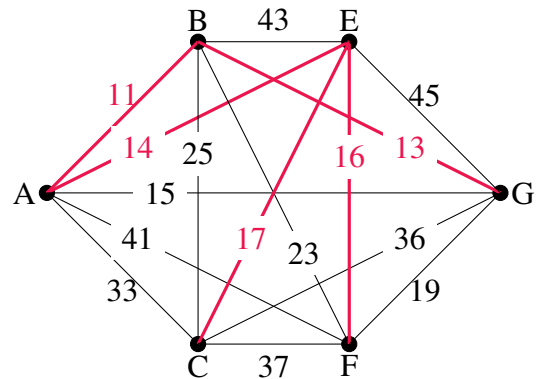
12.7 Prim's Algorithm
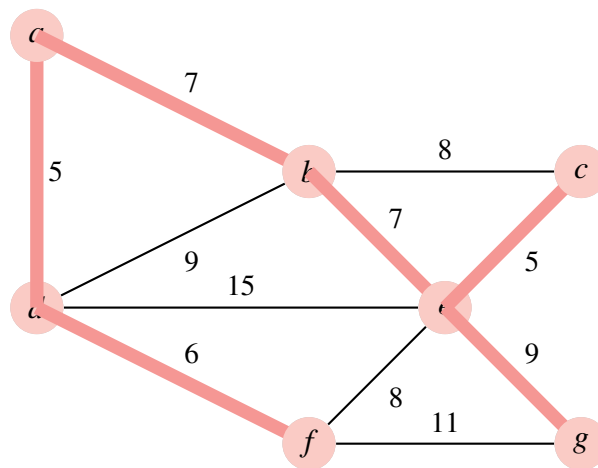
Figure 12.1: Prim's algorithm

12.8 Additional Exercises

Skills

1. To deliver mail in a particular neighborhood, the postal carrier needs to walk along each of the streets with houses (the dots). Create a graph with edges showing where the carrier must walk to deliver the mail.
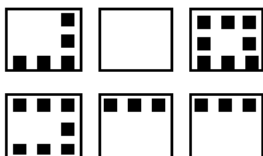
2. Suppose that a town has 7 bridges as pictured below. Create a graph that could be used to determine if there is a path that crosses all bridges once.
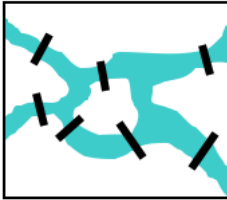
3. The table below shows approximate driving times (in minutes, without traffic) between five cities in the Dallas area. Create a weighted graph representing this data.

	Plano	Mesquite	Arlington	Denton
Fort Worth	54	52	19	42
Plano		38	53	41
Mesquite			43	56
Arlington				50

4. Shown in the table below are the one-way airfares between 5 cities[2]. Create a graph showing this data.

	Honolulu	London	Moscow	Cairo
Seattle	$159	$370	$654	$684
Honolulu		$830	$854	$801
London			$245	$323
Moscow				$329

5. Find the degree of each vertex in the graph below.
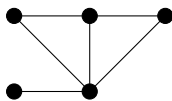
6. Find the degree of each vertex in the graph below.

---
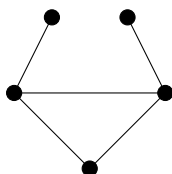
[2]Cheapest fares found when retrieved Sept. 1, 2009 for travel Sept. 22, 2009

