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: August 20, 2023

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 LaTeXusing 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,

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

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, π, e, $-11/5$)
- $\setminus$ - setminus, (e.g. $\{0,1,2,3\} \setminus \{0,3\} = \{1,2\}$)
- $\cup$ - union (e.g. $\{1,2\} \cup \{3,5\} = \{1,2,3,5\}$
- $\cap$ - intersection (e.g. $\{1,2,3,4\} \cap \{3,4,5,6\} = \{3,4\}$)
- $\{0,1\}^4$ - the set of 4 dimensional vectors taking values 0 or 1, (e.g. $[0,0,1,0]$ or $[1,1,1,1]$)
- $\mathbb{Z}^4$ - the set of 4 dimensional vectors taking integer values (e.g., $[1,-5,17,3]$ or $[6,2,-3,-11]$)

- $\mathbb{Q}^4$ - the set of 4 dimensional vectors taking rational values (e.g. $[1.5, 3.4, -2.4, 2]$)
- $\mathbb{R}^4$ - the set of 4 dimensional vectors taking real values (e.g. $[3, \pi, -e, \sqrt{2}]$)
- $\sum_{i=1}^{4} i = 1+2+3+4$
- $\sum_{i=1}^{4} i^2 = 1^2+2^2+3^2+4^4$
- $\sum_{i=1}^{4} x_i = x_1+x_2+x_3+x_4$
- $\square$ - this is a typical Q.E.D. symbol that you put at the end of a proof meaning "I proved it."
- For $x, y \in \mathbb{R}^3$, the following are equivalent (note, in other contexts, these notations can mean different things)
  - $x^\top y$ *matrix multiplication*
  - $x \cdot y$ *dot product*
  - $\langle x, y \rangle$ *inner product*

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

A sample sentence:

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

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

2. Mathematical Programming

Outcomes

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).

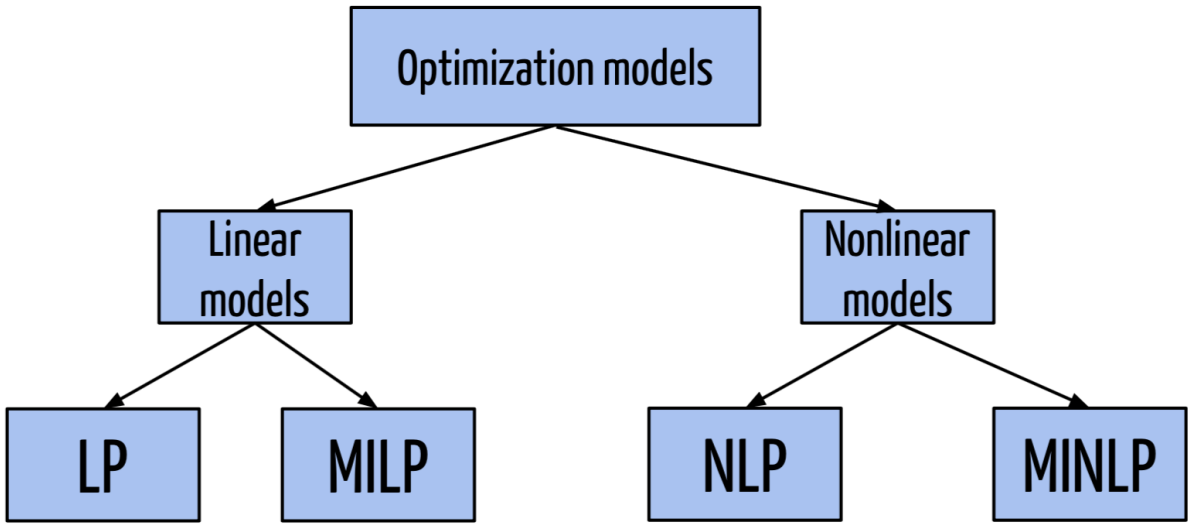


Figure 2.1: ***Tree of optimization problems.***

Along with each problem class, we will associate a complexity class for the general version of the problem. See chapter 5 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.

2.1 Linear Programming (LP)

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

Linear Programming (LP):

Polynomial time (P)

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

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

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

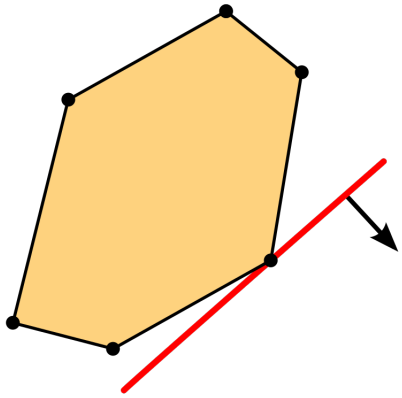
$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned} \tag{2.2}$$

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] *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.



© Ylloh CC0[2]

Figure 2.2: Linear programming constraints and objective.

2.2 Mixed-Integer Linear Programming (MILP)

Mixed-integer linear programming will be the focus of Sections 3, 4, 6, and **??**. Recall that the notation $\mathbb{Z}$ means the set of integers and the set $\mathbb{R}$ means the set of real numbers. The first problem of interest here is a *binary integer program* (BIP) where all n variables are binary (either 0 or 1).

Binary Integer programming (BIP):

NP-Complete

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

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

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

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.

