TIMOTHY R. ANDERSON

OPERATIONS RESEARCH USING R

Contents

Preface 5

1 Introduction 7

1.1 What is Operations Research 7

1.2 Purpose of this Book 8

1.3 Range of Operations Research Techinques 8

1.4 Relationship between Operations Research and Analytics 9

1.5 Why R? 9

1.6 Conventions Used in this Book 11

1.7 Getting Started with R 11

2 Introduction to Linear Programming 19

2.1 What is Linear Programming 19

2.2 Two Variable Base Case 19

2.3 Implementing the Base Case with Piping 26

2.4 Adding a Third Product (Variable) 27

2.5 Three Variable Case Results and Interpretation 28

2.6 Linear Programming Special Cases 29

2.7 Abstracting the Production Planning Model 34

2.8 Advice on Homeworks 36

2.9 Comments specific to R 36

2.10 Graphically Solving LPs with R 37

2.11 Next Steps 43
2.12 Miscellaneous Tips and Tricks 43
2.13 `ompr` Related Tips and Tricks 43
2.14 RMD Tips and Tricks 44
2.15 RStudio Tips and Tricks 44
2.16 General R Tips and Tricks 44
2.17 LaTeX Tips and Tricks 45
2.18 General Tips and Tricks 45

3 More Linear Programming Models 47
3.1 Types of LP Models 47
3.2 The Algebraic Model 47
3.3 Tips and Conventions for Algebraic Models 48
3.4 Building the Generalized Model in R 48
3.5 Examining the Results 50
3.6 Changing the Model 52
3.7 Allocation Models 55
3.8 Covering Models 55
3.9 Transportation Models 56
3.10 Transhipment Models 58
3.11 Standard Form 58
3.12 Exercises 58

4 Linear Programming Sensitivity Analysis 61
4.1 Base Case 61
4.2 Shadow Prices 62
4.3 Testing Adding Labor to Machining 63
4.4 Shadow Prices of Underutilzed Resources 64
4.5 Reduced Costs of Variables 65
4.6 Exercises 69

5 *Data Envelopment Analysis* 71

5.1 *Introduction* 71

5.2 *Creating the Data* 72

5.3 *Graphical Analysis* 73

5.4 *The Linear Programs for DEA* 74

5.5 *Creating the LP - The Algebraic Approach* 76

5.6 *Returns to Scale* 82

5.7 *Multiple Inputs and Multiple Ouptuts* 85

5.8 *Extracting Multiplier Weights from Envelopment Results* 91

5.9 *Slack Maximization* 93

5.10 *Further Reading* 97

5.11 *Exercises* 97

6 *Mixed Integer Optimization* 99

6.1 *Example of Integrality having Little Impact* 99

6.2 *Example of Integality having a Large Impact* 102

6.3 *The Branch and Bound Algorithm* 104

6.4 *Computational Complexity* 111

6.5 *Binary Variables and Logical Relations* 112

6.6 *Fixed Charge Models* 116

6.7 *Linking Constraints with "Big M"* 119

6.8 *Fixed Charge Implementation* 122

6.9 *Model Results and Interpretation* 123

7 *More Integer Programming Models* 127

7.1 *Overview* 127

7.2 *Revisiting the Warehouse Location Problem* 127

7.3 *Solving MIPs with Different Solvers* 141

7.4 *Comparing Results across Solvers* 146

7.5 *Popularity of LP Solvers* 147

7.6 *Solving Sudoku Puzzles using Optimization* 149

7.7 Implementing the Sudoku model in ompr 152
7.8 Conclusion 156
7.9 Exercises 156

8 Goal Programming and Multiple Objectives 161

A An Introduction to Summation Notation 163
A.1 Basic Summation Notation 163
A.2 Notation in LaTeX 164
A.3 Sums 164
A.4 Products 164
A.5 Summary of Mathematical Notations 165
A.6 Sequences and Summation Notation 166
A.7 Applications of Summation 167
A.8 Double Summation 169
A.9 Applications of Double Summation 169
A.10Exercises 170
A.11References 171

Preface

This book covers using R for doing Operations Research. The current focus is on linear optimization. In the future it may grow to include a broader range of Operations Research methods.

I would like to thank many people for their contributions, collaborations, and assistance over the years. All errors are my fault though.

- **Dirk Schumacher,** author of the `ompr` package used heavily throughout this book
- **Dr. Dong-Joon Lim,** applications and methodological work in DEA
- **Dr. Gerry Williams,** application of DEA to construction contracting
- **Dr. Janice Forrester,** application of DEA to the energy industry
- **Dr. Scott Leavengood,** application of DEA to wood products
- **Dr. Oliver (Lane) Inman,** early work on TFDEA
- **Dr. K. Louis Luangkesorn,** author of the first vignette on using `glpk` in R
- **Dr. Chester Ismay,** contributions to the Portland and broader R community
- **Dr. Jili Hu,** rich interactions during his sabbatical in Portland
- **Tom Shott,** primary author of the `TFDEA` package
- **Nina Chaichi, PhD student,** many suggestions over time
- **Aurobindh Kalathil Kumar,** PhD student, many suggestions over time
- **Maoloud Dabab,** PhD student, many suggestions over time
- **Kevin van Blommestein,** earlier DEA & R work
- **William (Ike) Eisenhauer,** LaTeX formulation improvements
- **Andey Nunes,** coding improvements
- **Christopher Davis,** graphical example of LP
- **Thanh Thuy Nguyen,** fixed charge example
- **Roland Richards,** converted format to Tufte-style
- **Caroline Blackledge,** Co-author of summation introduction appendix

- **Alexander Keller,** Co-author of summation introduction appendix
- **Shahram Khorasanizadeh,** Co-author of summation introduction appendix
- **Jose Banos,** Contributor of more formatting in the spirit of Tufte
- **Dawei Zhang,** Further editorial work on the book

In addition, several groups have been of tremendous help:

- The Portland Meetup, R User's Group
- The Extreme Technology Analytics Research Group
- Past ETM 540/640 Operations Research Group classes, particularly the Fall 2018 and Winter 2019 classes which helped test the early versions of this book.

Most of all, I would like to also express my appreciation for my family's patience while working on this book with many late nights: Carrie, Trent, and Paige.

1
Introduction

1.1 What is Operations Research

Operations Research is the application of mathematics and scientific approaches to decision making. It is a field deeply connected with applications.

The field of Operations Research has strong roots in World War II as nations and militaries urgently tried to best make use of their resources and capabilities. Leaders asked people from various disciplines to help improve a variety of problems such as submarine hunting, mining of resources, and production planning. Essentially, these tools are scientific approaches for making decisions for the design and operation of a system with limited resources. The tools that that they used came from combining applied math and scientific approaches resulting in tremendous improvements.

After the war, many of the same approaches were also then used for business applications. Military groups most commonly used the term Operations Research while business groups often used the term Management Science.

Neither term, Operations Research nor Management Science is perfect. Operations Research is often confused with the strongly overlapping area of Operations Management and also the word research implies to some that it is strictly theoretical. Conversely, Management Science might imply that the rest of management is non-scientific or that it is just limited to business applications. In this book, we will use the term Operations Research to mean both Operations Research and Management Science as well as often abbreviate it as OR.

1.2 Purpose of this Book

There are many comprehensive introductions to Operations Research available including classics from Hillier and Lieberman, Winston, and others. The goal of this book is to provide a timely introduction to Operations Research using a powerful open source tool, R. It was written to support the ETM 540/640 Operations Research course at Portland State University which is taught on a quarter basis for people without a background in Operations Research or R. As such the current scope is limited to what can be readily accomplished in a 10 week quarter. There are many other Operations Research tools available.

This book is meant to be a hybrid, both serving as an introduction to R and to Operations Research.

A quick getting started with R will be helpful. I strongly recommend using RStudio. *A Modern Dive* is a web book on using R and refreshing statistics that is very helpful (Ismay and Kim n.d.). I also like books such as R in a Nutshell (Adler 2012) or R in Action (Kabacoff 2011).

I usually find the best way for me to learn a new tool is to roll up my sleeves and jump right in. Hence the book can be approached in that way and just be ready for a little more experimentation. This book is available on Github and the R Markdown files are available for use. Code fragments are shown quite liberally for demonstrating how things work. While this may be a bit verbose and at times repetitious, it is meant to help people jump in at various points of the book.

While code fragments can be copied from the R markdown files for this book, it is often best to physically retype many of the code fragments shown as that gives time to reflect on what each statement is doing.

1.3 Range of Operations Research Techinques

Operations Research covers many different techniques. They can be classified in a wide range of ways. Some of the more common approaches are:

- Optimization
- Simulation
- Queuing Theory

- Markov Chains
- Inventory Control
- Forecasting
- Game Theory
- Decision Theory

Each of these topics can require a full graduate level class or more to cover. In particular, Optimization and Simulation can be further subdivided into separate sub areas that represent entire specialties of their own. In this book, we currently limit ourselves to the field of Optimization for this current book. The interested reader is welcome to explore each area further.

Analysis methods are typically characterized as either descriptive prescriptive. A descriptive technique would be one that describes the current state of the system. A prescriptive technique is one that would prescribe a certain course of action to achieve a desired outcome. For example, weather forecasting is a very sophisticated descriptive discipline since we don't generally have control over short term weather. On the other hand, if we were to look at intervention approaches such as cloud seeding we might try to come up a prescriptive application. Prescriptive applications are abundant and can often be wherever decisions need to be made.

1.4 Relationship between Operations Research and Analytics

The field of Operations Research significantly predates that of the term analytics but they are closely linked. This is strongly demonstrated by the leading professional organization for operations research, INFORMS (The Institute for Operations Research and Management Science), developing a professional certification for analytics. This certification process had leadership from industry representing the diverse users of Operations Research techniques.

1.5 Why R?

This book adopts the platform of R for doing operations research and analytics. R is a popular, open source programming language developed by statisticians, originally as an open source alternative to the statistics language S. As such, it has very strong numerical methods implementations. Given the open source nature, it has been popular for researchers and many of these researchers have written their own packages to extend the functionality of R. These packages cover a tremendously broad range of topics and well over 10,000 are available from CRAN (the Comprehensive R Archive Network).

In about 2012, my Extreme Technology Analytics Research Group was facing a decision. We had been using proprietary analytics tools for years but had problems with software licensing and the limited reach of specialized languages. We wanted a single platform that could do both statistics and optimization, was freely available with no ongoing license renewal, and was readily extensible.

In the end, we debated between Python and R. While R grew out from statisticians to encompass broader areas, Python grew from a general purpose programming language to add features to address a broad range of different application areas including statistics. While they both have deep roots with over 20 years of active development, this also means that aspects of their original designs are somewhat dated and carry forward the burden of legacies of past decisions when computers were more limited and the meaning of a big dataset would have been much smaller.

Python had some advantages from the optimization side while R had advantages from the statistical side. In the end, we chose R because Python's numerical methods capabilities were not quite as rich. Since that time, both R and Python have significantly matured.

As time has gone by, R has developed robust toolsets such as RStudio to making it more powerful and easier to use. R's extensive package community has become so diverse and powerful that proprietary statistics software such as SPSS and JMP now promote that they can use R packages. The result is that R is still a great choice for analysts and code developed for R should be usable for a long time.

As should be expected, new tools continue to arise. In particular, Julia is a new, modern language developed around numerical analysis. Industry adoption of Julia is far behind R and Python given their head start of multiple decades. Even if Julia or some other environment does someday become more popular, skills developed using R will be readily transferable and provide an excellent foundation for learning new languages.

A PhD student that I knew was preparing for his final defense and did a practice presentation with his advisor and a professor. He was proud of a well polished 250 page thesis with dozens of tables of numerical results. The analysis included dozens of regressions, all with similar form and dependent variables. The professor asked him why his dependent variables was the number of projects rather than projects per year. The student thought for a moment and realized that all of his analysis needed to be redone. This would take days or weeks if he had used a GUI based tool. Using R he did a search and replace for the dependent variable, cross-checked results, and met the next day to update that professor and his advisor of the new (and much better!) results. He successfully defended his thesis a couple weeks later.

1.6 Conventions Used in this Book

I've adopted the following conventions in this book.

- R code fragments, including packages will be denoted using monospaced text such as adding two R data objects `a+b`.
- Mathematical symbols and equations will be italicized. For example, if two variables are mathematical symbols and being added, then it would be $a+b$.
- Tables of information and code fragments are provided liberally and at times erring on the side of repetitious to assist in interpretation and reproducing analyses.
- There may be more efficient ways of performing operations but the focus is on readability.

This book is continuing to be updated. Comments, suggestions, and corrections are welcome.

1.7 Getting Started with R

Let's get started with R and explain some conventions to be used throughout the book. First, let me be clear, the goal of this section is not to provide a comprehensive introduction to R. Entire books are written on that subject. My goal here is to simply make sure that everyone can get started productively in the material covered later.

Since this book is focused on using R for operations research, we will focus on the capabilities that are needed for this area and introduce additional features and functions as needed. If you are already comfortable with R, RStudio, and RMarkdown, you may skip the remainder of this section.

Begin by ensuring that you have access to or installed R and RStudio. Both are available for Windows, Mac, and Linux operating systems as well as being available as web services.

Now, let's assume that you are running RStudio.

In this book, I will frequently show code fragments called chunks to show R code. In general, these code chunks can be run by simply retyping the command(s) in the Console.

```r
a <- 7
```

This command assigns the value of 7 to the variable a. It is standard in R to use `<-` as an assignment operator rather than an equals sign. We can then use R to process this information.

```
6*a
```

```
## [1] 42
```

Yes, not surprisingly, _6*7_ is *42*. Notice that if we don't assign that result to something, it gives an immediate result to the screen.

We will often be using or defining data that has more than one element. In fact, R is designed around more complex data structures. Let's go ahead and define a matrix of data.

```
b<-matrix(c(1,2,3,4,5,6,7,8))
```

By default, it is assuming that the matrix has one column which means that every data value is in a separate row. The `c` function is a concatenate operator meaning that it combines all the following items together.

Let's look at *b* now to see what it contains.

```
b
```

```
##      [,1]
## [1,]    1
## [2,]    2
## [3,]    3
## [4,]    4
## [5,]    5
## [6,]    6
## [7,]    7
## [8,]    8
```

Let's instead define this matrix to have four columns and two rows.

```
b<-matrix(c(1,2,3,4,5,6,7,8), ncol=4)
b
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    5    7
## [2,]    2    4    6    8
```

Notice that we only needed to tell R that the matrix has four columns and it then knew that there two rows. We could have set the number of rows to be two for the same result.

This is still a little boring. Let's give the rows and columns names.

```
b<-matrix(c(1,2,3,4,5,6,7,8), ncol=4,
          dimnames=c(list(c("Row1", "Row2")),
                     list(c("Col1", "Col2","Col3","Col4"))))
```

Okay, this command has a lot more going on. The term `dimnames` is a parameter that contain names for rows and columns. One thing to note is that this line fills up more space than a single line so it rolls over to multiple line.

The `dimnames` parameter will get get two concatenated (combined) lists. The first list is a combined list of two text strings "Row1" and "Row2". The next line does the same for columns.

Let's confirm that it works.

```
b
```

```
##      Col1 Col2 Col3 Col4
## Row1    1    3    5    7
## Row2    2    4    6    8
```

This table is still not that "nice" looking. Let's use a package that does a nicer job of formatting tables. To do this, we will use an extra package. Up to this point, everything that we have done just simply uses standard built in functions of R. The package that we will use is `grid.table` but there are plenty of others available such as `pander`, `kable`, `xtable`, and `huxtable`. Let's start by loading the package.

```
# install.packages("gridExtra")
# Use this command if pander is not installed.
library(gridExtra)
```

If R indicates that you don't have pander installed on your computer, you can press the "Install" command under the Packages tab and then type in pander or use the `install.packages` command.

pander: The main aim of the pander R package is to provide a minimal and easy tool for rendering R objects into Pandoc's markdown. The package is also capable of exporting/converting complex Pandoc documents (reports) in various ways

Notice the hash symbol is used to mark comment in the above code chunk. It is also used to "comment out" a command that I don't need to use at the current time. Using comments to explain what a command is doing can be helpful to anyone that needs to revisit your code in the future, including yourself!

Now, we have several options to draw a table. We could use the `pander` package to work or we can use the grid.table package.

gridExtra: Provides a number of user-level functions to work with “grid” graphics, notably to arrange multiple grid-based plots on a page, and draw tables.grid.table draw a text table.

```
grid.table(b)
```

	Col1	Col2	Col3	Col4
Row1	1	3	5	7
Row2	2	4	6	8

Table 1.1: Nicely formatted table using pander

Let’s continue experimenting with operators.

```
c<-a*b
grid.table(c)
```

	Col1	Col2	Col3	Col4
Row1	7	21	35	49
Row2	14	28	42	56

Table 1.2: Scalar Multiplication of Matrix b

Now let’s do a transpose operation on the original matrix. What this means that it converts rows into columns and columns into rows.

```
d<-t(b)
grid.table(d)
```

	Row1	Row2
Col1	1	2
Col2	3	4
Col3	5	6
Col4	7	8

Table 1.3: Transposition of Matrix b

Oops.. Still not right. Need to do some more...

Woops, notice that row and column names are also transposed. The result is that we now have rows labeled as columns and columns labeled as rows!

Now we have done some basic operations within R. Try your hand at the following exercises.

Exercise 1.1 (Changing Row and Column Names)**.** Change the row and column names for the transposed matrix above.