7. Which of these graphs are connected?
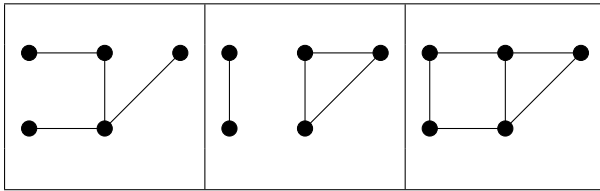
8. Which of these graphs are connected?
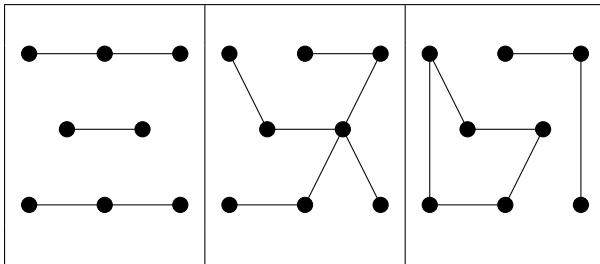
9. Travel times by rail for a segment of the Eurail system is shown below with travel times in hours and minutes. Find path with shortest travel time from Bern to Berlin by applying Dijkstra's algorithm.
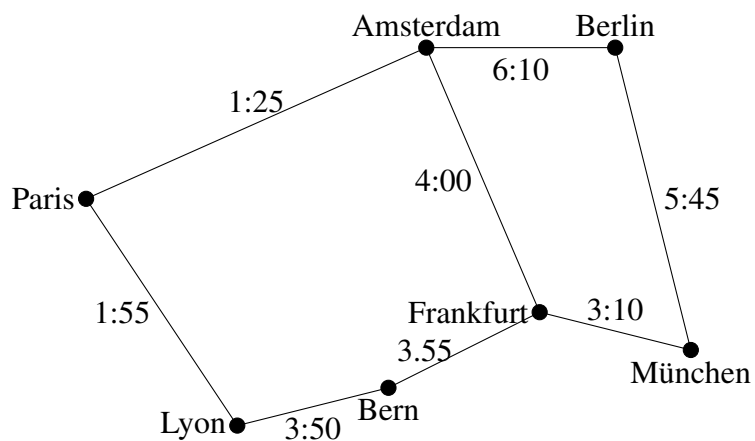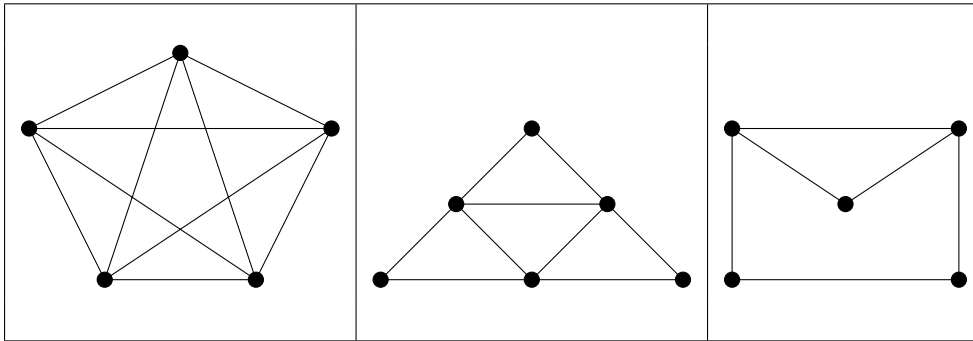

10. Using the graph from the previous problem, find the path with shortest travel time from Paris to München.
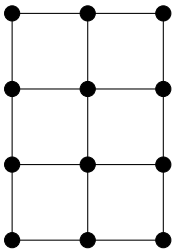
11. Does each of these graphs have an Euler circuit? If so, find it.
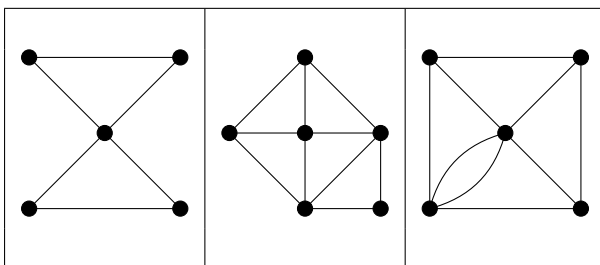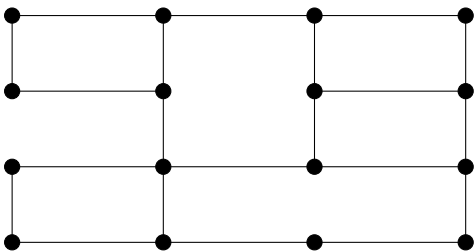
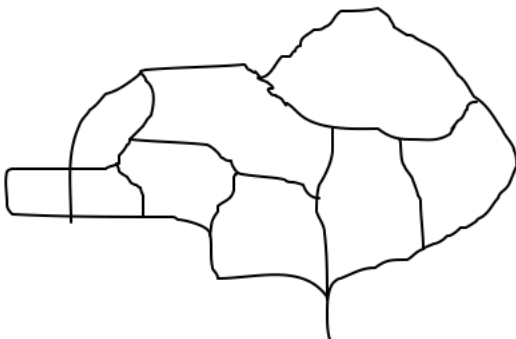12. Does each of these graphs have an Euler circuit? If so, find it.