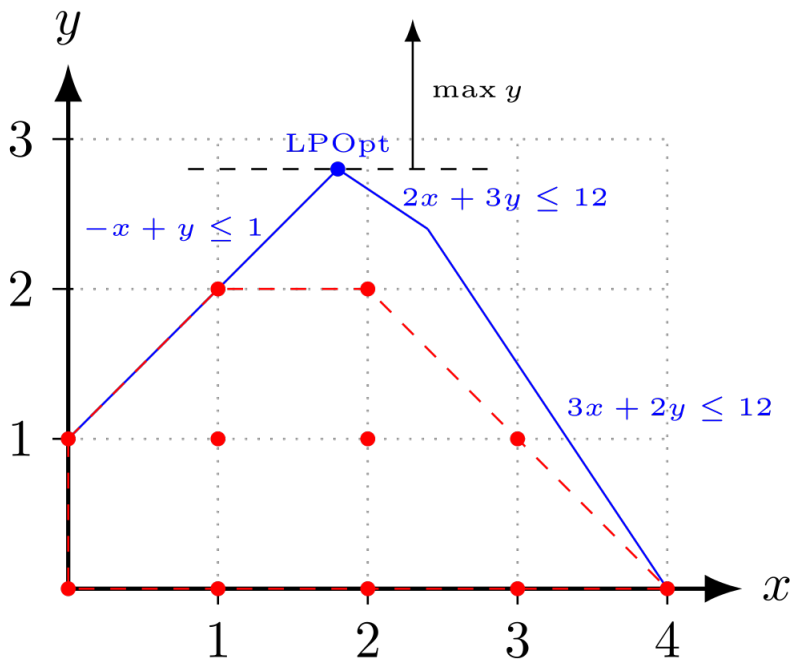


Figure 2.3: Comparing the LP relaxation to the IP solutions.

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 program-*

ming problem is

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

2.3 Non-Linear Programming (NLP)

NLP:

NP-Hard

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

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

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.

2.3.1. Convex Programming

Here the functions are all **convex!**

Convex Programming:

Polynomial time (P) (typically)

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

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

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

2.3.2. Non-Convex Non-linear Programming

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

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

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

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

Binary Integer programming (BIP) as a NLP:

NP-Hard

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

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

2.3.3. 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.

2.4 Mixed-Integer Non-Linear Programming (MINLP)

2.4.1. Convex Mixed-Integer Non-Linear Programming

2.4.2. Non-Convex Mixed-Integer Non-Linear Programming

Part I

Linear Programming

Part II

Integer Programming

3. Integer Programming Formulations

Outcomes

A. *Learn classic integer programming formulations.*
B. *Demonstrate different uses of binary and integer variables.*
C. *Demonstrate the format for modeling an optimization problem with sets, parameters, variables, and the model.*

In this section, we will describe classical integer programming formulations. These formulations may reflect a real world problem exactly, or may be part of the setup of a real world problem.

3.1 Knapsack Problem

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

Binary Knapsack Problem:

NP-Complete

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

$$\begin{aligned} \max \ & c^\top x \\ \text{s.t.} \ & a^\top x \leq b \\ & x \in \{0,1\}^n \end{aligned} \tag{3.1}$$

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

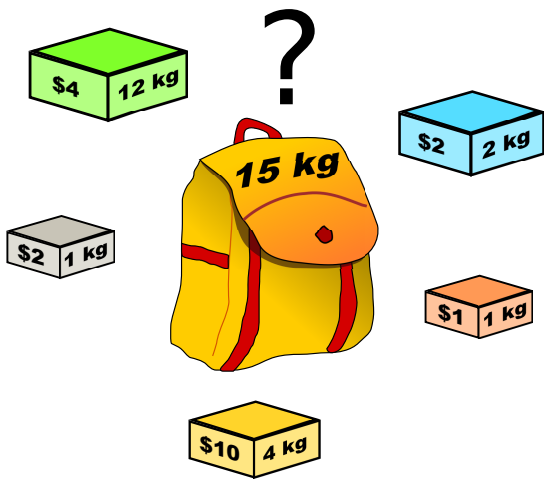


Figure 3.1: Knapsack Problem: which items should we choose take in the knapsack that maximizes the value while respecting the 15kg weight limit?

Example: Knapsack Gurobipy

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

Variables:

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

Model:

$$\begin{aligned} \max \quad & 4x_1 + 2x_2 + 2x_3 + 1x_4 + 10x_5 && \text{(Total value)} \\ \text{s.t.} \quad & 12x_1 + 2x_2 + 1x_3 + 1x_4 + 4x_5 \leq 15 && \text{(Capacity bound)} \\ & x_i \in \{0,1\} \text{ for } i = 1,\ldots,5 && \text{(Item taken or not)} \end{aligned}$$

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

Integer Knapsack Problem:

NP-Complete

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

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & a^\top x \leq b \\ & x \in \mathbb{Z}_+^n \end{aligned} \tag{3.2}$$

We can also consider an equality constrained version

Equality Constrained Integer Knapsack Problem:

NP-Hard

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

$$\max \quad c^\top x \tag{3.3}$$
$$\text{s.t.} \quad a^\top x = b \tag{3.4}$$
$$x \in \mathbb{Z}_+^n \tag{3.5}$$

Example: Min Coins Gurobipy

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

Variables:

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

Model

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

3.2 Capital Budgeting

The *capital budgeting* problem is a nice generalization of the knapsack problem. This problem as the same structure as the knapsack problem, except now it has multiple constraints. We will first describe the problem, give a general model, and then look at an explicit example.

Capital Budgeting:

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

- Project j expects to produce revenue c_j dollars overall.
- Project j requires investment of a_{ij} dollars in time period i for $i = 1,\ldots,m$.
- The capital available to spend in time period i is b_i.

Which projects should the firm invest in to maximize it's expected return while satisfying it's weekly budget constriaints?