Here is my solution to that...

```
c
```

```
##      Col1 Col2 Col3 Col4
## Row1    7   21   35   49
## Row2   14   28   42   56
```

```
d
```

```
##      Row1 Row2
## Col1    1    2
## Col2    3    4
## Col3    5    6
## Col4    7    8
```

Assume that your company's product has a demand of 10 widgets on Monday, increasing by five units for every day through Sunday. Create a matrix where the column indicates that the day of the week and the row is the product name.

Recall that we have created a variety of objects now: a, b, c, and d. R provides a lot of tools for slicing, dicing, and combining data.

```
grid.table(b)
```

	Col1	Col2	Col3	Col4
Row1	1	3	5	7
Row2	2	4	6	8

Table 1.4: Original Matrix b

Let's look at the original matrix, b and what we can do with it. Let's grab the second row, third column and last element.

```
temp <- as.matrix(b[2,])
grid.table(temp)
```

Col1	2
Col2	4
Col3	6
Col4	8

Table 1.5: Second Row of b

The `as.matrix` function is used to convert the object into a matrix so that pander can display it well. Note that I might explore alternatives to `grid.table` such as `pander`or `huxtable` in the future. Here is a discussion of huxtable and comparison of many different packages for displaying tables `https://hughjonesd.github.io/huxtable/design-principles.html`

```
temp2 <- as.matrix(b[,3])
grid.table(temp2)
```

Row1	5
Row2	6

Table 1.6: Third Column of b

Grabbing the third column is not terribly interesting but operations such as this will often be quite useful.

```
temp4 <- as.matrix(b[2,4])
grid.table(temp4)
```

8

Table 1.7: Last Element of b

A table made up of just one element is even less profound but is again useful. Now, let's show how we can combine two objects.

Recall tables *b* and *c* from earlier.

Table 1.1: Matrix b

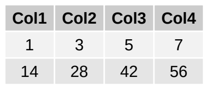
	Col1	Col2	Col3	Col4
Row1	1	3	5	7
Row2	2	4	6	8

Table 1.2: Matrix c

	Col1	Col2	Col3	Col4
Row1	7	21	35	49
Row2	14	28	42	56

Let's take the first row of `b` and combine it with the second row of `c` to form a new matrix, `e`. Since these are rows that are going to be combined, we will use a command, `rbind`, to bind these rows together.

```
e<-rbind(b[1,],c[2,])
grid.table(e)
```

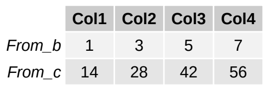


Col1	Col2	Col3	Col4
1	3	5	7
14	28	42	56

Table 1.8: Combined Matrix

Notice that *e* has inherited the column names but lost the row names. Let's set the row names for this matrix.

```
rownames(e)<-list("From_b", "From_c")
grid.table(e)
```

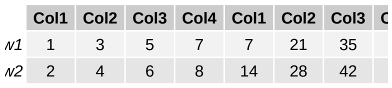


	Col1	Col2	Col3	Col4
From_b	1	3	5	7
From_c	14	28	42	56

Table 1.9: Combined Matrix with Explanation of Source

We could combine all of matrix `b` and matrix `c` together using row binding or column binding. Table 1.10 will bind using columns.

```
temp5<- cbind(b,c)
grid.table(temp5)
```

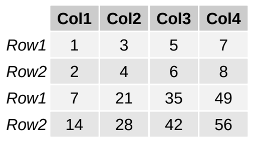


	Col1	Col2	Col3	Col4	Col1	Col2	Col3	C
w1	1	3	5	7	7	21	35	
w2	2	4	6	8	14	28	42	

Table 1.10: Column Binding of Matrices b and c

And Table 1.11 will use rows as bidding condition. pander(rbind(b,c), caption="Row Binding of Matrices b and c")

```
temp6 <- rbind(b,c)
grid.table(temp6)
```



	Col1	Col2	Col3	Col4
Row1	1	3	5	7
Row2	2	4	6	8
Row1	7	21	35	49
Row2	14	28	42	56

Table 1.11: Row Binding of Matrices b and c

Data organizing is a less glamorous part of the job for practicing analytics professionals but can consume a majority of the workday. There is a lot more that can be said about data wrestling but scripting the data cleansing in terms R commands will make the work more repeatable, consistent, and in the end save time.

Exercise 1.2 (Creating a Matrix of Daily Demand for Four Weeks)**.** Assume that your company's product has a demand of 10 widgets on Monday, increasing by five units for every day through Sunday. Build a matrix of four weeks of demand where each row is a separate week.

Exercise 1.3 (Creating a Matrix of Demand for Two Products)**.** Assume that your company's product has a demand of 10 widgets on Monday, increasing by five units for every day through Sunday. Gadgets have a demand of 20 on Monday, increasing by 3 units a day through Sunday. Build a matrix showing each product as a separate row.

2
Introduction to Linear Programming

2.1 What is Linear Programming

Linear programming is a tool for optimization. It is a widely used tool for planning, scheduling, resource allocation and many other applications.

2.2 Two Variable Base Case

This data and case is drawn from an example in Chapter 2 of Kenneth Baker's *Optimization Modeling with Spreadsheets*, 3rd Edition. The example is a modification of the example 2.1 on page 29.

The goal of the furniture manufacturer in this case is to find the best product mix of Chairs and Desks.

Making a piece of furniture requires time in fabrication, assembly, and machining as a certain amount of wood. For example, a chair requires 6 hours of fabrication time, 8 hours of assembly, and 6 hours of machining. It uses 40 square feet of wood but the net profit is $20. The characteristics of desks are similar but different as shown in the Table 1 on the right as well as the available amount of each limited resource.

Characteristic	Chairs	Desks	Available
Profit	$20	$14	
Fabrication	6	2	1440
Assembly	8	6	1440
Machining	6	4	2000
Wood	40	25	9600

Table 2.1: Two variable base case

A simple LP now is to find the production plan of products that results in the most profit. In order to do so, we need to define certain key items:

- the goal(s)
- the decisions
- the limitations

Let's start with the goal(s). In this case, the production manager is simply trying to make as much profit as possible. While cost cutting is also a goal for many organizations, in this case and many applications profit maximization is appropriate. Maximizing profit is the referred to as the *objective* of the model. People new to linear programming

will often think of the decisions as the amount of each resource to use. Instead, the decisions in this case would be how much to make of each particular product. This drives the resource usage and the resource usage is a byproduct of these decisions. These decisions can take on a range of values and are therefore called *decision variables.*

Single-Objective: This type of linear programming result when only one objective or goal can be accommodated.

The decision variables are then combined in some way to reflect the performance with respect to the organization's objective. The equation combining the decision variables to reflect this is then the *objective function.* In general we will assume that there is a single objective function, at least for now.

Multiple-Objective: This type of linear programming result when multiple objectives or goals can be accommodated. This will be explored in Chapter 7.

Lastly, what is limiting the organization from even better performance? There are typically many limits such as the number of customers, personnel, supplier capacity, etc. In this case, we focus on a set of resource limits based on staffing in different centers and the supply of raw material (wood). Since these limitations constrain the possible values of decision variables, they are called constraints.

Every optimization model can be thought of a collection of:

- an objective function (goal)
- decision variable(s) (decisions)
- constraint(s) (limitations)

Let's put things together in the context of this application. In the base case, our objective function is to Maximize Profit. We can't express it though until we define our decision variables. It is good practice to very clearly and precisely define decision variables. In this case it is straightforward but they can get much more complicated as we move into richer and larger models.

Let's define them:
* Chairs = # of Chairs to Make
* Desks = # of Desks to Make

$$\begin{aligned}
&\text{Max: } 20Chairs + 14Desks \\
&\quad \text{s.t.:} \\
&\qquad 6Chairs + 2Desks \leq 2000 \\
&\qquad 8Chairs + 6Desks \leq 2000 \\
&\qquad 6Chairs + 4Desks \leq 1440 \\
&\qquad 40Chairs + 25Desks \leq 9600 \\
&\qquad Chairs, Desks \geq 0
\end{aligned}$$

Formula 2.1: Optimization mode. Base equation with 2 variables

Our objective function and constraints can now be written as the optimization model shown on the right.

Note that since the objective function and each constraint is a simple linear function of the decision variables, this is what we call a *linear* programming model or LP for short. It would not be linear if any nonlinear function is made of the decision variables. For example, squaring a decision variable, using conditional logic based on the variable value, or multiplying two variables. These and other issues would then require using nonlinear programming or NLP. NLP is also widely used but has limitations.

Linear programming: (LP, also called linear optimization) is a method to achieve the best outcome (such as maximum profit or lowest cost) in a mathematical model whose requirements are represented by linear relationships.

It is impressive the number of situations that can be modeled well using linear programming. Keeping to the world of linear programming in general allows for finding the very best solution to very big problems in a short amount of time. For example, it is common for practitioners to be analyzing real-world problems with hundreds of thousands of decision variables and constraints.

For the sake of simplicity, we will implement our first R LP model using explicit variables and data consistent with the first formulation.

First, let us load the required libraries. Then we will move on to the actual implementation.

```
library (pander, quietly = TRUE)
   # Used for nicely formatted tables
library (ROI, quietly = TRUE)
   # R Optimization Interface
library (ROI.plugin.glpk, quietly = TRUE)
   # Plugin for solving
library (ompr, quietly = TRUE)
   # Allows specifying model algebraically
library (ompr.roi, quietly = TRUE)
   # Glue for ompr to solve with ROI
```

Each of these packages plays a role. As we discussed in the earlier chapter, pander makes for nicely formatted tables.
This brings us to four packages that provide the optimization functions that we be relying on frequently. The ROI package is the R Optimization Interface for for connecting various optimization solvers to R. The ROI.plugin.glpk provides a connection between the the glpk solver and R through ROI. While this would be sufficient, we are going to make use of the ompr package by Dirk Schumacher to provide algebraic representations for linear programs. The ompr package also requires a connector to ROI, aptly named ompr.roi.

As noted earlier, if these packages are not preinstalled, you may need to install them using the install.packages function or from the Packages tab in RStudio.

Now we move on to implement and solve the linear program. Let's go through it step by step.

```
model0  <- MIPModel()     # Initialize an empty model
```

The first line initializes an empty model and stores it in `model0`. We can see that the model is empty by simply displaying the model summary.

```
model0
```

```
## Mixed integer linear optimization problem
## Variables:
##   Continuous: 0
##   Integer: 0
##   Binary: 0
## No objective function.
## Constraints: 0
```

There are no constraints or variables. Next we will add variables.

```
model0a <- add_variable(model0, Chairs,
          type = "continuous", lb = 0)
model0b <- add_variable(model0a, Desks,
          type = "continuous",lb = 0)
```

The next line takes `model0` and adds the *Chairs* variable to it, creating an enhanced `model0a`. These variables are continuous (as compared to integer or binary), and non-negative. The variable is made non-negative by setting a zero lowerbound (`lb=0`). The lowerbound can be set to other values such as a minimum production level of ten. Also upperbounds for variables can be set using `ub` as a parameter. Do the same thing to add the *Desks* variable to `model0a` creating `model0b`. Let's check the new model.

```
model0b
```

```
## Mixed integer linear optimization problem
## Variables:
##   Continuous: 2
##   Integer: 0
##   Binary: 0
## No objective function.
## Constraints: 0
```

Next, we need to add the objective function. We set the objective function as well declaring it to be a *max* rather than a *min* function.

```
model0c<-set_objective(model0b,20*Chairs+14*Desks,"max")
```

Now we move on to adding constraints.

```
model0d<-add_constraint(model0c,6*Chairs+2*Desks<= 2000)
#fabrication
model0e<-add_constraint(model0d,8*Chairs+6*Desks<= 2000)
#assembly
model0f<-add_constraint(model0e,6*Chairs+4*Desks<= 1440)
#machining
model0g<-add_constraint(model0f, 40*Chairs + 25*Desks<= 9600)
#wood
result0<-solve_model(model0g, with_ROI(solver = "glpk"))
```

Notice that we do not need to include non-negativity constraints since they were in the earlier definition of the variables. Let's take a look at the summary of the model after all of the constraints have been added.

```
model0g
```

```
## Mixed integer linear optimization problem
## Variables:
##   Continuous: 2
##   Integer: 0
##   Binary: 0
## Model sense: maximize
## Constraints: 4
```

Our model has variables, constraints, and an objective function. Let's go ahead and solve it. The LP can be solved using different LP engines - we'll use `glpk` for now. We'll assign the result of the solved model to `result0`

nonlinear programming or NLP. NLP is also widely used but has limitations.

glpk: "GNU Linear Programming Kit" package is intended for solving large-scale linear programming (LP), mixed integer programming (MIP), and other related problems.

```
result0 <- solve_model(model0g, with_ROI(solver = "glpk"))
```

We built up the model line by line incrementally taking the previous model, adding a new element, and storing it in a new model. Note that we could keep placing them into the original model if we had wished such as the following.

```
model0  <- MIPModel()      # Initialize an empty model
model0 <- add_variable(model0, Chairs,
                       type = "continuous", lb = 0)
model0 <- add_variable(model0, Desks,
                       type = "continuous",lb = 0)
model0<- set_objective(model0,20*Chairs + 14*Desks, "max")
model0<- add_constraint(model0,6*Chairs + 2*Desks <= 2000)
#fabrication
model0 <- add_constraint(model0,8*Chairs + 6*Desks <= 2000)
#assembly
model0<- add_constraint(model0,6*Chairs + 4*Desks <= 1440)
#machining
model0<- add_constraint(model0,40*Chairs + 25*Desks <= 9600)
#wood
result0 <- solve_model(model0, with_ROI(solver = "glpk"))
```

In this case, we take the previous command's output, `model0`, add an element to it, and then store it back in the same object, `model0`. We keep doing it for each element until we are done.

This is equivalent to assigning the value of *2* to *a*. We then multiple *a* by *3* and assign the result again to *a*.

```
a <- 2
a <- 3*a
a
```

```
## [1] 6
```

Note that this is part of why `<-` as an assignment operator is better than an equals sign, `=` which would make the reader think that if $a=3a$ then a must be zero.

It might be helpful to see what the model looks like before examining the results.

```
model0
```

```
## Mixed integer linear optimization problem
## Variables:
##   Continuous: 2
##   Integer: 0
##   Binary: 0
## Model sense: maximize
## Constraints: 4
```

Furthermore, we can use the `extract_constraints` command to see what the actual constraints look like in the model.

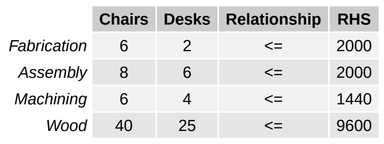
```
extract_constraints(model0)
```

```
## $matrix
## 4 x 2 sparse Matrix of class "dgCMatrix"
##
## [1,]  6  2
## [2,]  8  6
## [3,]  6  4
## [4,] 40 25
##
## $sense
## [1] "<=" "<=" "<=" "<="
##
## $rhs
## [1] 2000 2000 1440 9600
```

This should look familiar. Let's see if we can arrange to make it look more similar to what we had given the model.

```
constr0<-extract_constraints(model0)
# Pass the constraints out of  model
constr00<- (cbind(as.matrix(constr0$matrix),
                  # Get the matrix of data
                  as.matrix(constr0$sense),
                  # Get the inequality
                  as.matrix(constr0$rhs)))
                  # Get the Right Hand Side
dimnames(constr00)<-c(list(c("Fabrication",
                        "Assembly","Machining","Wood")),
                      list(c("Chairs",
                        "Desks","Relationship","RHS")))
grid.table(constr00)
```

In the first version of the LP model implementation, we created many different versions of the model along the way while we really just keep adding stuff to the same model. This uses extra memory and will be slower. The second implementation keeps reusing the same model object until the very end when solving is done and it places the results into an object. This may be more memory efficient but is still likely slower and carries along a lot of extra notation.



	Chairs	Desks	Relationship	RHS
Fabrication	6	2	<=	2000
Assembly	8	6	<=	2000
Machining	6	4	<=	1440
Wood	40	25	<=	9600

Table 2.2: Elements of the Constraints

2.3 Implementing the Base Case with Piping

In each step, we are simply taking the output of the previous step and feeding it in as the first term of the following step. It is almost like the output is a bucket that is then passed as a bucket into the next step. Rather than carrying a bucket from one step into the next one, a plumber might suggest a pipe connecting one step to the next. This is done so often that we refer to it as piping and a pipe operator.

The pipe symbol: Represented by `%>%` will let us do this more compactly and efficiently. It takes a little getting used to but is a better way to build the model.

Here is the equivalent process for building the base case implementation using a piping operator without all the intermediate partial models being built. Notice how it is a lot more concise. We will typically use this approach for model building but both approaches are functionally equivalent.

```r
library (magrittr, quietly = TRUE) # Used for pipes/dplyr
# library (dplyr, quietly = TRUE)    # Data management
result0 <- MIPModel() %>%
  add_variable(Chairs, type = "continuous", lb = 0) %>%
  add_variable(Desks, type = "continuous",lb = 0) %>%

  set_objective(20*Chairs + 14*Desks, "max") %>%

  add_constraint(6*Chairs + 2*Desks<= 2000) %>% #fabrication
  add_constraint(8*Chairs + 6*Desks<= 2000) %>% #assembly
  add_constraint(6*Chairs + 4*Desks<= 1440) %>% #machining
  add_constraint(40*Chairs + 25*Desks<= 9600) %>%   #wood
  solve_model(with_ROI(solver = "glpk"))
```

The magrittr package, offers a set of operators which make your code more readable by:
+structuring sequences of data operations left-to-right (as opposed to from the inside and out), +avoiding nested function calls, +minimizing the need for local variables and function definitions, and +making it easy to add steps anywhere in the sequence of operations.