13. Eulerize this graph using as few edge duplications as possible. Then, find an Euler circuit.


14. Eulerize this graph using as few edge duplications as possible. Then, find an Euler circuit.


15. The maintenance staff at an amusement park need to patrol the major walkways, shown in the graph below, collecting litter. Find an efficient patrol route by finding an Euler circuit. If necessary, eulerize the graph in an efficient way.

16. After a storm, the city crew inspects for trees or brush blocking the road. Find an efficient route for the neighborhood below by finding an Euler circuit. If necessary, eulerize the graph in an efficient way.
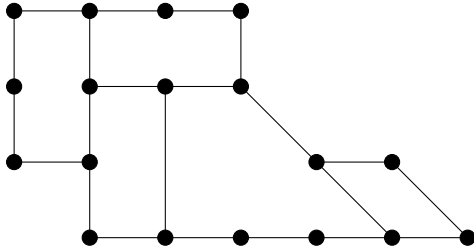
17. Does each of these graphs have at least one Hamiltonian circuit? If so, find one.
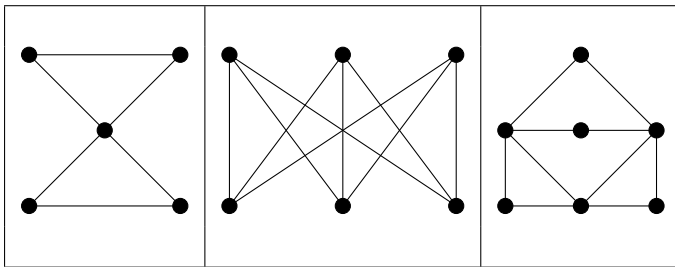
18. Does each of these graphs have at least one Hamiltonian circuit? If so, find one.
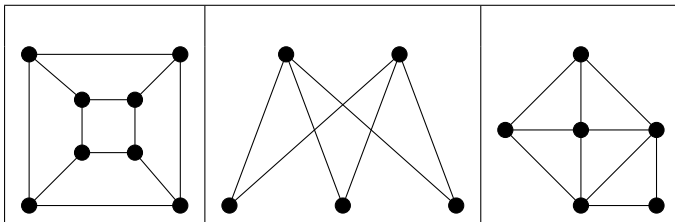
19. A company needs to deliver product to each of their 5 stores around the Dallas, TX area. Driving distances between the stores are shown below. Find a route for the driver to follow, returning to the distribution center in Fort Worth:

    (a) Using Nearest Neighbor starting in Fort Worth

    (b) Using Repeated Nearest Neighbor

    (c) Using Sorted Edges

	Plano	Mesquite	Arlington	Denton
Fort Worth	54	52	19	42
Plano		38	53	41
Mesquite			43	56
Arlington				50

20. A salesperson needs to travel from Seattle to Honolulu, London, Moscow, and Cairo. Use the table of flight costs from problem #4 to find a route for this person to follow:

    (a) Using Nearest Neighbor starting in Seattle

    (b) Using Repeated Nearest Neighbor

    (c) Using Sorted Edges

21. When installing fiber optics, some companies will install a sonet ring; a full loop of cable connecting multiple locations. This is used so that if any part of the cable is damaged it does not interrupt service, since there is a second connection to the hub. A company has 5 buildings. Costs (in thousands of dollars) to lay cables between pairs of buildings are shown below. Find the circuit that will minimize cost:

    (a) Using Nearest Neighbor starting at building A

    (b) Using Repeated Nearest Neighbor

    (c) Using Sorted Edges
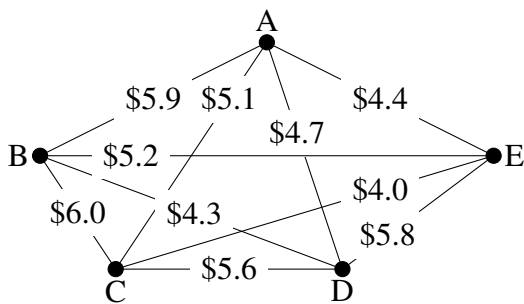
22. A tourist wants to visit 7 cities in Israel. Driving distances, in kilometers, between the cities are shown below[3]. Find a route for the person to follow, returning to the starting city:

    (a) Using Nearest Neighbor starting in Jerusalem

   (b) Using Repeated Nearest Neighbor

    (c) Using Sorted Edges

---

[3]From http://www.ddtravel-acc.com/Israel-cities-distance.htm

|	Jerusalem	Tel Aviv	Haifa	Tiberias	Beer Sheba	Eilat
Jerusalem	–					
Tel Aviv	58	–				
Haifa	151	95	–			
Tiberias	152	134	69	–		
Beer Sheba	81	105	197	233	–	
Eilat	309	346	438	405	241	–
Nazareth	131	102	35	29	207	488

23. Find a minimum cost spanning tree for the graph you created in problem #3.

24. Find a minimum cost spanning tree for the graph you created in problem #22.

25. Find a minimum cost spanning tree for the graph from problem #21.

Concepts

26. Can a graph have one vertex with odd degree? If not, are there other values that are not possible? Why?

27. A complete graph is one in which there is an edge connecting every vertex to every other vertex. For what values of n does complete graph with n vertices have an Euler circuit? A Hamiltonian circuit?

28. Create a graph by drawing n vertices in a row, then another n vertices below those. Draw an edge from each vertex in the top row to every vertex in the bottom row. An example when $n = 3$ is shown below. For what values of n will a graph created this way have an Euler circuit? A Hamiltonian circuit?
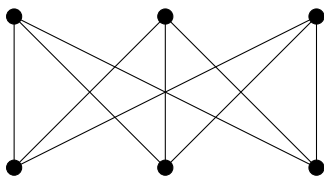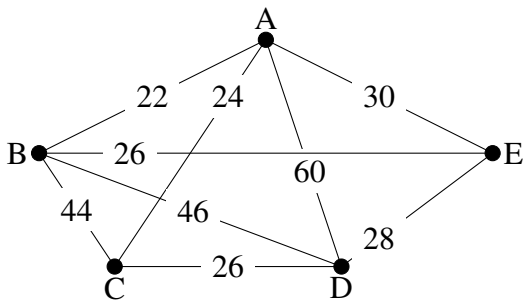

29. Eulerize this graph in the most efficient way possible, considering the weights of the edges.
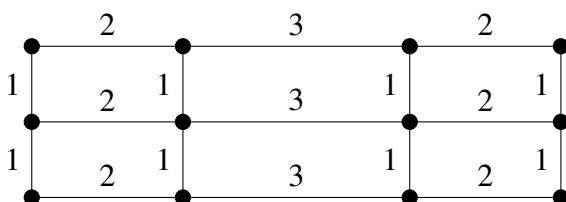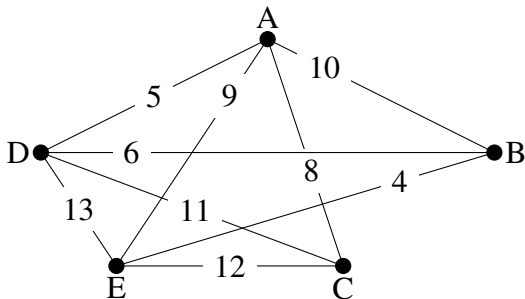
30. Eulerize this graph in the most efficient way possible, considering the weights of the edges.
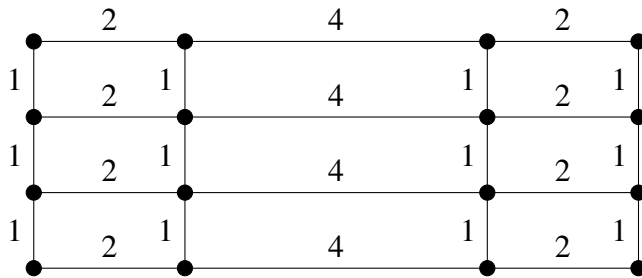
31. Eulerize this graph in the most efficient way possible, considering the weights of the edges.

32. Eulerize this graph in the most efficient way possible, considering the weights of the edges.


Explorations

33. Social networks such as Facebook and LinkedIn can be represented using graphs in which vertices represent people and edges are drawn between two vertices when those people are "friends." The table below shows a friendship table, where an X shows that two people are friends.