We will first provide a general formulation for this problem.

Capital Budgeting Model:

Sets:

- Let $I = \{1, \ldots, m\}$ be the set of time periods.
- Let $J = \{1, \ldots, n\}$ be the set of possible investments.

Parameters:

- c_j is the expected revenue of investment j for $j \in J$
- b_i is the available capital in time period i for i in I
- a_{ij} is the resources required for invesment j in time period i, for i in I, for j in J.

Variables:

- let $x_i = 0$ if investment i is chosen
- let $x_i = 1$ if investment i is not chosen

Model:

$$\max \quad \sum_{j=1}^{n} c_j x_j \qquad \text{(Total Expected Revenue)}$$

$$s.t. \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i = 1, \ldots m \qquad \text{(Resource constraint week } i\text{)}$$

$$x_j \in \{0, 1\}, \; j = 1, \ldots, n$$

Consider the example given in the following table.

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

Given this data, we can setup our problem explicitly as follows

---

Example: Capital Budgeting Gurobipy

Sets:

- Let $I = \{1,2\}$ be the set of time periods.
- Let $J = \{1,2,3\}$ be the set of possible investments.

Parameters:

- c_j is given in column "$\mathbb{E}$[Revenue]".
- b_i is given in row "Resources available".
- a_{ij} given in row j, and column for week i.

Variables:

- let $x_i = 0$ if investment i is chosen
- let $x_i = 1$ if investment i is not chosen

The explict model is given by
Model:

$$\begin{aligned} \max \quad & 10x_1 + 8x_2 + 6x_3 && \text{(Total Expected Revenue)} \\ s.t. \quad & 3x_1 + 1x_2 + 2x_3 \leq 5 && \text{(Resource constraint week 1)} \\ & 4x_1 + 2x_2 + 1x_3 \leq 6 && \text{(Resource constraint week 2)} \\ & x_j \in \{0,1\},\ j = 1,2,3 \end{aligned}$$

3.3 Set Covering

The *set covering* problem can be used for a wide array of problems. We will see several examples in this section.

Set Covering:

NP-Complete

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

The set cover problem can be modeled as

$$\begin{aligned} \min \quad & 1^\top x \\ \text{s.t.} \quad & \sum_{v \in V_i} x_v \geq 1 \text{ for all } i = 1, \ldots, l \\ & x_v \in \{0,1\} \text{ for all } v \in V \end{aligned} \tag{3.1}$$

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

Example: Capital Budgeting Gurobipy

Sets:

- Let $I = \{1,2\}$ be the set of time periods.
- Let $J = \{1,2,3\}$ be the set of possible investments.

Parameters:

- c_j is given in column "$\mathbb{E}$[Revenue]".
- b_i is given in row "Resources available".
- a_{ij} given in row j, and column for week i.

Variables:

- let $x_i = 0$ if investment i is chosen
- let $x_i = 1$ if investment i is not chosen

The explict model is given by
Model:

$$\begin{aligned} \max \quad & 10x_1 + 8x_2 + 6x_3 && \text{(Total Expected Revenue)} \\ s.t. \quad & 3x_1 + 1x_2 + 2x_3 \leq 5 && \text{(Resource constraint week 1)} \\ & 4x_1 + 2x_2 + 1x_3 \leq 6 && \text{(Resource constraint week 2)} \\ & x_j \in \{0,1\},\ j = 1,2,3 \end{aligned}$$

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

Example: Vertex Cover:

NP-Complete

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

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

$$\begin{aligned} \min \quad & 1^\top x \\ \text{s.t.} \quad & x_u + x_v \geq 1 \text{ for all } (u,v) \in E \\ & x_v \in \{0,1\} \text{ for all } v \in V. \end{aligned} \tag{3.2}$$

Example: Set cover: Fire station placement Gurobipy

In the fire station problem, we seek to choose locations for fire stations such that any district either contains a fire station, or neighbors a district that constains a fire station. Figure 3.2 depicts the set of districts and an example placement of locations of fire stations. How can we minimize the the total number of fire stations that we need?

Sets:

- Let V be the set of districts ($V = \{1,\ldots,16\}$)
- Let V_i be the set of districts that neighbor district i (e.g. $V_1 = \{2,4,5\}$).

Variables:

- let $x_i = 1$ if district i is chosen to have a fire station.
- let $x_i = 0$ otherwise.

Model:

$$\begin{aligned} \min \quad & \sum_{i \in V} x_i && && (\#\text{ open fire stations}) \\ \text{s.t.} \quad & x_i + \sum_{j \in V_i} x_j \geq 1 && \forall i \in V && (\text{Station proximity requirement}) \\ & x_i \in \{0,1\} && \text{for } i \in V && (\text{station either open or closed}) \end{aligned}$$

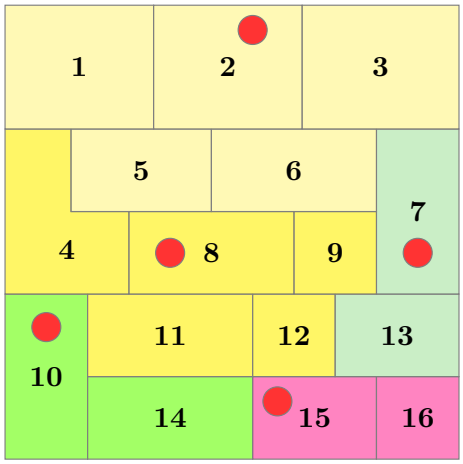


Figure 3.2: Layout of districts and possible locations of fire stations.