Note that we load a new package `magrittr` which gives us the piping function that simplifies model building.

The first line creates the basic model. The `%>%` serves as a pipe symbol at the end of each line. It basically means take product of the previous command and feed it in as the first argument of the following command.

Let's check to see the status of the solver. Did it find the optimal solution? We do this by extracting the solver status from the result object.

```r
print(solver_status(result0))
```

```
## [1] "optimal"
```

Furthermore, we can do the same thing to extract the objective function value.

```
print(objective_value(result0))
```

```
## [1] 4880
```

The command `objective_value(result0)` extracts the numerical result of the objective function (i.e. maximum possible profit) but it does not tell us what decisions give us this profit.

Since the LP solver found an optimal solution, let's now extract the solution values of the decision variables that give us this profit.

```
print(get_solution(result0, Chairs))
```

```
## Chairs
##    160
```

```
print(get_solution(result0, Desks))
```

```
## Desks
##   120
```

2.4 Adding a Third Product (Variable)

We will now extend the previous model to account for a third product, Tables. The goal is now to find the best product mix of Chairs, Desks, and Tables. See the Table 2 with the summary of the new situation.

Characteristic	Chairs	Desks	Tables	Available
Profit	$20	$14	$16	
Fabrication	6	2	4	1440
Assembly	8	6	8	1440
Machining	6	4	25	2000
Wood	40	25	16	9600

Table 2.3: Three variable base case

There is an additional constraint that the number of Tables made (sold) can not exceed 200.

A simple LP now is to find the production plan or amount of each of the products that results in the most profit. This will require a new decision variable, Tables, to be added to the model.

Let's extend our previous model. Just to reinforce the point, it is an important habit to very clearly define the decision variables.

- Chairs = # of Chairs to Make
- Desks = # of Desks to Make
- Tables = # of Tables to Make

Our objective function and constraints can now be written as the optimization model showed on the right.

Let's now implement the three variable case.

We have already loaded the required packages so it is not necessary to reload them and we can proceed directly into setting up the model.

$$
\begin{aligned}
\text{Max: } & 20Chairs + 14Desks + 16Tables \\
\text{S.t.: } & \\
& 6Chairs + 2Desks + 4Tables \leq 2000 \\
& 8Chairs + 6Desks + 4Tables \leq 2000 \\
& 6Chairs + 4Desks + 8Tables \leq 1440 \\
& 40Chairs + 25Desks + 16Tables \leq 9600 \\
& Tables \leq 200 \\
& Chairs, Desks, Tables \geq 0
\end{aligned}
$$

Formula 2.2: Optimization model adding a third product(variable)

```
model1 <- MIPModel() %>%
  add_variable(Chairs, type = "continuous", lb = 0) %>%
  add_variable(Desks, type = "continuous",lb = 0) %>%
  add_variable(Tables, type = "continuous", lb = 0) %>%

  set_objective(20*Chairs + 14*Desks + 16*Tables,"max")%>%

  add_constraint(6*Chairs + 2*Desks + 4*Tables<=2000)%>%
  #fabrication
  add_constraint(8*Chairs + 6*Desks + 4*Tables<=2000)%>%
  #assembly
  add_constraint(6*Chairs + 4*Desks + 8*Tables<=1440)%>%
  #machining
  add_constraint(40*Chairs + 25*Desks + 16*Tables<=9600)%>%
  #wood
  add_constraint(Tables <= 200)  #

result1 <-  solve_model(model1, with_ROI(solver="glpk"))
```

2.5 Three Variable Case Results and Interpretation

Let's check to see the status of the solver. Did it find the optimal solution?

```
print(solver_status(result1))
```

```
## [1] "optimal"
```

Again, the LP solver found an optimal solution, let's now extract the solution values found.

```
get_solution(result1, Chairs)
```

```
## Chairs
##    160
```

```
get_solution(result1, Desks)
```

```
## Desks
##   120
```

```
get_solution(result1, Tables)
```

```
## Tables
##      0
```

This is interesting - even though we added a new profitable product to our portfolio, the production decision stayed the same.

2.6 Linear Programming Special Cases

There are several special cases where a linear program does not give the simple unique solution that we might expect. These are:

- No feasible solution
- Multiple optima
- Redundant constraint
- Unbounded solution

Now, let's look at how we would modify the earlier formulation to come up with each of these situations.

Case 1: No Feasible Solution

Let's assume that the sales manager comes in and says that we have a contractual requirement to deliver 300 Chairs to customers.

This results in the LP on the right.

Now let's extend our formulation with this change.

$$\begin{aligned}
\text{Max: } & 20Chairs + 14Desks + 16Tables \\
\text{S.t.:} & \\
& 6Chairs + 2Desks + 4Tables \leq 2000 \\
& 8Chairs + 6Desks + 4Tables \leq 2000 \\
& 6Chairs + 5Desks + 8Tables \leq 1440 \\
& 40Chairs + 25Desks + 16Tables \leq 9600 \\
& Tables \leq 200 \\
& Chairs \geq 300 \\
& Desks, Tables \geq 0
\end{aligned}$$

Formula 2.3: Optimization model for a Non Feasible solution

```
model1infeas <-
  add_constraint(model1, Chairs >= 300)
  #THIS IS THE NEW CHANGE

result1infeas <- solve_model(model1infeas,
                             with_ROI(solver = "glpk"))
```

In this case, we are going to simply add the new constraint to `model1` and create a new model, `model1infeas` to solve.

Note that the constraint on the minimum number of chairs could also be implemented by changing the lower bound on the chairs variable to be 300 instead of zero. Also, the maximum limit on Tables could be set as an upper bound on the Tables variable.

```r
print(solver_status(result1infeas))
```

```
## [1] "infeasible"
```

```r
get_solution(result1infeas, Chairs)
```

```
## Chairs
##    240
```

```r
get_solution(result1infeas, Desks)
```

```
## Desks
##     0
```

```r
get_solution(result1infeas, Tables)
```

```
## Tables
##      0
```

Notice that since the solver status was infeasible, the values for the decision variables are not feasible and therefore cannot be considered a reliable (or possible) production plan. This highlights why the solver's status should always be confirmed to be "Optimal" before results are discussed.

Case 2: Multiple Optima

There are a couple of ways of creating situations for multiple optima. One situation is to have a decision variable be identical or a linear multiple of another variable. In this case, each table now consumes exactly half of the resources as a desk and generates half the profit of a desk. To clarify the objective function value, I will define a variable for Profit. Add a constraint to set the value of profit to what had been the objective function, and now simply maximize profit. This will allow me to find the objective function value by using the `get_solution` for the variable profit.

$$
\begin{aligned}
\text{Max: } & Profit = 20Chairs + 10Desks + 16Tables \\
\text{S.t.:} & \\
& 6Chairs + 3Desks + 4Tables \leq 2000 \\
& 8Chairs + 4Desks + 4Tables \leq 2000 \\
& 6Chairs + 3Desks + 8Tables \leq 1440 \\
& 40Chairs + 20Desks + 16Tables \leq 9600 \\
& Tables \leq 200 \\
& Chairs, Desks, Tables \geq 0
\end{aligned}
$$

Formula 2.4: Optimization model for Multiple Optima solution

The new LP is shown in the formulation on the right.

The implementation can be simplified again since we are only changing the objective function, let's change the objective function in `model1` and save it to `model2a`.

```r
model2a <- set_objective(model1,
20*Chairs + 10*Desks + 16*Tables, "max")
result2a <-  solve_model(model2a,
with_ROI(solver = "glpk"))
```

```r
print(solver_status(result2a))
```

```
## [1] "optimal"
```

```r
res2a <- cbind(objective_value (result2a),
           get_solution(result2a, Chairs),
           get_solution(result2a, Desks),
           get_solution(result2a, Tables))
colnames(res2a)<-list("Profit","Chairs","Desks","Tables")
rownames(res2a)<-list("Solution 2a")
```

```r
grid.table(res2a)
```

	Profit	Chairs	Desks	Tables
Solution 2a	4800	240	0	0

Table 2.4: Results 2a - A First of Multiple Optima

Okay. When I ran it, all the production was focused on Chairs. I think that there is an alternate solution producing Desks with the same total profit. The LP engine won't necessarily tell you that there is an alternate optimal solution. Let's see if we can "trick" the LP to show an alternate solution by disallowing the previous solution by setting Chairs=0.

```r
model2b <- add_constraint(model2a, Chairs <= 0)
            # FORCING LP TO FIND A DIFFERENT SOLUTION
result2b <- solve_model(model2b, with_ROI(solver = "glpk"))
print(solver_status(result2b))
```

```
## [1] "optimal"
```

```r
res2b <- cbind(objective_value(result2b),
               get_solution(result2b, Chairs),
               get_solution(result2b, Desks),
               get_solution(result2b, Tables))
colnames(res2b)<-list("Profit","Chairs","Desks","Tables")
rownames(res2b)<-list("Solution 2b")
grid.table(res2b)
```

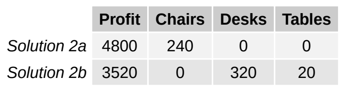
	Profit	Chairs	Desks	Tables
Solution 2b	3520	0	320	20

Table 2.5: Results 2b - A Second of Multiple Optima

Again, only one product is made but now it is Desks instead of Chairs. The number of Desks made is now double the number of Chairs previously made. The total profit is the same. This is an instance of multiple optima.

Let's summarize these two results more clearly by displaying them in a single table.

```r
res2c <- rbind(res2a,res2b)
grid.table(res2c)
```



	Profit	Chairs	Desks	Tables
Solution 2a	4800	240	0	0
Solution 2b	3520	0	320	20

Table 2.6: Results 2c - A Second of Multiple Optima

Which solution is the best? Within the limits of this problem, we can't distinguish between them and they are equally good. If an application area expert or the end decision maker prefers one solution over the other, there should be extra information that we can use to extend the model.

Also, it should be note that when there are two alternate optimal in a linear program with continuous variables, there is actually an infinite number of other optimal solutions between them.

Case 3: Redundant Constraint

For the redundant constraint, a new constraint for painting is created. Let's assume each item is painted and requires one gallon of paint. We have 1500 gallons. The corresponding constraint is then added to the model.

Now we can implement the model.

$$\begin{aligned}
&\text{Max:}20Chairs + 14Desks + 16Tables \\
&\text{S.t.:} \\
&\quad 6Chairs + 2Desks + 4Tables \leq 2000 \\
&\quad 8Chairs + 6Desks + 4Tables \leq 2000 \\
&\quad 6Chairs + 5Desks + 8Tables \leq 1440 \\
&\quad 40Chairs + 25Desks + 16Tables \leq 9600 \\
&\quad Chairs + Desks + Tables \leq 1500 \\
&\quad Tables \leq 200 \\
&\quad Chairs, Desks, Tables \geq 0
\end{aligned}$$

Formula 2.5: Optimization model with a redundant constraint

```r
model1redund <- add_constraint(model1,
                               Chairs + Desks + Tables
                               >= 300)
          #THIS IS THE NEW CHANGE
```

```r
result3 <- solve_model(model1redund,
                       with_ROI(solver = "glpk"))
print(solver_status(result3))
```

```
## [1] "optimal"
```

```r
get_solution(result3, Chairs)
```

```
##   Chairs
## 106.6667
```

```r
get_solution(result3, Desks)
```

```
##    Desks
## 186.6667
```

```r
get_solution(result3, Tables)
```

```
##   Tables
## 6.666667
```

This constraint was *redundant* because the other constraints would keep us from ever having 1500 pieces of furniture or therefore needing 1500 gallons of paint. In other words, there is no way that this constraint could ever be binding at any solution regardless of what the objective function is. More precisely, elimination of a redundant constraint does not change the size of the feasible region at all.

Note that not all non-binding constraints at an optimal solution are redundant. Deleting a non-binding constraint and resolving won't change the optimal objective function value. On the other hand, for a different objective function, that non-binding constraint might become binding and therefore different solutions would be found if it were deleted.

Challenge 1: Can you use a calculator to simply estimate the maximum number of Chairs that could be made? Desks? Tables?

Challenge 2: How would you modify the formulation to find the most pieces of furniture that could be produced?

Case 4: Unbounded Solution

As with other cases, there are multiple ways of triggering this condition. For the unbounded solution, instead of at *most* a certain amount of resources can be used, the constraints are changed to at *least* that amount of each resource must be used. This doesn't make a lot of sense in the setting of this application. Perhaps a cynic would say that in a cost-plus business arrangement or a situation where the factory manager has a limited purview and doesn't see issues such as downstream demand limits and cost impacts, it results in this kind of myopic perspective. More commonly, an unbounded solution might be a sign that the analyst had simply reversed one or more inequalities or the form of the objective (max vs. min).

The implementation simply requires changing a $<$ to a $>$ for each constraint as well.

$$
\begin{aligned}
\text{Max } & 20 Chairs + 14 Desks + 16 Tables \\
\text{S.t.:} & \\
& 6 Chairs + 2 Desks + 4 Tables \geq 2000 \\
& 8 Chairs + 6 Desks + 4 Tables \geq 2000 \\
& 6 Chairs + 5 Desks + 8 Tables \geq 1440 \\
& 40 Chairs + 25 Desks + 16 Tables \geq 9600 \\
& Tables \geq 200 \\
& Chairs, Desks, Tables \geq 0
\end{aligned}
$$

Formula 2.6: Optimization model with an unbounded solution

```
result4 <- MIPModel() %>%
 add_variable(Chairs, type = "continuous", lb = 0) %>%
 add_variable(Desks, type = "continuous", lb = 0) %>%
 add_variable(Tables, type = "continuous", lb = 0) %>%

 set_objective(20*Chairs + 14*Desks + 16*Tables,"max")%>%

 add_constraint(6*Chairs + 2*Desks + 4*Tables>= 2000) %>%
    #fabrication
 add_constraint(8*Chairs + 6*Desks + 4*Tables>= 2000) %>%
    #assembly
 add_constraint(6*Chairs + 4*Desks + 8*Tables>= 1440) %>%
    #machining
 add_constraint(40*Chairs + 25*Desks + 16*Tables>= 9600)%>%
    #wood

 add_constraint(Tables >= 200) %>%

 solve_model(with_ROI(solver = "glpk"))
```

```
print(solver_status(result4))
```

```
## [1] "infeasible"
```

```
get_solution(result4, Chairs)
```

```
## Chairs
##    200
```

```
get_solution(result4, Desks)
```

```
## Desks
##     0
```

```
get_solution(result4, Tables)
```

```
## Tables
##    200
```

The solver status reports that the problem is *infeasible* rather than *unbounded* but by inspection, the LP is feasible. A value for *Tables = 1000, Chairs=Desks=0,* satisfies all of the constraints and therefore the LP is feasible.

Infeasible vs Unbounded. This is a known issue in ompr as of 0.8.0 reported on github. It is caused by not distinguishing between the different status conditions that can result in a failed optimization. The result is that you should read the ompr status of “infeasible” to indicate no optimal solution.

This is another good reminder that it is important to always check the status of the solver.

2.7 Abstracting the Production Planning Model

We have explicitly created two variable model and a three variable model by naming each variable independently. This process doesn’t scale well for companies with dozens, hundreds, or thousands of different products. Simply writing out the full linear program gets very tedious, hard to read, and even maintain. An application for a company with a thousand products and a thousand resources would have a million terms. Assume that variables are on average seven letters long, each resource consumed is a single digit whole number and a plus symbol is used to add terms together and no spaces. This means that there will be (7+1)*1000+999=8999 characters in each line before the inequality. Just say each constraint corresponds to 9000. If a line has 60 characters, this would mean 150 lines or around two pages for each resource (constraint.) The 1000 resources would correspond to about 2000 pages, along with an objective function, and non-negativity constraints. All in all, this single model would make for some rather dry reading.

In practice, people don't write out the full LP explicitly for large models. This includes journals, no journal's page limit would be able to accommodate the above explicit linear program even if readers had the patience to wade through the model.

Rather than writing out models explicitly, instead we should express them algebraically. The products are numbered instead of given names: Chairs, Desks, and Tables become products 1, 2, and 3 respectively.

We could view the naming conventions of the variables as going from least to most generalized:

- Chairs, Desks, Tables
- Product1, Product2, Product3
- X1, X2, X3
- X[1], X[2], X[3]
- X_1, X_2, X_3

The second to last, X[1], one allows us to simply use a vector of X where we can use each element of the vector for each of the products to use. This connects in very well with the data structures available to us in R (and other languages.) It would also allow us to handle any number of products. If we had a thousand products, the thousandth product is simply X[1000].

The last one uses subscripting to more succinctly and compactly express the same concept.

Similarly, the resources: Fabrication, Assembly, Machining, and Wood resources are numbered as 1, 2, 3, and 4 respectively. Note that we do not need to separate the resources by units, the first three are in units of hours while the last is in units of square feet of wood.

Here we are talking about abstracting the model in terms of variable names and notation. In the next chapter we will continue with generalizing the model's number of products and resources.

2.8 Advice on Homeworks

- You can talk with classmates or colleagues but your markdown should be your own.
- The D2L Learning Management System has had issues with uploading HTML files. Instead use PDFs. Upload both the Rmd and PDF versions of your file.
- Include your name as the author.
- Explain your model or modifications and interpret the results. I explicitly had a subsection for each formulation, implementation, and results/interpretation. Frequently people try to mix all three together and this makes it very hard to help or debug. Also, formulations should be understood before moving into implementation.
- If the solution does not make sense, acknowledge and explain.
- Always show your LP. It is best to show it algebraically as the problems get bigger. This will also pay off for most projects where the models become larger.
- Discussion of results does not need to be long but this can be an interesting part of any paper.