	A	B	C	D	E	F	G	H	I
A		X	X			X	X		
B			X		X				
C					X				
D					X				X
E							X		X
F								X	X
G								X	
H									X

(a) Create a graph of this friendship table

(b) Find the shortest path from A to D. The length of this path is often called the "degrees of separation" of the two people.

(c) Extension: Split into groups. Each group will pick 10 or more movies, and look up their major actors (www.imdb.com is a good source). Create a graph with each actor as a vertex, and edges connecting two actors in the same movie (note the movie name on the edge). Find interesting paths between actors, and quiz the other groups to see if they can guess the connections.

34. A spell checker in a word processing program makes suggestions when it finds a word not in the dictionary. To determine what words to suggest, it tries to find similar words. One measure of word similarity is the Levenshtein distance, which measures the number of substitutions, additions, or deletions that are required to change one word into another. For example, the words spit and spot are a distance of 1 apart; changing spit to spot requires one substitution (i for o). Likewise, spit is distance 1 from pit since the change requires one deletion (the s). The word spite is also distance 1 from spit since it requires one addition (the e). The word soot is distance 2 from spit since two substitutions would be required.

   (a) Create a graph using words as vertices, and edges connecting words with a Levenshtein distance of 1. Use the misspelled word "moke" as the center, and try to find at least 10 connected dictionary words. How might a spell checker use this graph?

   (b) Improve the method from above by assigning a weight to each edge based on the likelihood of making the substitution, addition, or deletion. You can base the weights on any reasonable approach: proximity of keys on a keyboard, common language errors, etc. Use Dijkstra?s algorithm to find the length of the shortest path from each word to "moke". How might a spell checker use these values?

35. The graph below contains two vertices of odd degree. To eulerize this graph, it is necessary to duplicate edges connecting those two vertices.

   (a) Use Dijkstra's algorithm to find the shortest path between the two vertices with odd degree. Does this produce the most efficient eulerization and solve the Chinese Postman Problem for

this graph?
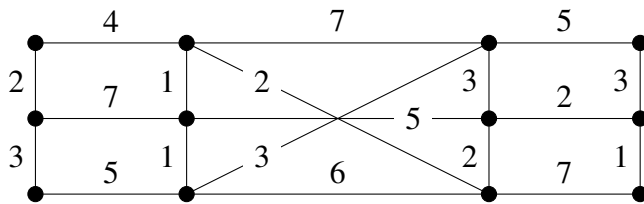
(b) Suppose a graph has n odd vertices. Using the approach from part a, how many shortest paths would need to be considered? Is this approach going to be efficient?

12.8.1. Notes

A paper entitled 'A Note on Two Problems in Connexion with Graphs' was published in the journal 'Numerische Mathematik' in 1959. It was in this paper where the computer scientist named Edsger W. Dijkstra proposed the Dijkstra's Algorithm for the shortest path problem; a fundamental graph theoretic problem. This algorithm can be used to find the shortest path between two nodes or a more common variant of this algorithm is to find the shortest path between a specific 'source' node to any other nodes in the network. https://www.overleaf.com/project/62472837411e2ce1b881337f

Notes, References, and Resources

Resources

Youtube! Video of many graph algorithms by Google engineer (6+ hours)

A. Linear Algebra

A.1 Contributors

lyryx
advancing learning[1]

Champions of Access to Knowledge

OPEN TEXT

All digital forms of access to our high-quality open texts are entirely FREE! All content is reviewed for excellence and is wholly adaptable; custom editions are produced by Lyryx for those adopting Lyryx assessment. Access to the original source files is also open to anyone!

ONLINE ASSESSMENT

We have been developing superior online formative assessment for more than 15 years. Our questions are continuously adapted with the content and reviewed for quality and sound pedagogy. To enhance learning, students receive immediate personalized feedback. Student grade reports and performance statistics are also provided.

SUPPORT

Access to our in-house support team is available 7 days/week to provide prompt resolution to both student and instructor inquiries. In addition, we work one-on-one with instructors to provide a comprehensive system, customized for their course. This can include adapting the text, managing multiple sections, and more!

INSTRUCTOR SUPPLEMENTS

Additional instructor resources are also freely accessible. Product dependent, these supplements include: full sets of adaptable slides and lecture notes, solutions manuals, and multiple choice question banks with an exam building tool.