Set Covering - Matrix description:

NP-Complete

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

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t..} \quad & Ax \geq 1 \\ & x \in \{0,1\}^n. \end{aligned} \tag{3.3}$$

[2] *Fire station district layout*, from https://github.com/open-optimization/open-optimization-or-book/tree/master/content/figures/figures-source/tikz/Illustration2.tex. © Robert Hildebrand CC BY-SA 4.0., 2020.

[3] *Set cover representation of fire station problem*, from https://github.com/open-optimization/open-optimization-or-book/tree/master/content/figures/figures-source/tikz/Illustration2.tex. © Robert Hildebrand CC BY-SA 4.0., 2020.

[4] *Graph representation of fire station problem*, from https://github.com/open-optimization/open-optimization-or-book/tree/master/content/figures/figures-source/tikz/Illustration3.tex. © Robert Hildebrand CC BY-SA 4.0., 2020.

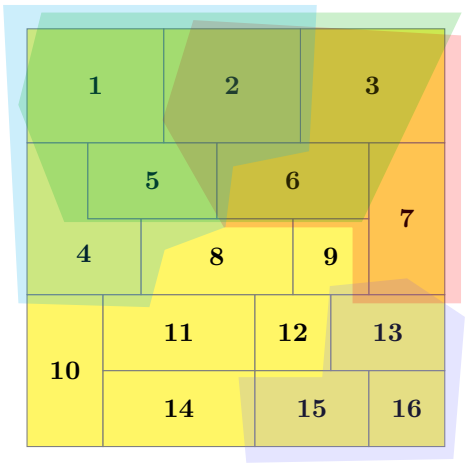


Figure 3.3: Set cover representation of fire station problem. For example, choosing district 16 to have a fire station covers districts 13, 15, and 16.

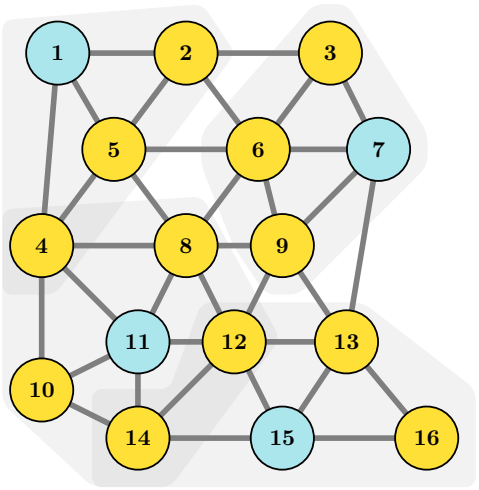


Figure 3.4: Graph representation of fire station problem. Every node is connected to a chosen node by an edge

Example: Vertex Cover with matrix

An alternate way to solve Example: Vertex Cover is to define the *adjacency matrix* A of the graph. The adjacency matrix is a $|E| \times |V|$ matrix with $\{0,1\}$ entries. The each row corresponds to an edge e and each column corresponds to a node v. For an edge $e = (u,v)$, the corresponding row has a 1 in

columns corresponding to the nodes u and v, and a 0 everywhere else. Hence, there are exactly two 1's per row. Applying the formulation above in Set Covering - Matrix description models the problem.

3.3.1. Covering (Generalizing Set Cover)

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

Covering:

NP-Complete

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

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t..} \quad & Ax \geq b \\ & x \in \mathbb{Z}_+^n. \end{aligned} \tag{3.4}$$

3.4 Assignment Problem

The *assignment problem* (machine/person to job/task assignment) seeks to assign tasks to machines in a way that is most efficient. This problem can be thought of as having a set of machines that can complete various tasks (textile machines that can make t-shirts, pants, socks, etc) that require different amounts of time to complete each task, and given a demand, you need to decide how to alloacte your machines to tasks.

Alternatively, you could be an employer with a set of jobs to complete and a list of employees to assign to these jobs. Each employee has various abilities, and hence, can complete jobs in differing amounts of time. And each employee's time might cost a different amout. How should you assign your employees to jobs in order to minimize your total costs?

Assignment Problem:

Given m machines and n jobs, find a least cost assignment of jobs to machines. The cost of assigning job j to machine i is c_{ij}.

Example: Machine Assignment Gurobipy

Sets:

- Let $I = \{0,1,2,3\}$ set of machines.
- Let $J = \{0,1,2,3\}$ be the set of tasks.

Parameters:

- c_{ij} - the cost of assigning machine i to job j

Variables:

- Let

$$x_{ij} = \begin{cases} 1 & \text{if machine } i \text{ assigned to job } j \\ 0 & \text{otherwise.} \end{cases}$$

Model:

$$\begin{aligned}
\min \quad & \sum_{i \in I, j \in J} c_{ij} x_{ij} && && \text{(Minimize cost)} \\
s.t. \quad & \sum_{i \in I} x_{ij} = 1 && \text{for all } j \in J && \text{(All jobs are assigned one machine)} \\
& \sum_{j \in J} x_{ij} = 1 && \text{for all } i \in I && \text{(All machines are assigned to a job)} \\
& x_{ij} \in \{0,1\} \forall i \in I, j \in J
\end{aligned}$$

Example: School Bus Routing Problem Gurobipy

A school district has a set of schools I and a fleet of school buses J. Each bus needs to be assigned to a school every morning to pick up students. The cost c_{ij} represents the fuel cost for bus j to reach school i and complete the route. The district aims to minimize the total fuel cost while ensuring that each school is served by exactly one bus and each bus is assigned to exactly one school.

The fuel costs can be found in the following table. They are also in csv file format here.