2.9 Comments specific to R

- Using the LaTeX equations and rendering it in rmarkdown is helpful. Getting the first one written is sometimes tricky but then it is just a matter of cut and paste.
- Installing LaTeX allows knitting to PDF which may be good for readability and turning in but please be sure to also turn in the .rmd file.
- The R markdown documents (*.rmd files) may cause problems with the previews in the D2L learning management system.
- Using R markdown documents allows you to mix both analysis and interpretation cleanly.
- Look over this .rmd file for information and try knitting it to HTML and PDF on your computer. If you have everything installed correctly, it should work fine including the mathematical notation. If it doesn't work, you may need to do a little extra free software installation.

In particular, you might want to look over this document with respect to:

- How to embed a linear programming formulation
- How to denote proper subscripts in text. Ex. a dollarsign-x-underscore-i-dollarsign becomes x_i
- Double subscripts such as $R_{i,j}$ would be done by replacing the i with i,j surrounded by curly brackets
- Summations are a little tricky in both creating equations and the *ompr* model. For the former, you can just emulate my material to learn enough LaTeX to make it work. For the latter, look at Dirk's online documentation for *ompr* or chapter 2 of my book, *DEA Using R.*
- Organizing information in Desks for display is helpful. Raw output can be verbose. I hard coded a table at the beginning that works for simple data in explicit LPs. For richer data models and nicer Desks, see Chapter 2 again. The pander package makes it easy to display matrices and data Desks nicely in HTML and PDF outputs.
- Use of section and subsection headings to organize your writeups.

ompr, Model mixed integer linear programs in an algebraic way directly in R. The model is solver-independent and thus offers the possibility to solve a model with different solvers. It currently only supports linear constraints and objective functions. See the 'ompr' website `https://dirkschumacher.github.io/ompr` for more information, documentation and examples

Exercise 2.1 (Adding Frames).

Your company has extended production to allow for producing picture frames and is now including a finishing department that primes and paints (or stains) the furniture. See Table 2.7 on the right.

a) Use R Markdown to create your own description of the model.

b) Extend the R Markdown to show your LP Model.Be sure to define models.

c) Solve the model in R.

d) Interpret and discuss the model in R Markdown.

e) Discuss how one parameter would need to change in order to result in a different production plan. Demonstrate how this affects the results.

Hint: Knit your RMarkdown as a PDF or open the HTML version in your browser and print to PDF.

Characteristic	Chairs	Desks	Frames	Tables
Profit	$20	$14	$3	$16
Fabrication	6	2	1	4
Assembly	8	6	1	8
Machining	6	4	1	25
Painting	7	10	2	12
Wood	40	25	5	16

Table 2.7: Exercise 2.1

2.10 Graphically Solving LPs with R

This example is based on work by Christopher Davis. Christopher is the author of *Agile Metrics in Action* from Manning Press (Davis 2015).

The goal of this section is to show linear programming graphically using R. Unfortunately there isn't a library that we can use to create a nice chart in R for our problems. To make this work there are a few steps we'll have to follow to take our problem and turn it into something that we can easily plot in R. Here is a brief outline, all of these steps are expanded on below.

1) Re-write our constraints to solve for Y. We have to do this in order to plot them on our graph.
2) Generate the points for the chart. To do this we'll use the function that we create in step 1 to create the points used for our lines.
3) Find the intersection points. Unfortunately there isn't a nice function that can tell us where lines intersect on our graphs so we'll need to put a simple function together that figures out so we can plot it.
4) Now that we have all the data we need in graphable format, we can plot it!

Let's make up a simple example where we want to build robots - because I love to build robots! There are 2 options, xbot (x) and ybot (y). It costs \$40 in material to build an xbot and \$60 in material to build a ybot. There is a maximum of \$7400 available for investment. There are 3300 hours of labor available. It takes 20 labor hours to build an xbot and 25 labor hours to build ybot. If an xbot is put into the field it will make \$150/hour, while a ybot in the field can generate \$200/hour. If we want to maximize profit, what do we do? First let's put the simple problem together:

$$\begin{aligned}&\text{Max:} 150x + 200y\\&\text{s.t.:}\\&\qquad 40x + 60y \leq 7400\\&\qquad 20x + 25y \leq 3300\\&\qquad x, y \geq 0\end{aligned}$$

Formula 2.7: Optimization model for Graphically Solving LPs with R exercise

Create a data series to plot

To graph a line in R you need to have a series of coordinates, you can't just input a formula and see it on a chart. Here are some steps to help turn your constraints into data series.

Reduce the constraints

To convert our constraints into a series of numbers the first thing we want to do is update the formulas to solve for y. Let's take a look at the constraints in this problem. We simply do a little rearranging of terms for the first constraint to get y on its own on the left hand side.

$$\begin{aligned}40x + 60y &\leq 7400\\ y &\leq \frac{7400 - 40x}{60}\end{aligned}$$

Formula 2.8: Update formalation for constraint 1.

We do the same thing for the second constraint.

$$\begin{aligned}20x + 25y &\leq 3300\\ y &\leq \frac{3300 - 25x}{25}\end{aligned}$$

Formula 2.9: Update formalation for constraint 2.

Next we can turn those into functions. Rather than inequalities, treat them as equalities so that we can find the edge defined by this constraint.

Note that R makes it very easy to define a function! In this case, the first function will take a value of `x` and return the value of `y` for the first constraint.

```r
r1y <- function(x) 7400 / 60 - 40 / 60 * x
r2y <- function(x) 3300 / 25 - 20 / 25 * x
```

Create the series

Secondly we will create an artificial series for x to plug into our functions, then create a data frame with a column for each series. Let's populate the data frame of possible plans of xbots `x1`. The `seq` function is very handy here in that it will generate a sequence of values from *0* to *200* xbots.

```
# create the series
x1 = seq(0,200)
tablex1 <- head(as.matrix(x1))
grid.table(tablex1)
```

0
1
2
3
4
5

Table 2.8: Beginning of Sequence x1

The next statement takes a little unpacking to fully understand and illustrates another strength of R. Now, we would like to evaluate each of these 201 values for `x1` using the functions for the two constraints that we just defined. We could use a `for-next loop` if we were using a language such as BASIC but R handles these operations more smoothly and efficiently. We can pass the data frame for `x1` into the function and R is smart enough to return the results in the same size data frame, `y1`. Of course we do the same thing for the second function (constraint).

```
# 2 constraints in this equation so there are 2 y variables,
# y1 and y2.
# If you have more constraints you can just keep adding them
# here.
seriesDf = data.frame(x1, y1=r1y(x1), y2=r2y(x1))
# take a look at the generated data
grid.table(format(head(seriesDf),digits=4))
```

	x1	y1	y2
1	0	123.3	132.0
2	1	122.7	131.2
3	2	122.0	130.4
4	3	121.3	129.6
5	4	120.7	128.8
6	5	120.0	128.0

Table 2.9: Some datapoints from the lines for the constraints

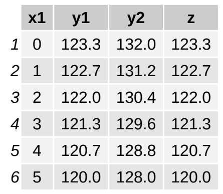
Note that we now have coordinates for each series. If you have more than 2 constraints you can just continue to add columns to your data frame.

To show our feasible region on a graph the x and y coordinates are not enough, we also need to determine the area of the region we want to shade in to show the region. To do this we really just need to get the minimum of y1 or y2. Note this is for a maximization problem, if we were looking at a minimization problem you would change 'pmin' to 'pmax', which would shade the region above the line rather than below the line. If this doesn't make sense at this point, it will when you see the graph.

```
library(ggplot2)

head(seriesDf)
```

```
##   x1       y1    y2
## 1  0 123.3333 132.0
## 2  1 122.6667 131.2
## 3  2 122.0000 130.4
## 4  3 121.3333 129.6
## 5  4 120.6667 128.8
## 6  5 120.0000 128.0
```

```
  # Add the Z region for shading
seriesDf <- transform(seriesDf, z = pmin(y1,y2))
  # take a look at the generated data
grid.table(format(head(seriesDf),digits=4))
```



	x1	y1	y2	z
1	0	123.3	132.0	123.3
2	1	122.7	131.2	122.7
3	2	122.0	130.4	122.0
4	3	121.3	129.6	121.3
5	4	120.7	128.8	120.7
6	5	120.0	128.0	120.0

Table 2.10: Some datapoints from the lines for regions

Finding the intersection of constraints

To find the optimal value we will look for the intersection of the constraints. Unfortunately in R there isn't a nice easy way to get this so we'll need to write a function that takes in our lines and returns the x,y coordinates of their intersection.

```
# this function returns a data frame with the intersection
# point of 2 lines pass in the model for each line
lmInt <- function(line1, line2) {
  b1<- line1$coefficient[1]  #y-int for line1
  m1<- line1$coefficient[2]  #slope for line1
  b2<- line2$coefficient[1]  #y-int for line2
  m2<- line2$coefficient[2]  #slope for line2
  x <- (b2-b1)/(m1-m2)       #solved general equation for x
  y <- m1*x + b1             #plug in the result
  data.frame(x=round(x, 2), y=round(y, 2))
  #create a data frame with x and y
}
```

Next we can use this function to find the intercept of these lines:

```
# find the intersect of the lines
intersection <- lmInt(lm(seriesDf$y1~seriesDf$x1),
lm(seriesDf$y2~seriesDf$x1))
head(intersection)
```

```
##              x  y
## (Intercept) 65 80
```

At this point we have everything we need to see our results on a graph. There are a few ways to build graphs in R but one very popular and powerful library is *ggplot*.

ggplot was written by Hadley Wickham to implement the *Grammar of Graphics*. That book is kind of expensive, a more affordable intro would be Hadley's overview.

In this example we are going to just scratch the surface of ggplot functionality, so know that if you study up a bit you can do much more!

```
#make a plot with PSU colors for the lines!
ggplot(seriesDf, aes(x = x1)) +
  geom_line(aes(y = y1), color = 'white') +
  geom_line(aes(y = y2), color = 'green') +
  geom_ribbon(aes(ymin=0,ymax = z), fill = 'gray60') +
  geom_point(x = intersection$x, y = intersection$y)
```

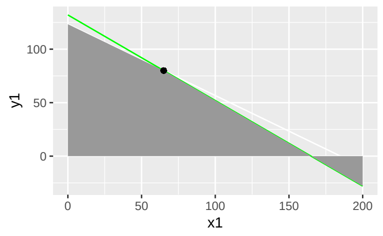


Figure 2.1: Example of ggplot

Awesome, it looks good! Though PSU colors don't work so well, that white is hard to see.

Now let's solve the model for real to see if it's right!

```
result0 <- MIPModel() %>%
  add_variable(R1, type = "continuous", lb = 0) %>%
  add_variable(R2, type = "continuous",lb = 0) %>%

  set_objective(150*R1 + 200*R2, "max") %>%

  add_constraint(40*R1 + 60*R2 <= 7400) %>%
  add_constraint(20*R1 + 25*R2 <= 3300) %>%
  solve_model(with_ROI(solver = "glpk"))
print(solver_status(result0))
```

```
## [1] "optimal"
```

```
print(objective_value(result0))
```

```
## [1] 25750
```

```
print(get_solution(result0, R1))
```

```
## R1
## 65
```

```
print(get_solution(result0, R2))
```

```
## R2
## 80
```

Same results as our graph, great!

Putting it all together in a more complex example

Now that you've seen all of these steps let's change the scenario and put all our code in one place. Let's still say we know we want to maximize profit and each robot can still make the same amount of money. However there are new constraints to the model.

$$\begin{aligned}&\text{Max:}150x + 200y\\&\text{S.t.:}\\&\quad 5x + 3y \geq 210\\&\quad x + y \leq 110\\&\quad 4x + y \leq 200\\&\quad x, y \geq 0\end{aligned}$$

Formula 2.10: More complex formulation

```r
# first translate the constraints:
# 5*x + 3*y >=  210    becomes y = 210/3 - 5/3*x
# x + y <= 110         becomes y = 110 - x
# 4*x + y <= 200       becomes y = 200 - 4*x
constraint1 <- function(x) 210/3 -5/3*x
constraint2 <- function(x) 110 -x
constraint3 <- function(x) 200 -4*x
x2 = seq(0,100)
dataFrameTwo = data.frame(x2, y1=constraint1(x2),
                          y2=constraint2(x2) ,
                          y3=constraint3(x2))
# because constraints are mixed leq and geq,
#we take the min of y2 and y3
dataFrameTwo <-  transform(dataFrameTwo,
                           z = pmax(y1,pmin(y2,y3)))
# get the intersections of the lines
intersection1 <-
  lmInt(lm(dataFrameTwo$y3~dataFrameTwo$x2),
        lm(dataFrameTwo$y2~dataFrameTwo$x2))
intersection2 <-
  lmInt(lm(dataFrameTwo$y3~dataFrameTwo$x2),
        lm(dataFrameTwo$y1~dataFrameTwo$x2))
#plot it
ggplot(dataFrameTwo, aes(x = x2)) +
  geom_line(aes(y = y1), color = 'aquamarine', size = 2) +
  #you can change line size
  geom_line(aes(y = y2), color = '#99ffbb', size = 2) +
  #you can use hex colors
  geom_line(aes(y = y3), color = 'white', size = 2) +
  geom_ribbon(aes(ymin=y1,ymax = z), fill = 'gray80') +
  #you can change alpha on colors
  geom_point(x = intersection1$x,
             y = intersection1$y,color= "green",size = 5)+
  geom_point(x = intersection2$x,
             y = intersection2$y, color = "red")
```

Note now our chart looks much different because we have combination of lesser than and greater than constraints.

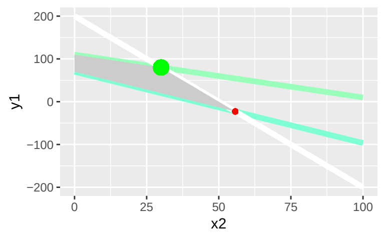


Figure 2.2: Example of ggplot (2)

2.11 Next Steps

Note that we only used 2 variables, so we were able to see this in a 2 dimensional space. If you add another variable to equation you will have to map this in a 3 dimensional chart which you may want to try a different charting library for like plotly. If you have more variables than that using the graphical method will become very hard!

Plotly R graphing library makes interactive, publication-quality graphs online. Examples of how to make line plots, scatter plots, area charts, bar charts, error bars, box plots, histograms, heatmaps, subplots, multiple-axes, and 3D (WebGL based) charts.

This section on graphically exploring linear programming is a great first start and may be further refined in future versions of this book. Challenges for the future include:

- Adding more annotations to the graphs
- Adjusting colors
- Adding more aesthetics
- Refining limits for the plotting area to be non-negative.

2.12 Miscellaneous Tips and Tricks

At this point, you are able to build your own optimization model. Here are a collection of tips and tricks that have confounded past students.

2.13 `ompr` Related Tips and Tricks

- `ompr` will complain if you use the same variable name a data object in R and in `ompr`. For example, if you have variables x and y in a formulation, then build an `ompr` model with `x` and `y` variables. You might then assign the results to `x` and `y` objects in R, and run it again. You will then get a notification of a name space conflict from `ompr`. One work around is to prefix the `ompr` variables with a V to indicate variable thereby using `Vx` and `Vy` in `ompr` for is assigned to x and y in r.
- Typos can happen easily in piped models being built but are not typically localized. Try commenting and in and lines until the problem goes away. This will help to identify where the problem is.
- Check for ompr updates. It is under active development. You may find versions on the author's github repository that are not yet available through CRAN.
- `ompr` is featuring a much faster form of model building with a different syntax using `MILPmodel` rather than "MIPmodel". If you are running into delays running ompr but not solving the problem, this may be worth investigating.

2.14 RMD Tips and Tricks

RMarkdown is very helpful for integrating analysis and results but the code can also be brittle. Knitting this full book can be stopped by one small error.

- Make sure to get your header information correct. Small typos can cause problems preventing knitting from occuring.
- Name your code chunks uniquely. Not naming code chunks can make it harder to find errors or to navigate complex documents. Once you start naming code chunks though and copying code chunks for building more complex models, you run into duplicate code chunk problems. Each name must be unique.
- Use code chunk options smartly. Look over the list of common options such as `eval` and `echo` are particularly helpful.
- Remember to put a blank line before a bulleted or numbered list.
- Putting two blank spaces at the end of a paragraphy will give a little more spacing between paragraphs.
- Sometimes running all chunks will make more visible where a problem is than when an RMD is knitted.

2.15 RStudio Tips and Tricks

- RStudio.cloud can be a very helpful service sometimes it can help be a test to see if a problem is with a personal software installation. Some things will `knit` on RStudio.cloud that do not knit on local computer.
- Installing LaTeX before RStudio resolves some problems with knitting. If you have problems with knitting documents to PDF, even the simple "New RMarkdown" document. Try uninstalling RStudio, install a LaTeX system (ex. MikTeX) if one is not already installed, and reinstalling RStudio.
- Use the RStudio cheat sheets!

2.16 General R Tips and Tricks

- If you are getting an error message read it carefully. You may also be able to google generic parts that do not reference your own R code to get clues.
- R is case sensitive. Be careful in watching for typos regarding case. A student spent many hours trying to find why a document was no longer knitting before discovering that `echo=FALSE` in one code chunk was typed in as `echo=False`.
- Many problems might be due to R data objects now having the data as you think it is. Displying the data or a part using the

`head()` or `tail()` functions can be helpful for debugging.