[1]This book was not produced by Lyryx, but this book has made substantial use of their open source material. We leave this page in here as a tribute to Lyryx for sharing their content.

Contact Lyryx Today!

info@lyryx.com

lyryx
advancing Learning

A First Course in Linear Algebra
an Open Text

BE A CHAMPION OF OER!

Contribute suggestions for improvements, new content, or errata:

A new topic
A new example
An interesting new question
A new or better proof to an existing theorem
Any other suggestions to improve the material

Contact Lyryx at info@lyryx.com with your ideas.

CONTRIBUTIONS

Ilijas Farah, York University

Ken Kuttler, Brigham Young University

Lyryx Learning Team

Foundations of Applied Mathematics

`https://github.com/Foundations-of-Applied-Mathematics`

CONTRIBUTIONS

List of Contributors

E. Evans
Brigham Young University

R. Evans
Brigham Young University

J. Grout
Drake University

J. Humpherys
Brigham Young University

T. Jarvis
Brigham Young University

J. Whitehead
Brigham Young University

J. Adams
Brigham Young University

J. Bejarano
Brigham Young University

Z. Boyd
Brigham Young University

M. Brown
Brigham Young University

A. Carr
Brigham Young University

C. Carter
Brigham Young University

T. Christensen
Brigham Young University

M. Cook
Brigham Young University

R. Dorff
Brigham Young University

B. Ehlert
Brigham Young University

M. Fabiano
Brigham Young University

K. Finlinson
Brigham Young University

J. Fisher
Brigham Young University

R. Flores
Brigham Young University

R. Fowers
Brigham Young University

A. Frandsen
Brigham Young University

R. Fuhriman
Brigham Young University

S. Giddens
Brigham Young University

C. Gigena
Brigham Young University

M. Graham
Brigham Young University

F. Glines
Brigham Young University

C. Glover
Brigham Young University

M. Goodwin
Brigham Young University

R. Grout
Brigham Young University

D. Grundvig
Brigham Young University

E. Hannesson
Brigham Young University

J. Hendricks
Brigham Young University

A. Henriksen
Brigham Young University

I. Henriksen
Brigham Young University

C. Hettinger
Brigham Young University

S. Horst
Brigham Young University

K. Jacobson
Brigham Young University

J. Leete
Brigham Young University

J. Lytle
Brigham Young University

R. McMurray
Brigham Young University

S. McQuarrie
Brigham Young University

D. Miller
Brigham Young University

J. Morrise
Brigham Young University

M. Morrise
Brigham Young University

A. Morrow
Brigham Young University

R. Murray
Brigham Young University

J. Nelson
Brigham Young University

E. Parkinson
Brigham Young University

M. Probst
Brigham Young University

M. Proudfoot
Brigham Young University

D. Reber
Brigham Young University

H. Ringer
Brigham Young University

C. Robertson
Brigham Young University

M. Russell
Brigham Young University

R. Sandberg
Brigham Young University

C. Sawyer
Brigham Young University

M. Stauffer
Brigham Young University

J. Stewart
Brigham Young University

S. Suggs
Brigham Young University

A. Tate
Brigham Young University

T. Thompson
Brigham Young University

M. Victors
Brigham Young University

J. Webb
Brigham Young University

R. Webb
Brigham Young University

J. West
Brigham Young University

A. Zaitzeff
Brigham Young University

A.1.1. Graph Theory

Chapter on Graph Theory adapted from: CC-BY-SA 3.0 Math in Society A survey of mathematics for the liberal arts major Math in Society is a free, open textbook. This book is a survey of contemporary mathematical topics, most non-algebraic, appropriate for a college-level quantitative literacy topics course for liberal arts majors. The text is designed so that most chapters are independent, allowing the instructor to choose a selection of topics to be covered. Emphasis is placed on the applicability of the mathematics. Core material for each topic is covered in the main text, with additional depth available through exploration exercises appropriate for in-class, group, or individual investigation. This book is appropriate for Washington State Community Colleges' Math 107.

The current version is 2.5, released Dec 2017. `http://www.opentextbookstore.com/mathinsociety/2.5/GraphTheory.pdf`

Communicated by Tricia Muldoon Brown, Ph.D. Associate Professor of Mathematics Georgia Southern University Armstrong Campus Savannah, GA 31419`http://math.armstrong.edu/faculty/brown/MATH1001.html`