Bus/School	School A	School B	School C	School D	School E
Bus 1	50	80	60	90	100
Bus 2	60	85	55	70	110
Bus 3	75	65	50	85	90
Bus 4	70	90	55	80	95
Bus 5	80	70	60	75	85

We can model this as follows:

Indices:

- i - Index for schools, $i \in I$
- j - Index for buses, $j \in J$

Parameters:

- c_{ij} - Fuel cost for bus j to serve school i.

Decision Variables:

- x_{ij} - 1 if bus j is assigned to school i, 0 otherwise.

Model:

$$\min \sum_{i \in I, j \in J} c_{ij} x_{ij} \qquad \text{(Minimize total fuel cost)}$$

$$s.t. \sum_{i \in I} x_{ij} = 1 \quad \text{for all } j \in J \qquad \text{(Each bus is assigned to one school)}$$

$$\sum_{j \in J} x_{ij} = 1 \quad \text{for all } i \in I \qquad \text{(Each school is served by one bus)}$$

$$x_{ij} \in \{0,1\} \forall i \in I, j \in J$$

3.5 Facility Location

The basic model of the facility location problem is to determine where to place your stores or facilities in order to be close to all of your customers and hence reduce the costs transportation to your customers. Each customer is known to have a certain demand for a product, and each facility has a capacity on how much of that demand it can satisfy. Furthermore, we need to consider the cost of building the facility in a given location.

This basic framework can be applied in many types of problems and there are a number of variants to this problem. We will address two variants: the *capacitated facility location problem* and the *uncapacitated facility location problem*.

3.5.1. Capacitated Facility Location

Capacitated Facility Location:

NP-Complete

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

Sets:

- Let $I = \{1,\ldots,n\}$ be the set of facilities.
- Let $J = \{1,\ldots,m\}$ be the set of customers.

Parameters:

- f_i - the cost of opening facility i.
- c_{ij} - the cost of fulfilling the complete demand of customer j from facility i.
- u_i - the capacity of facility i.
- d_j - the demand by customer j.

Variables:

- Let
$$y_i = \begin{cases} 1 & \text{if we open facility } i, \\ 0 & \text{otherwise.} \end{cases}$$
- Let $x_{ij} \geq 0$ be the fraction of demand of customer j satisfied by facility i.

Model:

$$\min \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} + \sum_{i=1}^{n} f_i y_i \qquad \text{(total cost)}$$

$$\text{s.t.} \sum_{i=1}^{n} x_{ij} = 1 \text{ for all } j = 1,\ldots,m \qquad \text{(assign demand to facility)}$$

$$\sum_{j=1}^{m} d_j x_{ij} \leqslant u_i y_i \text{ for all } i = 1\ldots,n \qquad \text{(capacity of facility } i\text{)}$$

$$x_{ij} \geqslant 0 \text{ for all } i = 1,\ldots,n \text{ and } j = 1,\ldots,m \qquad \text{(nonnegative fraction of demand satisfied)}$$

$$y_i \in \{0,1\} \text{ for all } i = 1,\ldots,n \qquad \text{(open/not open facility)}$$

Alternative model!

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{i=1}^{m} f_i y_i \\
s.t. & \sum_{i=1}^{m} x_{ij} = 1, \quad j = 1,\dots n \\
& \sum_{j=1}^{m} d_j x_{ij} \leqslant u_i \text{ for all } i = 1\dots,n \qquad \text{(capacity of facility } i\text{)} \\
& x_{ij} \leq y_i, \quad i = 1,\dots m, \quad j = 1,\dots,n \\
& x_{ij} \geq 0 \quad i = 1,\dots,m, \quad j = 1,\dots,n \\
& y_i \in \{0,1\}, \quad i = 1,\dots,m
\end{aligned}
$$

Example: Capacitated Facility Location: Retail Distribution Example Gurobipy

Context: A retail company plans to establish distribution centers across a country to serve its stores efficiently. The company faces decisions on which distribution centers to open and which stores each center should serve. Costs associated with opening each center and serving stores from them are known, and each center has a maximum capacity. Additionally, each store has a known demand.

*Given Data: (distribution-data.xlsx)

- Number of potential distribution centers (n): 3
- Number of stores (m): 4

Costs to Open Distribution Centers (f_i):

- Distribution Center 1: \$150,000
- Distribution Center 2: \$100,000
- Distribution Center 3: \$180,000

Cost to Serve a Store from a Distribution Center (c_{ij}):

	Store 1	Store 2	Store 3	Store 4
Center 1	\$50	\$60	\$70	\$85
Center 2	\$75	\$45	\$55	\$50
Center 3	\$65	\$80	\$40	\$60

Capacity of Distribution Centers (u_i):

- Distribution Center 1: 100 units
- Distribution Center 2: 80 units

- Distribution Center 3: 90 units

Demand by Each Store (d_j):

- Store 1: 40 units
- Store 2: 30 units
- Store 3: 20 units
- Store 4: 25 units

*Model Formulation: The capacitated facility location model described earlier can be applied to this scenario with the given data to determine the optimal number and location of distribution centers to open, and which stores each center should serve.

3.5.2. Uncapacitated Facility Location

Uncapacitated Facility Location:

NP-Complete

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

$$\begin{aligned} \min \quad & \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}z_{ij} + \sum_{i=1}^{n} f_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} z_{ij} = 1 \text{ for all } j = 1,\ldots,m \\ & \sum_{j=1}^{m} z_{ij} \leqslant M x_i \text{ for all } i = 1\ldots,n \\ & z_{ij} \in \{0,1\} \text{ for all } i = 1,\ldots,n \text{ and } j = 1,\ldots,m \\ & x_i \in \{0,1\} \text{ for all } i = 1,\ldots,n \end{aligned} \tag{3.1}$$

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

Alternative model!