2.17 LaTeX Tips and Tricks

- Find good, working LaTeX formulations and then reuse and change it to fit the situation.
- Using inline LaTeX is helpful and easy.

2.18 General Tips and Tricks

- After spending *too much* time stuck on a problem, get up walk away, and look at it freshly.
- Join an R User's Group.
- When you learn something new or interesting for R or related items, write it down in a Tips and Tricks section.

3
More Linear Programming Models

3.1 Types of LP Models

In this chapter, we will examine a range of classic applications of linear programs. These applications will give ideas for how to model a variety of situations. In each case, try to follow along with the application.

3.2 The Algebraic Model

In the previous chapter, we examined situations with only a few products and constraints. In general, most companies have many more products and we won't be wanting to name each product explicilty and uniquely. Instead, we use sets of products and resources.

We could also reframe the model more algebraically. Let's use subscripts to differentiate between products and resources. We can define that *i=1* represents Chairs, *i=2* represents Desks, and *i=3* represents Tables.

Similarly, *j=1* represents fabrication, *j=2*, represents assembly, etc. Now, let's move on to defining the data. Let's define the amount to produce of each product, *i*, as x_i and resource j consumed by product i as $R_{i,j}$. The available resource j is then A_j. The profit per product i is then P_i. The LP can now be rewritten as showed on the right margin, formula 3.1, if there are three products and four constraints.

$$\begin{aligned} \text{Maximize } & \sum_{i=1}^{3} P_i x_i \\ \text{subject to } & \sum_{i=1}^{3} R_{i,j} x_i \leq A_j \text{ , } j = 1,2,3,4 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

Formula 3.1: Linear MOdel with 3 products and 4 contraints

We could further generalize this by instead of hard coding the number of products and resource constraints, we instead define the number of products and resources as NProd and NResources respectively. We can then rewrite the above formulation as the following.

$$\begin{aligned} \text{Max: } & \sum_{i=1}^{NProd} P_i x_i \\ \text{subject to } & \sum_{i=1}^{NProd} R_{i,j} x_i \leq A_j \; \forall \; j \\ & x_i \geq 0 \; \forall \; i \end{aligned}$$

Formula 3.2: Linear MOdel with 3 products and 4 contraints. Not hardcoding the products and constraints.

3.3 Tips and Conventions for Algebraic Models

The symbol, ∀, is read as "for all" and can be interpreted to mean "repeat this line substituting in a value for this subscript **for all** possible values of this subscript." In other words, given that i is used consistently for the three products, then $x_i \geq 0 \ \forall \ i$ is equivalent to $x_i \geq 0 \ , i=1,2,3$ or even $x_1 \geq 0, \ x_2 \geq 0, \ x_3 \geq 0$. The result is that the ∀ symbol can simplify the description of complex models when index ranges are clear.

Also, another good practice is to use a mnemonic to help suggest the meaning of data. That is why I chose "R" for Resource, "P" for Profit, and "A" for Available resources. Another helpful convention is

to use capital letters for data and lower case letters for variables. (Some people will swap this around and use capital letters for variables and lower case letters for data - it doesn't really matter as long as a model is consistent.)

More complex models often run out of letters that make sense. A common approach in these models is to use a superscript. For example, perhaps labor cost for each worker, w, could have different values for both normal and overtime rates. Rather than separate data terms for such closely related concepts, we might denote regular hourly labor cost for worker w as C_w^R and for overtime as C_w^O. Again, it is very important to clearly define all the data and variables used in the model.

3.4 Building the Generalized Model in R

This concise, algebraic representation can be easily scaled to any number of products and resources. Let's write this as the following. I'll expand the names of data slightly for making the R code more readable but this is meant to be consistent with the above formulation.

	Prod1	Prod2	Prod3	Prod4
Profit	20	14	3	16

Table 3.1: Profit per product

We will define the number of products as four and five resource constraints.

```
NProd <- 4
NResources <- 5
ProdNames <- lapply(list(rep("Prod",NProd)),paste0,1:NProd)
                    # Product names: Prod1, Prod2, ...
ResNames<- lapply(list(rep("Res",NResources)),
                     paste0,1:NResources)
                    # Resource names: Res1, Res2, ...
Resources <- matrix(c( 6, 8, 6, 7 , 40, 2, 6, 4, 10,
                       25, 1, 1, 1, 2, 5, 4, 8, 25, 12, 16),
                  ncol=NProd,dimnames=c(ResNames,ProdNames))
Available <- matrix(c(1440, 1440, 2000, 1000, 9600),
                  ncol=1,dimnames=c(ResNames,"Available"))
```

On the right side we can see the data. This should match the data that we hard coded into the R linear programming model in the previous chapter.

Similarly, we can display the resources used by each product and the amount of each resource available. See both tables on the right. Table 3.2 displays the resources used by each product and Table 3.3 the amount of resources available from each product.

	Prod1	Prod2	Prod3	Prod4
Res1	6	2	1	4
Res2	8	6	1	8
Res3	6	4	1	25
Res4	7	10	2	12
Res5	40	25	5	16

Table 3.2: Resources Used by Each Product

cbind function: This function join 2 tables using the columns as binding element between tables.

rbind function: Same concept as previous one, but in this case using rows as binding element.

	Available
Res1	1440
Res2	1440
Res3	2000
Res4	1000
Res5	9600

Table 3.3: Amount of Each Resource Available

Now we need to define variables.

```
Combined <- cbind(Resources, Available)
grid.table(Combined)
```

On the right table, we can see the resources used and available in a single table. We have used the *cbind* function to do a column binding of the data. In this way the representation is more visually intuitive.

To ensure that we know how to access the data, if we want to see how the amount of the first resource used by the second product, you can enter *Resources[1,2]* in R Studio's console which is 2.

	Prod1	Prod2	Prod3	Prod4	Available
Res1	6	2	1	4	1440
Res2	8	6	1	8	1440
Res3	6	4	1	25	2000
Res4	7	10	2	12	1000
Res5	40	25	5	16	9600

Table 3.4: Resources Used by Each Product and Available

Notice that the resources available are listed as a column (vertical) vector rather than a row (flat) vector. This differentiation will be important later.

Now, let's begin building our optimization model.

First, we'll start by loading the packages that we are using.

```
library (tufte)
library (magrittr, quietly = TRUE)
suppressPackageStartupMessages(
  library (dplyr, quietly = TRUE))
suppressPackageStartupMessages(
  library (ROI, quietly = TRUE))
library (ROI.plugin.glpk, quietly = TRUE)
library (ompr, quietly = TRUE)
library (ompr.roi, quietly = TRUE)
```

And we will continue wiht the code to build the mode in a generic format.

```r
prodmodel <- MIPModel() %>%
  add_variable (x[i], i=1:NProd,
                type="continuous", lb=0) %>%
  set_objective (sum_expr(Profit[i] * x[i],
                          i=1:NProd ), "max") %>%
  add_constraint (sum_expr(Resources[j,i]*x[i],
                           i=1:NProd)
                  <= Available[j],
                  j=1:NResources) %>%

  solve_model(with_ROI(solver = "glpk"))

prodmodel
```

```
## Status: optimal
## Objective value: 2857.143
```

Let's walk through what is done line by line.

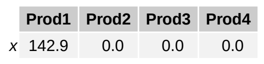
`prodmodel <- MIPModel() %>%`

3.5 Examining the Results

Displaying the object of `prodmodel` only shows a simple summary of the results of the analysis. It indicates whether the model was solved to optimality (and it was!) and the objective function value (profit).

This is useful to know but we are really interested in how to generate this profit. To do this, we need to extract the values of the variables.

```r
results.products <- matrix (rep(-1.0,NProd),
                            nrow = NProd, ncol=1,
                            dimnames=c(ProdNames,c("x")))
temp <- get_solution (prodmodel, x[i])
   # Extracts optimal values of variables
results.products <- t(temp [,3] )
   #Extracts third column
results.products <- matrix (results.products,
                            nrow = 1, ncol=NProd,
                            dimnames=c(c("x"),ProdNames))
                            # Resizes and renames
grid.table(format(head(results.products),digits=4))
```



	Prod1	Prod2	Prod3	Prod4
x	142.9	0.0	0.0	0.0

Table 3.5: Optimal Production Plan

The table on the right displays the Optimal production plan.

Let's examine how the resources are consumed. To do this, we can multiply the amount of each product by the amount of each resource used for that product. For the first product, this would be a term by term sum of each product resulting in `857.1428571` which is less than `1440`. We can do this manually for each product. Another approach is to use the command, `Resources[1,]%*%t(results.products)`. This command will take the first row of the Resources matrix and multiplies it by the vector of results.

Another thing to take note of in learning how to use RMarkdown is that results from r code can be shown by inserting an inline code chunk. Inline code chunks can be inserted into text by using a single tick at the beginning and end of the chunk instead of the triple tick mark for regular code chunks. Also, the inline r code chunk starts with the letter r to indicate that it is an r code code. A common use for this might be to show the results of an earlier analysis.

One thing to note is that the first row of `Resources` is by definition a row vector and `result.products` is also a row vector. What we want to do is do a row vector multiplied by a column vector. In order to do this, we need to convert the row vector of results into a column vector. This is done by doing a *transpose* which changes the row to a column. This is done often enough that that the operation is just one letter, `t`.

We can go one further step now and multiply the matrix of resources by the column vector of production.

```
Results.Resources <- Resources[]%*%t(results.products)
# Multiply matrix of resources by amount of
# products produced
ResourceSlacks <- cbind (Results.Resources,
                         Available,
                         Available-Results.Resources)
colnames(ResourceSlacks)<-c("Used", "Available", "Slack")
grid.table(format(head(ResourceSlacks),digits=4))
```

	Used	Available	Slack
Res1	857.1	1440.0	582.9
Res2	1142.9	1440.0	297.1
Res3	857.1	2000.0	1142.9
Res4	1000.0	1000.0	0.0
Res5	5714.3	9600.0	3885.7

Table 3.6: Resources Used

This section covered a lot of concepts including defining the data, setting names, using indices in *ompr*, building a generalized *ompr* model, extracting decision variable values, and calculating constraint right hand sides. If you find this a little uncomfortable, try doing some experimenting with the model. It may take some experimenting to get familiar and comfortable with this.

3.6 Changing the Model

Let's modify the above model. We can do this by simply changing the data that we pass into the model.

Let's change the time required for painting a chair to 20. Recall that this is the first product and the fourth resource. This is how we can change the value.

```
Resources[4,1] <- 20
# Set value of the 4th row, 1st column to 20
# In our example, this is the paint resource
# used by making a chair
```

Now we will rebuild the the optimization model.

```
prodmodel <- MIPModel() %>%
  add_variable (x[i], i=1:NProd,
                type="continuous", lb=0) %>%
  set_objective (sum_expr(Profit[i] * x[i] ,
                          i=1:NProd ), "max") %>%
  add_constraint (sum_expr(Resources[j,i]*x[i],
                           i=1:NProd)
                  # Left hand side of constraint
                  <= Available[j],
                  # Inequality and Right side of constraint
                  j=1:NResources) %>%
                  # Repeat for each resource, j.
  solve_model(with_ROI(solver = "glpk"))

prodmodel
```

```
## Status: optimal
## Objective value: 1500
```

Note that the objective function has changed.

```
results.products <- matrix (rep(-1.0,NProd),
                            nrow = NProd,
                            ncol=1,
                            dimnames=c(ProdNames,c("x")))
temp <- get_solution (prodmodel, x[i])
   # Extracts optimal values of variables
results.products <- t(temp [,3] )
```

```
    # Extracts third column
results.products <- matrix (results.products,
                            nrow = 1,
                            ncol=NProd,
                            dimnames=c(c("x"),ProdNames))
    # Resizes and renames
grid.table(results.products)
```

	Prod1	Prod2	Prod3	Prod4
x	0	0	500	0

Table 3.7: Revised Optimal Production Plan

The table on the right displays the new Production plan and the optimal distribution of the resources.

##Blending Problems

Specific blend limitations arise in many situations. In our example, we might say that Chairs can make up no more than 10% of the overall product plan. In our original case, this would be expressed as the following.

Note that the production plan has significantly changed. At this point, we could discuss why this makes sense or not.

Blending constraints can appear as nonlinear constraints but can be readily linearized. We can see the formulation on the right side.

Max

$$20 Chairs + 14 Desks + 16 Tables$$

S.t.

$$\begin{aligned} &6 Chairs + 2 Desks + 4 Tables \leq 2000 \\ &8 Chairs + 6 Desks + 4 Tables \leq 2000 \\ &6 Chairs + 4 Desks + 8 Tables \leq 1440 \\ &40 Chairs + 25 Desks + 16 Tables \leq 9600 \\ &Tables \leq 200 \\ &Chairs,\ Desks,\ Tables \geq 0 \end{aligned}$$

Formula 3.3: Chairs can make up no more than 10% of the overall

```
Base3VarModel <- MIPModel() %>%
 add_variable(Chairs, type = "continuous", lb = 0) %>%
 add_variable(Desks, type = "continuous",lb = 0) %>%
 add_variable(Tables, type = "continuous", lb = 0) %>%

 set_objective(20*Chairs + 14*Desks + 16*Tables,"max")%>%

 add_constraint(6*Chairs + 2*Desks + 4*Tables<= 2000)%>%
      #fabrication
 add_constraint(8*Chairs + 6*Desks + 4*Tables<= 2000)%>%
      #assembly
 add_constraint(6*Chairs + 4*Desks + 8*Tables<= 1440)%>%
      #machining
 add_constraint(40*Chairs + 25*Desks + 16*Tables<= 9600)%>%
      #wood
  add_constraint(Tables <= 200) #
res3base <- solve_model(Base3VarModel,
                        with_ROI(solver = "glpk"))
xchairs <- get_solution (res3base , Chairs)
xdesks  <- get_solution (res3base , Desks)
xtables <- get_solution (res3base , Tables)
base_case_res           <- cbind(xchairs,xdesks,xtables)
rownames(base_case_res) <- "Amount to Produce"
grid.table(base_case_res)
```

	xchairs	xdesks	xtables
Amount to Produce	160	120	0

Table 3.8: Production Plan for Base Case

The table on the right displays the optimal production plan for the formulation described in Formula 3.3.

Let's add a constraint that chairs can't make up more than 40% of the total production. Also, Desks can't be more than 50% of total production. Lastly, let's say that Tables can't be more more than 40% of total production.

Effectively, we are saying that $\frac{Chairs}{Chairs+Desks+Tables} \leq 0.40$

Alas, this is not a linear function so we need to clear the denominator.

$Chairs \leq 0.40 * (Chairs + Desks + Tables)$

We like to get all the variables on the left side so let's move them over.

$Chairs - 0.4 * Chairs - 0.4 * Desks - 0.4 * Tables \leq 0$

Let's simplify this a little, which gives us the following:

$0.6 * Chairs - 0.4 * Desks - 0.4 * Tables \leq 0$
The result is displayed as a formula on the right.

$$
\begin{aligned}
&\text{Max:}\\
&\quad 20Chairs + 14Desks + 16Tables\\
&\text{S.t.}\\
&\quad 6Chairs + 2Desks + 4Tables \leq 2000\\
&\quad 8Chairs + 6Desks + 4Tables \leq 2000\\
&\quad 6Chairs + 4Desks + 8Tables \leq 1440\\
&\quad 40Chairs + 25Desks + 16Tables \leq 9600\\
&\quad 0.6Chairs - 0.4Desks - 0.4 * Tables \leq 0\\
&\quad Tables \leq 200\\
&\quad Chairs,\ Desks,\ Tables \geq 0
\end{aligned}
$$

Formula 3.4: Three constraints

```
ModelBlending<- add_constraint(Base3VarModel,
                  0.6*Chairs -0.4*Desks - 0.4*Tables <= 0)
resBlending<- solve_model(ModelBlending,
                             with_ROI(solver = "glpk"))
```

```
blendres<-cbind(get_solution (resBlending, Chairs),
                 get_solution (resBlending, Desks),
                 get_solution (resBlending, Tables))
```

Okay, now let's put both side by side in a table to show the results.

```
rownames(base_case_res)<-"Base Model Plan"
rownames(blendres)<-"with Blend Constraint"
blendingresults <- rbind(base_case_res,blendres)
colnames (blendingresults) <- c("Chairs","Desks","Tables")
comparative <- rbind(base_case_res,blendres)
grid.table(comparative)
```

As a summary, we can see on the table on the right a comparative between the baseline case and the new production plan with the blending constraint.

xchairs	xdesks
160	120
118.260869565217	172.17391304347

Table 3.9: Compare Baseline and Production Plan due to a Blending Constraint

3.7 Allocation Models

An allocation model divides resources and assigns them to competing activities. Typically has a maximization objective with less than or equal to constraints.

See the Formula 3.5 on the right to understand the mathematical expression.

$$\text{Max:} \sum_{i=1}^{3} P_i x_i$$
$$\text{S.t.}$$
$$\sum_{i=1}^{3} R_{i,j} x_i \leq A_j \; \forall \; j$$
$$x_i \geq 0 \; \forall \; i$$

Formula 3.5: Allocation

3.8 Covering Models

A covering model combines resources and coordinates activities. A classic covering application would be what mix of ingredients "covers" the requirements at the lowest possible cost. Typically it has a minimization objective function and greater than or equal to constraints.

See the Formula 3.5 on the right to understand the mathematical expression.

$$\text{Min:} \sum_{i=1}^{3} C_i x_i$$
$$\text{S.t.} \sum_{i=1}^{3} A_{i,j} x_i \geq R_j \; \forall \; j$$
$$x_i \geq 0 \; \forall \; i$$

Formula 3.6: Covering

Consider the case of Trevor's Trail Mix Company. Trevor creates a variety of custom trail mixes for health food fans. He can use a variety of ingredients displayed in the table on the right.

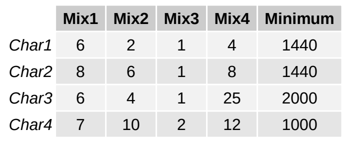
Charact.	Mix1	Mix2	Mix3	Mix4	Min Req
Cost	\$20	\$14	\$3	\$16	
Calcium	6	2	1	4	1440
Protein	8	6	1	8	1440
Carbohyd.	6	4	1	25	2000
Calories	7	10	2	12	1000

Table 3.10: Trevor Trail Mix Company ingredients

Let's go ahead and build a model in the same way as we had done earlier for production planning.

```
NMix <- 4
NCharacteristic <- 4
MixNames <- lapply(list(rep("Mix",NMix)),paste0,1:NMix)
                 # Mix names: Mix1, Mix2, ...
CharNames <-lapply(list(rep("Char",NCharacteristic)),
                   paste0,1:NCharacteristic)
                 # Characteristics of each mix
Cost <- matrix(c(20, 14, 3, 16),
              ncol=NProd,dimnames=c("Profit",MixNames))
MixChar <- matrix(c( 6, 8, 6, 7,
                     2, 6, 4, 10,
                     1, 1, 1, 2,
                     4, 8, 25, 12),
              ncol=4, dimnames=c(CharNames,MixNames))
CharMin <- matrix(c(1440, 1440, 2000, 1000),
              ncol=1,dimnames=c(CharNames,"Minimum"))
```

```
TTMix <-cbind(MixChar,CharMin)
grid.table(TTMix)
```



	Mix1	Mix2	Mix3	Mix4	Minimum
Char1	6	2	1	4	1440
Char2	8	6	1	8	1440
Char3	6	4	1	25	2000
Char4	7	10	2	12	1000

Table 3.11: Data for Trevor Trail Mix Company

Hint: You might need to add a total amount to make! Modify the numbers until it runs...

Now let's build our model.

```
trailmixmodel <- MIPModel() %>%
 add_variable(x[i],i=1:NMix,type="continuous",lb=0) %>%
 set_objective(sum_expr(Cost[i]*x[i],i=1:NMix ),"min")%>%
 add_constraint(sum_expr(MixChar[j,i]*x[i],i=1:NProd)
                  # Left hand side of constraint
                  >= CharMin[j],
                  # Inequality and Right side of constraint
                  j=1:NCharacteristic)
                  # Repeat for each resource, j.
results.trailmix <- solve_model(trailmixmodel,
                              with_ROI(solver = "glpk"))

results.trailmix
```

```
## Status: optimal
## Objective value: 4426.667
```

```
xvalue <- t(get_solution(results.trailmix, x[i])[,3])
```

We'll leave it to the reader to clean up the output of results.

Another classic example of a covering problem is a staff scheduling problem. In this case, a manager is trying to assign workers to cover the required demands throughout the day, week, or month.

3.9 Transportation Models

A transportation model is typically for getting material from one place to another at the lowest possible costs. It has sets of source points or nodes as well as ending or destination nodes. The decision variables are the amount to send on each route. Constraints are typically based on supply from the source nodes and capacity at the destination nodes. This naturally lends itself to potential network diagrams. In this case, $x_{i,j}$ is the amount of product to ship from node i to node j. The cost per unit to ship from node i to node j is $C_{i,j}$. The supply available from each supply node is S_i and the maximum demand that can be accommodate from each destination node is D_j.

In this formulation we need to make sure that we don't ship out more than capacity from each supply node. Similarly, we need to ensure that we don't take in more than demand capacity at any destination.

If we simply run this model, as is, the minimum cost plan would be to just do nothing! The cost would be zero. In reality, even though we

$$\begin{aligned}\text{Minimize } & \sum_i \sum_j C_{i,j} x_{i,j} \\ \text{subject to } & \sum_i x_{i,j} \leq D_j \; \forall \, j \\ & \sum_j x_{i,j} \leq S_i \; \forall \, i \\ & x_{i,j} \geq 0 \; \forall \, i,j\end{aligned}$$

Formula 3.7: Base Case for Transportation model are focused on costs in this application, there is an implied revenue and therefore profit (we hope!) that we aren't directly modeling. We are likely to instead be wanting to ship all of the product that we can at the lowest possible cost. More precisely, what we want to do is instead determine if the problem is supply limited or demand limited. This is a simple matter of comparing the net demand vs. the net supply and making sure that the lesser is satisfied completely.

If...	Then Situation is:	Source Constraints	Demand Constraints
$\sum_i S_i < \sum_j D_j$	Supply Constrained	$\sum_j x_{i,j} = S_i$	$\sum_i x_{i,j} \leq D_j$
$\sum_i S_i > \sum_j D_j$	Demand Constrained	$\sum_j x_{i,j} \leq S_i$	$\sum_i x_{i,j} = D_j$
$\sum_i S_i = \sum_j D_j$	Balanced	$\sum_j x_{i,j} = S_i$	$\sum_i x_{i,j} = D_j$

In the balanced situation, either source or demand constraints can be equalities.

Similarly, if we try to use equality constraints for both the supply and demand nodes but the supply and demand are not balanced, the LP will not be feasible.

Hint: In `ompr`, a double subscripted non-negative variable, $x_{i,j}$ can be defined easily as the following: `add_variable(x[i, j], type = "continuous", i = 1:10, j = 1:10, lb=0)`

Let's show how to set up a basic implementation of the tranportation problem in ompr. This demonstrates the use of double subscripted variables. Note that I have not defined the data. Also, you it should be customized for a particular application of being demand or supply constrained but it will give a good running head start.

```
transportationmodel <- MIPModel() %>%
 add_variable(x[i, j], type = "continuous",
              i = 1:NSupply,
              j = 1:NDest, lb=0) %>%
 set_objective (sum_expr(Cost[i,j] * x[i,j] ,
                         i=1:NSupply,
                         j=1:NDest ), "min") %>%
 add_constraint (sum_expr(x[i,j], i=1:NSupply)
                # Left hand side of Demand constraints
                >= D[j],
                # Inequality and Right side of constraint
                j=1:NDest)%>% #Repeat for each demand node j.
 add_constraint (sum_expr(x[i,j], j=1:NDest)
                # Left hand side of Supply constraints
                <= S[i],
                # Inequality and Right side of constraints
                i=1:NSupply)
                # Repeat for each supply node i.
results.transportation <-solve_model(transportationmodel,
                                      with_ROI(solver ="glpk"))
```

3.10 Transhipment Models

A generalization of the transportation model is that of transhipment where some nodes are intermediate nodes that are neither pure sources or destinations but can have both inflow and outflow.

3.11 Standard Form

Any linear program in inequality constraints can be converted into what is referred to as standard form. First, all strictly numerical terms are collected or moved to the right hand side and all variables are on the left hand side. It makes little difference as to whether the objective function is a *min* or a *max* function since a min objective function can be converted to a max objective function by multiplying everything a negative one. The converse is also true.

Lastly is a step where all of the inequalities are replaced by strict equality relations. The conversion of inequalities to equalities warrants a little further explanation. This is done by introducing a new, non-negative "slack" variable for each inequality.

3.12 Exercises

Exercise 3.1 (Transportation)**.**

Four manufacturing plants are supplying material for distributors in four regions. The four supply plants are located in Chicago, Beaverton, Eugene, and Dallas. The four distributors are in PDX (Portland), SEA (Seattle), MSP(Minneapolis), and ATL (Atlanta). Each manufacturing plant has a maximum amount that they can produce. For example, Chicago can produce at most 500. Similarly, the PDX region can handle at most 700 units. The cost to transport from Dallas to MSP is three times as high as the cost from Dallas to Atlanta. The table on the right displays the transportation cost between all the cities.

Formulate an explicit model for the above application that solves this transporation problem to find the lowest cost way of transporting as much as product as we can to distributors. Hint: You might choose to define variables based on the first letter of source and destination so XCP is the amount to ship from Chicago to PDX.

Node	PDX	SEA	MSP	ATL	*Supply*
Chicago	20	21	8	12	**500**
Beaverton	6	7	18	24	**500**
Eugene	8	10	22	28	**500**
Dallas	16	26	15	5	**600**
Capacity	*700*	*500*	*500*	*600*	

Table 3.12: Transportation cost between cities

Implement and solve the model using ompr. Be sure to discuss the solution as to why it makes it sense.

Exercise 3.2 (Explicit Transportation).

Formulate a generalized model for the above application that solves this transporation problem to find the lowest cost way of transporting as much as product as we can to distributors.

Hint: Feel free to use my LaTeX formulation for the general transportation model and make change(s) to reflect your case.

Implement and solve the model using ompr. Be sure to discuss the solution as to why it makes sense.

Exercise 3.3 (Convert LP to Standard Form).

Convert the three variable LP represented in Table 3.13 into standard form.

Max:

$$Profit = 20Chairs + 10Desks + 16Tables$$

S.t.

$$\begin{aligned}
&6Chairs + 3Desks + 4Tables \leq 2000\\
&8Chairs + 4Desks + 4Tables \leq 2000\\
&6Chairs + 3Desks + 8Tables \leq 1440\\
&40Chairs + 20Desks + 16Tables \leq 9600\\
&Tables \leq 200\\
&Chairs, Desks, Tables \geq 0
\end{aligned}$$

Table 3.13: Base LP Model

Exercise 3.4 (Convert LP to Standard Form).

Implement and solve the standard form of the LP using R. Be sure to interpret the solution and discuss how it compares to the solution from the original model.

Exercise 3.5 (Convert Generalized LP to Standard Form).

Convert the generalized production planning LP, displayed on the right in formula 3.7, into standard form.

Hint: define a set of variables, s_j to reflect these changes and add it to the following formulation.

$$\begin{aligned}
\text{Maximize } & \sum_{i=1}^{3} P_i x_i\\
\text{subject to } & \sum_{i=1}^{3} R_{i,j} x_i \leq A_j \; \forall \; j\\
& x_i \geq 0 \; \forall \; i
\end{aligned}$$

Formula 3.7: Generalized LP case

4
Linear Programming Sensitivity Analysis

We can get a lot more than just the objective function value and the decision variable from a solved linear program. In particular, we can potentially explore the impact of changes in constrained resources, changes in the objective function, forced changes in decision variables, and the introduction of additional decision variables.

4.1 Base Case

To demonstrate this, let's revisit our explicit model of production planning. We will use the explicit version for the sake of clarity and simplicity but the same techniques could be used for the generalized model or other linear programs.

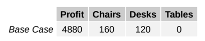
Recall the explicit production planning problem from earlier on Formula 4.1.

$$
\begin{split}
\text{Max: } & 20Chairs + 14Desks + 16Tables \\
\text{s.t.} & \\
& 6Chairs + 2Desks + 4Tables \leq 2000 \\
& 8Chairs + 6Desks + 4Tables \leq 2000 \\
& 6Chairs + 4Desks + 8Tables \leq 1440 \\
& 40Chairs + 25Desks + 16Tables \leq 9600 \\
& Tables \leq 200 \\
& Chairs,\ Desks,\ Tables \geq 0
\end{split}
$$

Formula 4.1: Explicit model of production planning

The implementation that we did earlier for production planning was straightforward.

```
Base3VarModel <- MIPModel() %>%
  add_variable(Chairs, type ="continuous", lb = 0) %>%
  add_variable(Desks, type ="continuous",lb = 0) %>%
  add_variable(Tables, type ="continuous", lb = 0) %>%

  set_objective(20*Chairs+14*Desks+16*Tables,"max")%>%

  add_constraint(6*Chairs+2*Desks+4*Tables<= 2000)%>%
  #fabrication
  add_constraint(8*Chairs+6*Desks+4*Tables<= 2000)%>%
  #assembly
  add_constraint(6*Chairs+4*Desks+8*Tables<= 1440)%>%
  #machining
  add_constraint(40*Chairs+25*Desks+16*Tables<= 9600)%>%
  #wood
  add_constraint(Tables <= 200) %>% #
  solve_model(with_ROI(solver = "glpk"))
  obj_val <- objective_value(Base3VarModel)
  xchairs <- get_solution (Base3VarModel, Chairs)
  xdesks  <- get_solution (Base3VarModel, Desks)
  xtables <- get_solution (Base3VarModel, Tables)
base_case_res <- cbind(objective_value(Base3VarModel),
                    get_solution(Base3VarModel, Chairs),
                    get_solution(Base3VarModel, Desks),
                    get_solution(Base3VarModel, Tables))
colnames(base_case_res)<-list("Profit",
                              "Chairs",
                              "Desks",
                              "Tables")
rownames(base_case_res)<-list("Base Case")
grid.table(base_case_res)
```



	Profit	Chairs	Desks	Tables
Base Case	4880	160	120	0

Table 4.1: Base Case Production Plan

In the base case, we are producing chairs and desks but not tables to generate a total profit of $4880.

4.2 Shadow Prices

There are many resources, some are fully used while some are not fully utilized. How could we prioritize the importance of each resource? For example, if the factory manager could add a worker to one department, which should it be? Conversely, if an outside task came up where should she draw the time from? We could modify the model and rerun. For complex situations, this may be necessary. On the other hand, we could also use sensitivity analysis to explore the relative value of the resources.

Let's begin by examining the row duals, also known as shadow prices.

```
#rduals1 <-as.matrix(get_row_duals(result1), ncol=5)
rduals1 <-as.matrix(get_row_duals(Base3VarModel))
dimnames(rduals1)<-list(c("fabrication",
                          "assembly",
                          "machining",
                          "wood",
                          "demand"),
                          c("Row Duals"))
grid.table(format(rduals1,digits=4))
```

	Row Duals
fabrication	0
assembly	1
machining	2
wood	0
demand	0

Table 4.2: Shadow Prices of Constrained Resources

The row duals or shadow prices for fabrication is zero. This means that the marginal value of one additional hour of fabrication labor time is $0. This makes sense if you examine the amount of fabrication time used and realize that the company is not using all of the 2000 fabrication department's time available. Therefore adding for fabrication hours certainly can't help improve the production plan.

On the other hand, all of the assembly time (resource) is used in the optimal solution. The shadow price of an hour of assembly time is $1.0. This means that for every hour of additional assembly time within certain limits, the objective function will increase by $1.0 also.

All of the 1440 hours of labor available in the machining center are consumed by the optimal production plan. Increasing the hours available may allow the company to change the production plan and increase the profit. While you could rerun the model with increased machining hours to determine the new optimal production plan but if only want to know the change in the optimal objective function value, you can determine that from the shadow price of the machining constraint. Each additional hour (within a certain range) will increase the profit by $2.0.

This potential for increased profit can't be achieved by simply increasing the resource - it requires a modified production to utilize this increased resource. To find the production plan that creates this increased profit, let's solve a modified linear program.

4.3 Testing Adding Labor to Machining

Let's test the numerical results from the Shadow Price table by adding an hour of labor to the machining center. The model is represented in the Formula 4.2.

$$
\begin{aligned}
\text{Max: } & 20 Chairs + 14 Desks + 16 Tables \\
\text{s.t.} & \\
& 6 Chairs + 2 Desks + 4 Tables \leq 2000 \\
& 8 Chairs + 6 Desks + 4 Tables \leq 2000 \\
& 6 Chairs + 4 Desks + 8 Tables \leq 1441 \\
& 40 Chairs + 25 Desks + 16 Tables \leq 9600 \\
& Tables \leq 200 \\
& Chairs, \; Desks, \; Tables \geq 0
\end{aligned}
$$

Formula 4.2: Explicit model of production planning adding labor to machining

And the code is showed in the following chunk, with Table 4.3 showing the optimal model result.

```r
IncMachiningHrs <- MIPModel() %>%
  add_variable(Chairs, type = "continuous", lb = 0) %>%
  add_variable(Desks, type = "continuous",lb = 0) %>%
  add_variable(Tables, type = "continuous", lb = 0) %>%

  set_objective(20*Chairs + 14*Desks + 16*Tables,"max")%>%

  add_constraint(6*Chairs + 2*Desks + 4*Tables<= 2000)%>%
  #fabrication
  add_constraint(8*Chairs + 6*Desks + 4*Tables<= 2000)%>%
  #assembly
  add_constraint(6*Chairs + 4*Desks + 8*Tables<= 1441)%>%
  #machining
  add_constraint(40*Chairs + 25*Desks + 16*Tables<= 9600)%>%
  #wood
  add_constraint(Tables <= 200) %>% #
  solve_model(with_ROI(solver = "glpk"))
inc_mc_res  <- cbind(objective_value(IncMachiningHrs),
                     get_solution(IncMachiningHrs, Chairs),
                     get_solution(IncMachiningHrs, Desks),
                     get_solution(IncMachiningHrs, Tables))
colnames(inc_mc_res)<-list("Profit",
                           "Chairs",
                           "Desks",
                           "Tables")
rownames(inc_mc_res)<-list("Increased M/C Hrs")
temp1 <- rbind(base_case_res, inc_mc_res)
grid.table(temp1)
```

	Profit	Chairs	Desks	Tables
Base Case	4880	160	120	0
Increased M/C Hrs	4882	161.5	118	0

Table 4.3: Production Plan with One Additional Machining Hour

These results confirmed that adding one hour of machining time results in a *new* production plan that generates a profit of two more dollars, exactly as expected.