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} z_{ij} + \sum_{i=1}^{n} f_i x_i \\
\text{s.t.} \quad & \sum_{i=1}^{n} z_{ij} = 1 \text{ for all } j = 1, \ldots, m \\
& z_{ij} \leqslant x_i \text{ for all } i = 1 \ldots, n \text{ for all } j = 1, \ldots, m \\
& z_{ij} \in \{0, 1\} \text{ for all } i = 1, \ldots, n \text{ and } j = 1, \ldots, m \\
& x_i \in \{0, 1\} \text{ for all } i = 1, \ldots, n
\end{aligned}
\tag{3.2}
$$

3.6 Basic Modeling Tricks - Using Binary Variables

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

Binary variables can allow you to model many types of constraints. We discuss here varios logical constraints where we assume that $x_i \in \{0,1\}$ for $i = 1,\ldots,n$. We will take the meaning of the variable to be selecting an item.

1. If item i is selected, then item j is also selected.

$$x_i \leq x_j \tag{3.1}$$

   (a) If any of items $1,\ldots,5$ are selected, then item 6 is selected.

$$x_1 + x_2 + \cdots + x_5 \leq 5 \cdot x_6 \tag{3.2}$$

   Alternatively!

$$x_i \leq x_6 \quad \text{for all } i = 1,\ldots,5 \tag{3.3}$$

2. If item j is not selected, then item i is not selected.

$$x_i \leq x_j \tag{3.4}$$

   (a) If item j is not selected, then all items $1,\ldots,i$ are not selected.

$$x_1 + x_2 + \cdots + x_i \leq i \cdot x_j \tag{3.5}$$

3. If item j is not selected, then item i is not selected.

$$x_i \leq x_j \tag{3.6}$$

4. Either item i is selected or item j is selected, but not both.

$$x_i + x_j = 1 \tag{3.7}$$

5. Item i is selected or item j is selected or both.

$$x_i + x_j \geq 1 \tag{3.8}$$

6. If item i is selected, then item j is not selected.

$$x_j \leq (1 - x_i) \tag{3.9}$$

7. At most one of items i, j, and k are selected.

$$x_i + x_j + x_k \leq 1 \tag{3.10}$$

8. At most two of items i, j, and k are selected.

$$x_i + x_j + x_k \leq 2 \tag{3.11}$$

9. Exactly one of items i, j, and k are selected.

$$x_i + x_j + x_k = 1 \tag{3.12}$$

These tricks can be connected to create different function values.

Example 3.1: Variable takes one of three values

Suppose that the variable x should take one of the three values $\{4, 8, 13\}$. This can be modeled using three binary variables as

$$\begin{aligned} x &= 4z_1 + 8z_2 + 13z_3 \\ z_1 + z_2 + z_3 &= 1 \\ z_i \in \{0,1\} &\text{ for } i = 1,2,3. \end{aligned}$$

As a convenient addition, if we want to add the possibility that it takes the value 0, then we can model this as

$$\begin{aligned} x &= 4z_1 + 8z_2 + 13z_3 \\ z_1 + z_2 + z_3 &\leq 1 \\ z_i \in \{0,1\} &\text{ for } i = 1,2,3. \end{aligned}$$

We can also model variable increases at different amounts.

Example 3.2: Discount for buying more

Suppose you can choose to buy 1, 2, or 3 units of a product, each with a decreasing cost. The first unit is \$10, the second is \$5, and the third unit is \$3.

$$\begin{aligned} x &= 10z_1 + 5z_2 + 3z_3 \\ z_1 &\geq z_2 \geq z_3 \\ z_i \in \{0,1\} &\text{ for } i = 1,2,3. \end{aligned}$$

Here, z_i represents if we buy the ith unit. The inequality constraints impose that if we buy unit j, then we must by all units i with $i < j$.

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

3.6.1. Big M constraints - Activating/Deactivating Inequalities

Big M comes again! It's extremely useful when trying to activate constrants based on a binary variable.

For intance, if we don't rent a bus, then we can have at most 3 passengers join us on our trip. Consider passengers A,B,C,D,E and let $x_i \in \{0,1\}$ be 1 if we take passenger i and 0 otherwise. We can model the constraint that we can have at most 5 passengers as

$$x_A + x_B + x_C + x_D + x_E \leq 3.$$

We want to be able to activate this constraint in the event that we don't rent a bus.

Let $\delta \in \{0,1\}$ be 1 if rent a bus, and 0 otherwise.

Then we want to say

If $\delta = 0$, then

$$x_A + x_B + x_C + x_D + x_E \leq 3.$$

We can formulate this using a big-M constraint as

$$x_A + x_B + x_C + x_D + x_E \leq 3 + M\delta. \tag{3.13}$$

Notice the two case

$$\begin{cases} x_A + x_B + x_C + x_D + x_E \leq 3 & \text{if } \delta = 0 \\ x_A + x_B + x_C + x_D + x_E \leq 3 + M & \text{if } \delta = 1 \end{cases}$$

In the second case, we choose M to be so large, that the second case inequality is vacuous. That said, choosing smaller M values (that are still valid) will help the computer program solve the problem faster. In this case, it suffices to let $M = 2$.

We can speak about this technique more generally as

Big-M: If then:

We aim to model the relationship

$$\text{If} \quad \delta = 0, \quad \text{then} \quad a^\top x \leq b. \tag{3.14}$$

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

$$\begin{aligned} a^\top x - b &\leq M\delta \\ \delta &\in \{0,1\} \end{aligned} \tag{3.15}$$

3.6.2. Either Or Constraints

"At least one of these constraints holds" is what we would like to model. Equivalently, we can phrase this as an *inclusive or* constraint. This can be molded with a pair of Big-M constraints.