The Shadow Prices can be very helpful in larger problems where questions might come up of trying to evaluate or prioritize limited resources.

4.4 Shadow Prices of Underutilzed Resources

The shadow price on wood is zero (as was also the case for assembly hours). This means that even a large increase in wood would not affect the maximum profit or the optimal production plan. Essentially there is plenty of wood, having more would not allow enable a better profit plan to be possible. Let's confirm this as well with a numerical example by adding *10,000* more square feet of wood. See formula 4.3.

$$
\begin{aligned}
\text{Max: } & 20 * Chairs + 14 * Desks + 16 * Tables \\
\text{s.t.} & \\
& 6 * Chairs + 2 * Desks + 4 * Tables \leq 2000 \\
& 8 * Chairs + 6 * Desks + 4 * Tables \leq 2000 \\
& 6 * Chairs + 4 * Desks + 8 * Tables \leq 1440 \\
& 40 * Chairs + 25 * Desks + 16 * Tables \leq 19600 \\
& Tables \leq 200 \\
& Chairs, \ Desks, \ Tables \geq 0
\end{aligned}
$$

Formula 4.3: Explicit model for Underutilzed Resources

In the same way that it was done in previous point, the next chunk shows how to build in R an underutilized scenario. Table 4.3 contains the result for this model.

```
IncWood<- MIPModel() %>%
  add_variable(Chairs, type = "continuous", lb = 0) %>%
  add_variable(Desks, type = "continuous",lb = 0) %>%
  add_variable(Tables, type = "continuous", lb = 0) %>%

  set_objective(20*Chairs + 14*Desks + 16*Tables,"max")%>%

  add_constraint(6*Chairs + 2*Desks + 4*Tables<= 2000)%>%
  #fabrication
  add_constraint(8*Chairs + 6*Desks + 4*Tables<= 2000)%>%
  #assembly
  add_constraint(6*Chairs + 4*Desks + 8*Tables<= 1440)%>%
  #machining
  add_constraint(40*Chairs + 25*Desks + 16*Tables<= 19600)%>%
  #wood
  add_constraint(Tables <= 200) %>% #
  solve_model(with_ROI(solver = "glpk"))
inc_wd_res  <- cbind(objective_value(IncWood),
                     get_solution (IncWood, Chairs),
                     get_solution (IncWood, Desks),
                     get_solution (IncWood, Tables))
colnames(inc_wd_res)<-list("Profit",
                           "Chairs",
                           "Desks",
                           "Tables")
rownames(inc_wd_res)<-list("Increased Wood")
temp2 <- rbind(base_case_res, inc_mc_res, inc_wd_res)
       grid.table(temp2)
```

	Profit	Chairs	Desks	Tables
Base Case	4880	160	120	0
Increased M/C Hrs	4882	161.5	118	0
Increased Wood	4880	160	120	0

Table 4.4: Production Plan with 10,000 More Wood

The demand constraint of 200 is an upper limit only on Tables. Since far fewer than 200 Tables are produced in the optimal production plan, relaxing this constraint also has no impact (shadow price=0).

4.5 Reduced Costs of Variables

Next, we move on to the *reduced costs* of variables. This concept often requires rereading several times. The mathematical details rely on the structure of linear programming and the simplex algorithm. We won't go into great detail on the origin of this mathematically in detail here.

Reduced Cost of Chairs

Let's start by examining the Chairs. The reduced cost for Chairs is the per unit marginal profit minus the per unit value (in terms of shadow prices) of the resources used by a unit in production.

Next, let's discuss the column duals which are often referred to as reduced costs of variables. Let's extract these from the results just as we did for the shadow prices.

```
#cduals1 <-as.matrix(get_column_duals(result1), ncol=1 )
cduals1 <-as.matrix(get_column_duals(Base3VarModel) )
dimnames(cduals1)<-list(c("Chairs", "Desks", "Tables"),
                        c("Column Duals"))
grid.table(format(cduals1,digits=4))
```

	Column Duals
Chairs	0
Desks	0
Tables	−4

Table 4.5: Reduced Costs of Variables

These results are interesting. The reduced costs of variables that are between simple upper and lower bounds will be zero. The reduced cost of a variable can be thought as the difference between the value of the resources consumed by the resources and the value of the variable in the objective function. All of our variables have simple lower bounds of zero and no upper bounds. The first two (Chairs and Desks) have a zero reduced cost and the last one (Tables) is negative.

Let's start by examining the Shadow Prices of the resources along with the number of each resource used in the production of a single chair.

```
chair_res_used<-cbind(rduals1,c(2,8,6,40,0))
colnames(chair_res_used)<-c("Row Duals", "Resources Used")
grid.table(format(chair_res_used, digits=4))
```

	Row Duals	Resources Used
fabrication	0	2
assembly	1	8
machining	2	6
wood	0	40
demand	0	0

Table 4.6: Resources Used by a Chair and their Shadow Prices

Taking the product in each row and adding them up will give the marginal value of the resources consumed by an incremental change in production of chairs. In this case, the marginal value is $0 + 1 * 8 + 2 * 6 + 0 + 0 = 20$. Since the profit per chair (from the objective function coefficient on Chairs) is also \$20, they are in balance and the difference between them is zero which is why the reduced cost for Chairs is zero. This can be thought of saying that there is marginal benefit of cost to a very small forced change in the value of the Chairs variable. The value of the resources used in producing a chair equals that of the profit of a chair.

We can repeat the same process and calculations for Desks.

```
desk_res_used<-cbind(rduals1,c(2,6,4,25,0))
colnames(desk_res_used)<-c("Row Duals", "Resources Used")
grid.table(format(desk_res_used, digits=4))
```

	Row Duals	Resources Used
fabrication	0	2
assembly	1	6
machining	2	4
wood	0	25
demand	0	0

Table 4.7: Resources Used by a Desk and their Shadow Prices

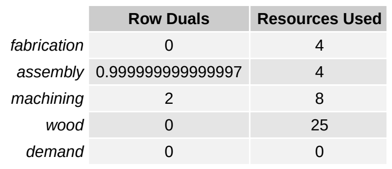
In this case, using the columns from Table 4.6 we can calulate the marginal value as $0 + 1 * 6 + 2 * 4 + 0 + 0 = 14$. Since the profit per desk (from the objective function coefficient on Desks) is also \$14, they are in balance and the difference between them is zero which is why the reduced cost for the *Desks* variable is also zero.

Reduced Price of Tables

The situation is more interesting for Tables. The production plan does not call for producing any tables. This means that it is sitting right at the lower bound of zero Tables. This hints that perhaps we would like to make even less than zero Tables if we could and that being *forced* to make even one table would be costly. A reduced cost of -4 means that the opportunity cost of resources used in producing a table is \$4 more than the profit of producing a single table. In other words, we would expect that if we force the production of a single table, the over all production plan's profit will go down by \$4.

Let's go ahead and test this interpretation by again looking at the value of the resources consumed in the production of a table and the number used.

```r
table_res_used<-cbind(rduals1,c(4,4,8,25,0))
colnames(table_res_used)<-c("Row Duals", "Resources Used")
grid.table(table_res_used)
```



	Row Duals	Resources Used
fabrication	0	4
assembly	0.999999999999997	4
machining	2	8
wood	0	25
demand	0	0

Table 4.8: Resources Used by a Table and their Shadow Prices

Notice that the values based on shadow prices of the resources used by a table are $\$1 * 4 + \$2 * 8 = \$20$. Alas, the profit for each table is just \$16 which means that forcing the production of a single table will decrease the production plan's profit by \$4. In other words, the impact on the objective function is $\$ - 4$ which is the same as the reduced price entry of Tables.

Okay, now let's put both side by side in a table to show the results.

Okay, now let's test it. We will modify the formulation to set a lower bound on the number of Tables to be 1. Note that we do this in case by setting the *lb* option in the *add_variable* to be 1. Also, the demand constraint for tables could also be accommodated by setting the table variable's upper bound (ub) to 200.

```r
Table1Model <- MIPModel() %>%
  add_variable(Chairs, type = "continuous", lb = 0) %>%
  add_variable(Desks, type = "continuous",lb = 0) %>%
  add_variable(Tables, type = "continuous", lb = 1) %>%

  set_objective(20*Chairs + 14*Desks + 16*Tables,"max")%>%
  add_constraint(6*Chairs + 2*Desks + 4*Tables<= 2000)%>%
  #fabrication
  add_constraint(8*Chairs + 6*Desks + 4*Tables<= 2000)%>%
  #assembly
  add_constraint(6*Chairs + 4*Desks + 8*Tables<= 1440)%>%
  #machining
  add_constraint(40*Chairs + 25*Desks + 16*Tables<= 9600)%>%
  #wood
  add_constraint(Tables <= 200) %>% #
  solve_model(with_ROI(solver = "glpk"))
  obj_val <- objective_value(Table1Model)
  xchairs <- get_solution (Table1Model, Chairs)
  xdesks  <- get_solution (Table1Model, Desks)
  xtables <- get_solution (Table1Model, Tables)
  Table1_case_res           <- cbind(xchairs,
                                     xdesks,
                                     xtables,
                                     obj_val)
  rownames(Table1_case_res) <- ""
  #rownames(Table1_case_res) <- "Amount to Produce"
  grid.table(Table1_case_res)
```

xchairs	xdesks	xtables	obj_val
152	130	1	4876

Table 4.9: Production Plan with Table Set to One

Let’s compare the results for the new production plan and the original base case looking at the Table 4.9.

```
rownames(base_case_res) <- "Base Case"
Table1_case_res <- cbind(obj_val, xchairs,xdesks,xtables)
rownames(Table1_case_res) <- "After change"
temp3 <-rbind(base_case_res,Table1_case_res)
grid.table(temp3)
```

	Profit	Chairs	Desks	Tables
Base Case	4880	160	120	0
After change	4876	152	130	1

Table 4.10: Impact of a Forced Change in Tables

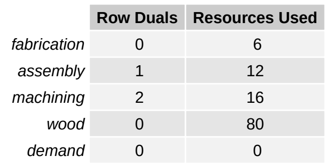
As we expected, the forced change of making one additional table resulted in a decrease of the overall profit from $4800 to $4876. This occurred because making one table meant that we had fewer of the precious, limited resources.

This meant that the number of chairs and desks were changed resulting lower profit even though a table is on its own profitable.

- Evaluating a New Product Design with Shadow Prices *

Let’s consider a design proposal for a bookcase. The bookcase has a projected profit of $35 each and uses 6 hours of fabrication time, 12 of assembly, and 16 of machining. It uses 80 square feet of wood.

```
bookcase_res_used<-cbind(rduals1,c(6,12,16,80,0))
colnames(bookcase_res_used)<-c("Row Duals",
                               "Resources Used")
grid.table(format(bookcase_res_used,digits=4))
```



	Row Duals	Resources Used
fabrication	0	6
assembly	1	12
machining	2	16
wood	0	80
demand	0	0

Table 4.10: Resources Used by a Bookcase and their Shadow Prices

Even without adding it to the model, we can check to see if it is worthwhile to consider seriously. The opportunity cost of producing one bookcase would result in the $$35 - $1*12* - *$2*16=$-9 $. In other words, even though a bookcase has higher profit than any other product, producing one would cost the company nine dollars of overall profit.

We could interpret this as a simple hard stop on the decision to produce bookcases but we could go one step further by setting a target for redesigning the product or its production. If the machining time could be reduced by 4.5 hours to 11.5 hours, then the value of the resources consumed would be equal to the profit and we would be indifferent to producing some bookcases.

4.6 Exercises

Exercise 4.1 (Adding Frames).

Your company has extended production to allow for producing picture frames and is now including a finishing department that primes and paints (or stains) the furniture.

Characteristic	Chairs	Desks	Frames	Tables	Available
Profit	$20	$14	$3	$16	
Fabrication	6	2	1	4	1440
Assembly	8	6	1	8	1440
Machining	6	4	1	25	2000
Painting	7	10	2	12	1000
Wood	40	25	5	16	9600

a) Use R Markdown to create your own description of the model.
b) Extend the R Markdown to show your LP Model. Be sure to define models.
c) Solve the model in R.
d) Interpret and discuss the model in R Markdown.
e) Examine and reflect upon the reduced costs and shadow prices from the context of which products to produce and not produce.
f) Using the results from e), (i.e. reduced cost and shadow prices)

make one change to the base model's **objective function** that will change the production plan. Rerun and discuss the new results.

g) Using the results from e), (i.e. reduced cost and shadow prices) make one change to the base model's **resource usage values** that will change the production plan. Rerun and discuss the new results.

h) Using the results from e), (i.e. reduced cost and shadow prices) make one change to the base model's **available resource values** that will change the production plan. Rerun and discuss the new results.

i) Combine the results of the base case e), as well as the variations f) through h) into a single table and discuss the results.

Exercise 4.2 (Revisiting Transportation)**.**

Using sensitivity analysis, revisit the transportation exercise from chapter 3.

a) If one more unit of supply was available, where would it be prioritized and why?

b) If demand could be increased by one unit, would it affect the result and at which destination node it be preferred and why?

5
Data Envelopment Analysis

5.1 Introduction

Data envelopment analysis or DEA is a powerful tool for conducting studies of efficiency and has been used in thousands of publications since its inception in the 1970s.[1]

[1] This chapter is drawn from an introduction chapter in the book, DEA Using R by the same author. More details on DEA are available from that book.

While tools exist for conducting the evaluations, it is important to understand how the tools work. Many DEA studies have been conducted and published by authors with only a superficial understanding of the technique. This is equivalent to having a house built by carpenters that only (barely?) understand how to use a hammer. The purpose of this document is to show how DEA works, what it means, and how to use R for getting started with DEA. In order to keep things simple, this first step only looks at the input-oriented envelopment model with constant returns to scale. We will consider other models soon.

This chapter walks through how DEA works and then shows how to implement the model in R using two very different approaches. Over the years, I have built DEA models in many languages and platforms: Pascal, LINDO, LINGO, Excel Macros, Excel VBA, GAMS, AMPL, XPress-MOSEL, and GLPK among others.

Recently, my research group at PSU adopted the R platform based on the following strengths:

- Open-source so that people can dive as deep as they need and no risk of future vendor lock-in
- Freely available so it does not strain research budgets
- Extensive collection of numerical add-in packages (over 10000 at CRAN) to build upon
- Robust distribution of add-in packages could help improve distribution of our own contributions

- Multiple linear programming packages to choose from among
- Widespread usage and acceptance in the research and analytics communities
- No arbitrary capacity limits
- One platform that combines optimization and statistical analysis

5.2 Creating the Data

Let's start by defining data. DEA typically has inputs and outputs. The input(s) are typically resources that are consumed in the production of output(s). Inputs are referred to as "bads" in that higher levels at the same level of output is considered worse. Similarly, holding everything else constant, an increase in any single output is laudable. Examples of inputs might include capital, labor, or number of machines.

In contrast, outputs are the "good" items that are being produced by the actions of the producers. Examples of outputs might include automobiles produced, customers served, or graduating students.

Let's start by creating a simple dataset. We'll assume that we have a small group of units, named A, B, C, and D. These use inputs which we will label x and produce products or outputs that we will label y.

Which of these units are the best? Which are laggards? Simply looking at the data is difficult. Let's get started now with R.

```r
x <- matrix(c(10,20,30,50),ncol=1,
            dimnames=list(LETTERS[1:4],"x"))
y <- matrix(c(75,100,300,400),ncol=1,
            dimnames=list(LETTERS[1:4],"y"))
```

```r
pander(cbind(x,y), caption="First Dataset for DEA")
```

Table 5.1: First Dataset for DEA

	x	y
A	10	75
B	20	100
C	30	300
D	50	400

While we aren't giving a formal introduction to R, let's start by explaining the above commands. The first line `library (pander)` loads a common library that makes for nicely formatted tables.[2]

[2] Pander is one of many thousands of contributed external packages that can be downloaded to extend R. If it does not work to load it using the library, you likely need to install it onto your computer using the `install` command.

The following commands create matrices that hold the data and have named rows and columns to match. The <- symbol is a key function in R and means to assign what is in the right to the object on the left.

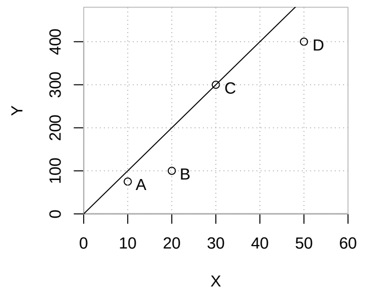
For benchmarking, we want to know which ones are doing the best job.

Can you tell which units represent the best tradeoff between inputs and outputs? None of the units are dominated by any of the other units. Dominance would be producing more outputs using less input so let's move on to looking at it graphically.

5.3 Graphical Analysis

Let's start by doing a simple plot of the data. For now, I'm going to make use of a function in Peter Bogetoft and Lars Otto's Benchmarking package which provides a very handy two-dimensional plot in the format often used for showing production.

```
library(Benchmarking, quietly=TRUE)
dea.plot(x, y, RTS="crs", ORIENTATION="in-out",
         txt=LETTERS[1:length(x)],
         add=FALSE, wx=NULL, wy=NULL, TRANSPOSE=FALSE,
         fex=1, GRID=TRUE, RANGE=FALSE, param=NULL)
```



This chart clearly shows that unit C has the best ratio of output (y) to input (x). The diagonal line represents an efficiency frontier of best practices that could be achieved by scaling up or down unit C. As the input is scaled up or down, it is assumed that the output of unit C would be scaled up or down by the same value. We will revisit this assumption in a later section but for now, think of this as saying that unit C cannot enjoy economies of scale by getting larger or suffer from diseconomies of scale by getting smaller so it is referred to as constant returns to scale or CRS.