Either Or:

$$\text{Either} \quad a^\top x \leq b \quad \text{or} \quad c^\top x \leq d \quad \text{holds} \tag{3.16}$$

can be modeled as

$$\begin{aligned} a^\top x - b &\leq M_1 \delta \\ c^\top x - d &\leq M_2(1-\delta) \\ \delta &\in \{0,1\}, \end{aligned} \tag{3.17}$$

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

Example 3.3

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

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

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

This constraint can be modeled as

$$\begin{aligned} x - 2 &\leq 5\delta \\ y - 10 &\leq 20(1-\delta) \\ \delta &\in \{0,1\}, \end{aligned} \tag{3.18}$$

3.6.3. If then implications - opposite direction

Suppose that we want to model the fact that if we have at most 10 students attending this course, then we must switch to a smaller classroom.

Let $x_i \in \{0,1\}$ be 1 if student i is in the course or not. Let $\delta \in \{0,1\}$ be 1 if we need to switch to a smaller classroom.

Thus, we want to model

If

$$\sum_{i \in I} x_i \leq 10$$

then

$$\delta = 1.$$

We can model this as

$$\sum_{i \in I} x_i \geq 10 + 1 + M\delta. \tag{3.19}$$

If inequality, then indicator:

W

e let m be a lower bound on the quantity $a^\top x - b$ and we let ε be a tiny number that is an error bound in verifying if an inequality is violated. **If the data a, b are integer and x is an integer, then we can take $\varepsilon = 1$.**

Now

$$\text{If} \quad a^\top x \leq b \quad \text{then} \quad \delta = 1 \tag{3.20}$$

can be modeled as

$$a^\top x - b \geq \varepsilon(1 - \delta) + m\delta. \tag{3.21}$$

Proof. We not justify the statement above.

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

$$\text{If } \delta = 0, \text{ then } a^\top x - b > 0. \tag{3.22}$$

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

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

Thus, the contrapositive holds.

If instead we wanted a direct proof:

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

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

Therefore

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

After rearranging

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

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

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

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

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

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

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

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

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

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

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

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

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

Lastly, if you insist on having exact correspondance, that is, "$\delta = 0$ if and only if $a^\top x \leq b$" you can simply include both constraints for "if $\delta = 0$, then $a^\top x \leq b$" and if "$a^\top x \leq b$, then $\delta = 0$". Although many problems may be phrased in a way that suggests you need "if and only if", it is often not necessary to use both constraints due to the objectives in the problem that naturally prevent one of these from happening.

For example, if we want to add a binary variable δ that means

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

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

3.6.4. Multi Term Disjunction with application to 2D packing

A disjunction is a generalization of an "or" statement. Suppose that we have n constraints

$$\mathbf{a}_1^\top \mathbf{x} \leq b_1, \quad \mathbf{a}_2^\top \mathbf{x} \leq b_2, \quad \ldots, \quad \mathbf{a}_n^\top \mathbf{x} \leq b_n$$

and we want to enforce at least k of them. This can be accomplished linearly by introducing a new binary indicator variable δ_i for each of the disjunctive constraints $i \in \{1, \ldots, n\}$:

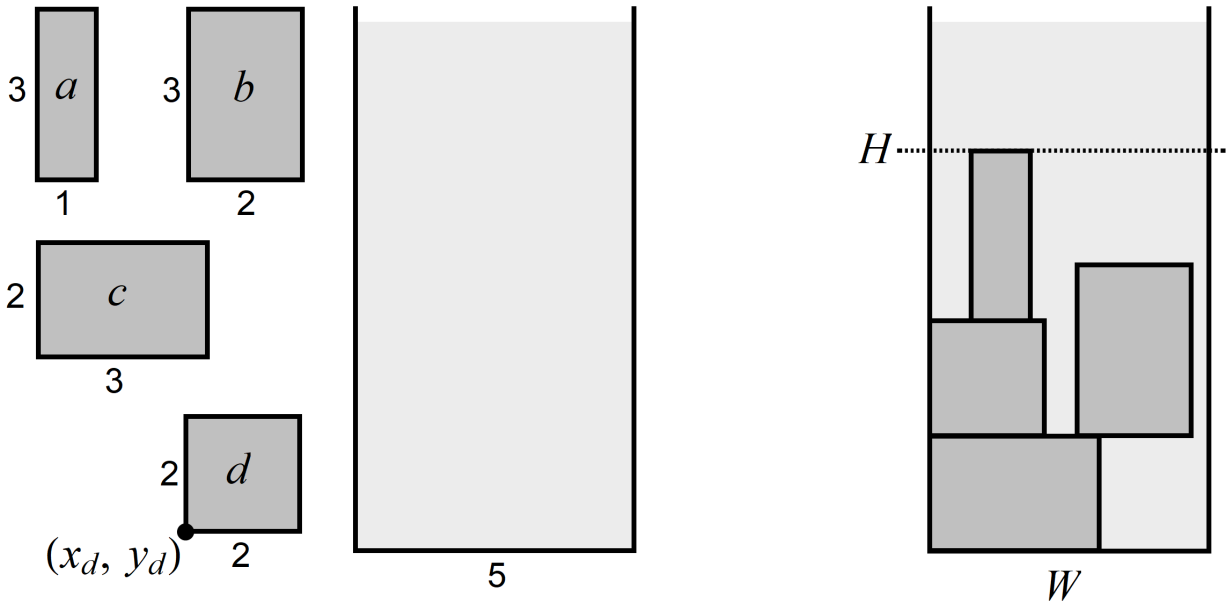
$$\begin{aligned} \mathbf{a}_1^\top \mathbf{x} &\leq b_1 + M(1 - \delta_1) \\ \mathbf{a}_2^\top \mathbf{x} &\leq b_2 + M(1 - \delta_2) \\ &\vdots \\ \mathbf{a}_n^\top \mathbf{x} &\leq b_n + M(1 - \delta_1) \\ \sum_{i=1}^{n} \delta_i &\geq k \end{aligned}$$

If $\delta_i = 1$, then the i^{th} disjunctive constraint is actively enforced. The last constraint ($\sum_{i=1}^{n} \delta_i \geq 1$) ensures that at least k of the constraints is active.

3.6.4.1. Strip Packing Problem

Suppose we a selection of rectangles I and a 2-dimensional strip with width W and infinite height. Each rectangle $i \in I$ has width w_i and height h_i and we want to pack the rectangles into the strip so that (3.23a) overall height is minimized, (3.23b) overall width is less than W, and (3.23c) none of the rectangles overlap. See Figure 3.5 for an example.

Let (x_i, y_i) denote the position of the lower-left-hand corner of each rectangle $i \in I$. The overlapping constraint (3.23c) is the trickiest part. Consider a pair of rectangles i and j: rectangle j is located entirely to the left of rectangle i if x_j has a value larger than $x_j + w_j$. That is, if $x_j \geq x_j + w_j$. On the other hand, if $y_j \geq y_i + h_i$, then rectangle j is located entirely above rectangle i. If either of these constraints is satisfied then rectangles i and j do not overlap. We could also place i above or to the right of j. This gives four



(a) Pack the rectangles into the strip **(b) Minimize overall height H.**

Figure 3.5: An Example of the Strip Packing Problem

constraints that we need to satisfy at least one of. The model can be thought of thusly:

$$\text{Minimize} \quad \max_{i \in I}\{y_i + h_i\} \tag{3.23a}$$

$$\text{s.t.} \quad \max_{i \in I}\{x_i + w_i\} \leq W \tag{3.23b}$$

$$\left.\begin{aligned} x_i + w_i &\leq x_j \\ x_j + w_j &\leq x_i \\ y_i + h_i &\leq y_j \\ y_j + h_j &\leq y_i \end{aligned}\right\} \text{At least one of these for every distinct pairs of rectangles } i, j \in I \tag{3.23c}$$

$$x_i, y_i \geq 0 \qquad \forall\, i \in I \tag{3.23d}$$

Constraint (3.23c) is a set of four *disjunctive* constraints. This can be expressed linearly by introducing a

new binary indicator variable δ_{ijk} for each of the disjunctive constraints in (3.23c):

$$
\begin{aligned}
\text{Minimize} \quad & H & & \\
\text{s.t.} \quad & y_i + h_i \leq H & \forall\, i \in I & \quad (3.23a)\\
& x_i + w_i \leq W & \forall\, i \in I & \quad (3.23b)\\
& x_i + w_i \leq x_j + M(1-\delta_{ij1}) & \forall\, i,j \in I:\ i > j & \\
& x_j + w_j \leq x_i + M(1-\delta_{ij2}) & \forall\, i,j \in I:\ i > j & \\
& y_i + h_i \leq y_j + M(1-\delta_{ij3}) & \forall\, i,j \in I:\ i > j & \quad (3.23c)\\
& y_j + h_j \leq y_i + M(1-\delta_{ij4}) & \forall\, i,j \in I:\ i > j & \\
& \sum_{k=1}^{4} \delta_{ijk} \geq 1 & \forall\, i,j \in I:\ i > j & \\
& x_i, y_i \geq 0 & \forall\, i \in I & \\
& \delta_{ij} \in \{0,1\}^4 & \forall\, i,j \in I:\ i > j &
\end{aligned}
$$

If $\delta_{ijk} = 1$, then the k^{th} disjunctive constraint is actively enforced. The last constraint $\left(\sum_{k=1}^{4} \delta_{ijk} \geq 1\right)$ ensures that at least one of the constraints is active. An optimal solution to the example is given in Figure 3.6; it was found via GurobiPy.

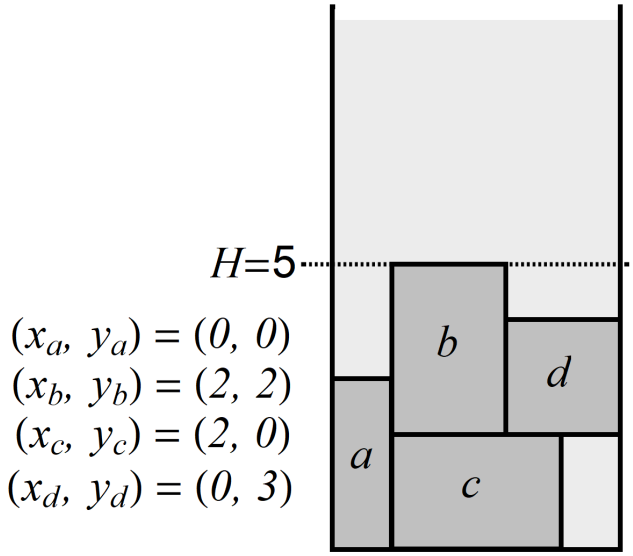


Figure 3.6: The Optimal Solution to the example problem

This problem is considered strongly NP-Hard.