Furthermore, we can graphically examine the technical efficiency of each of the other units. I'm going to start with unit B since it is a little easier to visualize. For now, let's think of the question, how much more or less input would C require to produce as much output as B.

To determine this efficiency score, simply draw a horizontal line from B to the efficiency frontier on the left. This point can be thought of a target for B to be efficient. This point has the same output as B but uses only half as much input. The efficiency score can be calculated as the ratio of the distance from the vertical axis to the target divided by the distance from the vertical axis to B. This distance is simply 10/20 or 50%.

Another question is how to construct the target for B's evaluation. It is simply made by scaling down C to a third of its original size. This results in a target that is composed of about 0.333 of C. Also, it should be noted that it makes use of no part of A, B, or D.

The same steps can be followed for analyzing units A, C, and D resulting in efficiencies of 75%, 100%, and 80% respectively.

5.4 The Linear Programs for DEA

The graphical approach is intuitive but accuracy is limited to that of drawing tools. More importantly, it does not scale to more complex models with multiple inputs and outputs. Let's frame this topic mathematically so that we can proceed systematically.

A key way to begin the mathematical development of the envelopment model is to ask, can you find a combination of units that produces a target with at least as much output using less input? The blend of other units is described by a vector λ. Another way to denote this is λ_j is the specific amount of a unit j used in setting the target for for performance for unit k.

This can be easily expanded to the multiple input and multiple output case by defining $x_{i,j}$ to be the amount of the i'th input used by unit j and $y_{r,j}$ to be the amount of the r'th output produced by unit j. For simplicity, this example will focus on the one input and one output case rather than the m input and s output case but the R code explicitly allows for $m, s > 1$. To make the code more readable, I will use a slightly different convention N^X or NX instead of m to refer to the number of inputs (x's) and N^Y or NY to be the number of outputs (y's) instead of s. Also, the normal mathematical convention is to use n to denote the number of Decision Making Units (DMUs) so I will use N^D or ND to indicate that in the R code.

Let's connect these ideas together now using these mathematical building blocks. The core idea of the envelopment model of a DMU k can be thought of as to find a target constructed of a mix of the

DMU's described by a vector λ that uses no more input to achieve the same or more every output as DMU *k*. The amount of the *i'th* input used by the target is then $\sum_{j=1}^{N^D} x_{i,j}\lambda_j$. By the same token, the amount of the *r'th* output produced by the target is $\sum_{j=1}^{N^D} y_{r,j}\lambda_j$.

This gives us two sets of constraints along with a restriction of non-negativity. These are shown in the following relationships that must all be satisfied simultaneously.

$$\begin{aligned}
&\sum_{j=1}^{N^D} x_{i,j}\lambda_j \leq x_{i,k} \ \forall \ i\\
&\sum_{j=1}^{N^D} y_{r,j}\lambda_j \geq y_{r,k} \ \forall \ r\\
&\lambda_j \geq 0 \ \forall \ j
\end{aligned}$$

This is not yet a linear program because it is missing an objective function. It defines what is an acceptable target of performance that is at least as good as DMU *k* but does not try to find a *best* target.

The two most common approaches to finding the best target are the input-oriented and output-oriented models. In the output-oriented model, the first (input) constraint is satisfied while trying to *exceed* the second constraint (output) by as much possible. This focus on increasing the output is then called an *output orientation*.

In this chapter, we will focus on satisfying the second constraint while trying to improve upon the first by as much as possible. In other words, we will satisfy the second (output) constraint but try to form a target that uses as little input as possible.

Let's define the proportion of the studied unit's input needed by the target as θ. A value of $\theta = 1$ then means no input reduction can be found in order to produce that unit's level of output.

In fact, we will go one step further and say that we want to find the maximum input possible reduction in k's input or conversely, the minimum amount of the input that could be used by the target while still producing the same or more output. We do this by adding a new variable, θ, which is the radial reduction in the amount of DMU k's input. We want to find how low we can drive this by *minimizing* θ. This gives us the following linear program, often abbreviated as an LP.[3]

[3] The mathematics in this book is written using LaTeX embedded within the rmarkdown document. Another benefit of using rmarkdown is that it is possible embed LaTeX without requiring using a full LaTeX document.

$$\begin{aligned}
\text{minimize } & \theta \\
\text{subject to } & \sum_{j=1}^{N^D} x_{i,j}\lambda_j \leq \theta x_{i,k} \; \forall \; i \\
& \sum_{j=1}^{N^D} y_{r,j}\lambda_j \geq y_{r,k} \; \forall \; r \\
& \lambda_j \geq 0 \; \forall \; j
\end{aligned}$$

Expressing the target on the left and the actual unit's value and radial reduction on the right is conceptually straightforward to understand. Unfortunately, optimization software typically requires collecting all the variables on the left and putting constants on the right hand side of the inequalities. This is easily done.

$$\begin{aligned}
\text{minimize } & \theta \\
\text{subject to } & \sum_{j=1}^{N^D} x_{i,j}\lambda_j - \theta x_{i,k} \leq 0 \; \forall \; i \\
& \sum_{j=1}^{N^D} y_{r,j}\lambda_j \geq y_{r,k} \; \forall \; r \\
& \lambda_j \geq 0 \; \forall \; j
\end{aligned}$$

5.5 Creating the LP - The Algebraic Approach

There are two fundamentally different approaches to setting up linear programs for solving. The first approach is to define data structures to pass vectors for the objective function coefficients and constraint right hand sides along with a matrix of data describing the constraints. This requires careful setting up of the linear programs and is a big cognitive step away from the mathematical representation. Another approach is to use algebraic modeling languages. Standalone algebraic optimization modeling languages include LINGO, AMPL, GAMS, GMPL, and others.

Until recently, R did not have the ability to do algebraic modeling optimization but recently a few efforts have provided support for this. A new package, *ompr*, provides an algebraic perspective that matches closely to the summation representation of a linear program shown earlier. Don't worry, if you want to see the data structure format approach, that is covered in the *DEA Using R* book.

Let's define some data structures for holding our data and results.

```
ND <- nrow(x); NX <- ncol(x); NY <- ncol(y);
               # Define data size
xdata<-x[1:ND,]
dim(xdata)<-c(ND,NX)
ydata<-y[1:ND,]
dim(ydata)<-c(ND,NY)
               # Now we will create lists of names
DMUnames <- list(c(LETTERS[1:ND]))
   # DMU names: A, B, ...
Xnames<- lapply(list(rep("X",NX)),paste0,1:NX)
   # Input names: x1, ...
Ynames<- lapply(list(rep("Y",NY)),paste0,1:NY)
   # Output names: y1, ...
Vnames<- lapply(list(rep("v",NX)),paste0,1:NX)
   # Input weight names: v1, ...
Unames<- lapply(list(rep("u",NY)),paste0,1:NY)
   # Output weight names: u1, ...
SXnames<- lapply(list(rep("sx",NX)),paste0,1:NX)
   # Input slack names: sx1, ...
SYnames<- lapply(list(rep("sy",NY)),paste0,1:NY)
   # Output slack names: sy1, ...
Lambdanames<- lapply(list(rep("L_",ND)),
                     paste0,LETTERS[1:ND])
results.efficiency <- matrix(rep(-1.0, ND),
                             nrow=ND, ncol=1)
dimnames(results.efficiency)<-c(DMUnames,"CCR-IO")
   # Attach names

results.lambda <- matrix(rep(-1.0, ND^2),
                         nrow=ND,ncol=ND)
dimnames(results.lambda)<-c(DMUnames,Lambdanames)
results.xslack     <- matrix(rep(-1.0, ND*NX),
                             nrow=ND,ncol=NX)
dimnames(results.xslack)<-c(DMUnames,SXnames)
results.yslack     <- matrix(rep(-1.0, ND*NY),
                             nrow=ND,ncol=NY)
dimnames(results.yslack)<-c(DMUnames,SYnames)
```

We're going to use our data from earlier but first we will load a collection of libraries to be used later. The ompr package is for optimization and serves as a general human readable format of optimization models that can then interface with a variety of solver engines. The ROI.plugin.glpk package is for the specific solver engine, glpk, that we used. Other LP solving engines are available and can be used instead.

```
library(dplyr, quietly=TRUE)
   # For data structure manipulation
library(ROI, quietly=TRUE)
   # R Optimization Interface package
library(ROI.plugin.glpk, quietly=TRUE)
   # Connection to glpk as solver
library(ompr, quietly=TRUE)
   # Optimization Modeling using R
library(ompr.roi, quietly=TRUE)
   # Connective tissue
```

Now that we have loaded all of the packages that we use as building blocks, we can start constructing the model.

We are going to start by building a model for just one DMU, in this case, the second DMU (B).

```
k<-2    # DMU to analyze.
        # Let's start with just one DMU, B, for now.
result <- MIPModel() %>%
  add_variable(vlambda[j], j = 1:ND, type = "continuous",
               lb = 0) %>%
  add_variable(vtheta, type = "continuous") %>%
  set_objective(vtheta, "min") %>%
  add_constraint(sum_expr(vlambda[j] * xdata[j,1], j = 1:ND)
                 <= vtheta * xdata[k,1]) %>%
  add_constraint(sum_expr(vlambda[j] * ydata[j,1], j = 1:ND)
                 >= ydata[k,1]) %>%
  solve_model(with_ROI(solver = "glpk"))
omprtheta <-  get_solution(result, vtheta)
omprlambda <-  get_solution(result, vlambda[j])
ND <- 4 # Four Decision Making Units or DMUs
NX <- 1 # One input
NY <- 1 # One output
   # Only doing analysis for one unit at a time to start
results.efficiency <- matrix(rep(-1.0, 1), nrow=1, ncol=1)
results.lambda     <- matrix(rep(-1.0, ND), nrow=1,ncol=ND)
results.efficiency <- t(omprtheta)
colnames(results.efficiency) <- c("CCR-IO")
results.lambda <- t(omprlambda[3])
   # Takes third column from results and transposes results
   #  to be structured correctly for later viewing
colnames(results.lambda) <-
```

```
  c("$\\lambda_A$", "$\\lambda_B$",
    "$\\lambda_C$", "$\\lambda_D$")
pander(cbind(results.efficiency, results.lambda),
 caption=
   "Input-Oriented Envelopment Analysis for DMU B (CCR-IO)")
```

Table 5.2: Input-Oriented Envelopment Analysis for DMU B (CCR-IO)

	CCR-IO	λ_A	λ_B	λ_C	λ_D
value	0.5	0	0	0.3333	0

The above table follows a few convenient conventions. First, rather than using θ, we label the efficiency score by the model used for the model, in this case the constant returns to scale is labeled as CCR after Charnes, Cooper, and Rhodes. Later we will cover other models including the variable returns to scale model, labeled BCC after Banker, Charnes, and Cooper.

Another convention is to embed the *orientation* in the table. The term IO refers to an input-oriented model where the primary focus is on achieving efficiency through input reduction. Output-orientation is also very common where the primary goal is for unit to increase output with the requirement of not using any more input.

Another convention is to use "L" in place of the Greek symbol λ due to complications with using Greek symbols in R matrix row or column names.

The results in the table indicate that DMU B has an efficiency score of 50%. The target of performance is made of DMU C scaled down by a factor of 0.33. These results match the graphical results from earlier.

Let's now extend it to handle multiple inputs, NX, and outputs, NY. Of course this doesn't have any impact on our results just yet since we are still only using a single input and output but we now have the structure to accommodate the more general case. To provide a little variety, we'll change it to the first DMU, A, to give a little more variety.

```
k<-1    # Analyze first unit, DMU A.
result <- MIPModel() %>%
  add_variable(vlambda[j], j = 1:ND, type = "continuous",
               lb = 0) %>%
  add_variable(vtheta, type = "continuous") %>%
  set_objective(vtheta, "min") %>%
  add_constraint(sum_expr(vlambda[j] * xdata[j,i], j = 1:ND)
                 <= vtheta * xdata[k,i], i = 1:NX) %>%
  add_constraint(sum_expr(vlambda[j] * ydata[j,r], j = 1:ND)
                 >= ydata[k,r], r = 1:NY) %>%
  solve_model(with_ROI(solver = "glpk"))
omprtheta <-  get_solution(result, vtheta)
omprlambda <-  get_solution(result, vlambda[j])
results.efficiency <- t(omprtheta)
colnames(results.efficiency) <- c("CCR-IO")
results.lambda <- t(omprlambda[3])

colnames(results.lambda) <- c("$\\lambda_A$", "$\\lambda_B$",
                              "$\\lambda_C$", "$\\lambda_D$")
pander(cbind(results.efficiency, results.lambda),
       caption=
         "Input-Oriented Envelopment Analysis for DMU A")
```

Table 5.3: Input-Oriented Envelopment Analysis for DMU A

	CCR-IO	λ_A	λ_B	λ_C	λ_D
value	0.75	0	0	0.25	0

Again, the results match what would be expected from Figure 5.1.

Now we should extend this to handle all four of the decision making units.

```
results.efficiency <- matrix(rep(-1.0, 1), nrow=ND, ncol=1)
results.lambda     <- matrix(rep(-1.0, ND), nrow=ND,ncol=ND)
for (k in 1:ND) {
  result <- MIPModel() %>%
    add_variable(vlambda[j], j=1:ND, type = "continuous",
                 lb = 0) %>%
    add_variable(vtheta, type = "continuous") %>%
    set_objective(vtheta, "min") %>%
    add_constraint(sum_expr(vlambda[j]*xdata[j,i], j=1:ND)
                   <= vtheta * xdata[k,i], i = 1:NX) %>%
    add_constraint(sum_expr(vlambda[j] * ydata[j,r], j=1:ND)
                   >= ydata[k,r], r = 1:NY) %>%
    solve_model(with_ROI(solver = "glpk"))

  print(c("DMU=",k,solver_status(result)))
#  print(get_solution(result, vlambda[1:(ND)]) )
    results.efficiency[k] <-  get_solution(result, vtheta)
    results.lambda[k,] <- t(as.matrix(as.numeric(
              get_solution(result, vlambda[j])[,3] )))
# Put resulting from solution in lambda matrix
# Comes out of ompr as a dataframe, use as.matrix
#    to convert from data frame to matrix
#    grabs 3rd column to put into results matrix.
# First two columns of dataframe are for other info
#    as.numeric forces the numbers to treated as
#    numbers rather than text.
# I'm sure there are better ways of handling this
#    but I'll leave that for future work.
}
```

```
## [1] "DMU="    "1"       "optimal"
## [1] "DMU="    "2"       "optimal"
## [1] "DMU="    "3"       "optimal"
## [1] "DMU="    "4"       "optimal"
```

Success! This indicates each of the four linear programs was solved to optimality. By itself, it doesn't help much though. We need to now display each column of results. Lambda, λ, reflects the way that a best target is made for that unit.

```
Lambdanames <- list("$\\lambda_A$", "$\\lambda_B$",
                    "$\\lambda_C$", "$\\lambda_D$")
DMUnames <- list("A", "B", "C", "D")
dimnames(results.efficiency)<-list(DMUnames,"CCR-IO")
dimnames(results.lambda)<-list(DMUnames,Lambdanames)
pander (cbind(results.efficiency, results.lambda),
        caption="Input-Oriented Efficiency Results")
```

Table 5.4: Input-Oriented Efficiency Results

	CCR-IO	λ_A	λ_B	λ_C	λ_D
A	0.75	0	0	0.25	0
B	0.5	0	0	0.3333	0
C	1	0	0	1	0
D	0.8	0	0	1.333	0

The results match those observed graphically but let's discuss them. These results indicate that only DMU C is efficient. Rescaled versions of C could produce the same level of output of A, B, or D, while using just 25%, 50%, and 20% less input respectively. The targets of performance for A and C are constructed by scaling down unit C to a much smaller size, as shown by the the values of λ. In contrast, D's performance is surpassed by a 33% larger version of C.

It is an important step of any DEA study to carefully examine the results. In this case, it might be argued that for certain applications, C's business practices do not scale linearly and therefore could not be assumed to operate at a much smaller or larger size. In that case, we should consider modeling returns to scale.

5.6 Returns to Scale

Let's also add a constraint that will accommodate returns to scale. All it needs to do is constrain the sum of the λ variables equal to 1 enforces variables returns to scale or VRS. or redundant under CRS.

$$
\begin{aligned}
\text{minimize } & \theta \\
\text{subject to } & \sum_{j=1}^{n} \lambda_j = 1 \\
& \sum_{j=1}^{n} x_{i,j}\lambda_j - \theta x_{i,k} + s^x_i = 0 \;\forall\; i \\
& \sum_{j=1}^{n} y_{r,j}\lambda_j - s^y_r = y_{r,k} \;\forall\; r \\
& \lambda_j, s^x_i, s^y_r \geq 0 \;\forall i,\ r,\ j
\end{aligned}
$$

To incorporate this, we can add another constraint to our previous model and solve it. Let's define a parameter, "RTS" to describe which returns to scale assumption we are using and only add the VRS constraint when RTS="VRS".

The other very common returns to scale option is constant returns to scale or CRS. For CRS, you can delete the constraint but it is helpful to maintain a consistent model size for reading out sensitivity information so we can make it a redundant constraint by constraining it to be greater than or equal to zero, $\sum_{j=1}^{n} \lambda_j \geq 0$ Since λ's are by definition non-negative, the sum of λ's is also non-negative and therefore the constraint is superfluous or redundant.

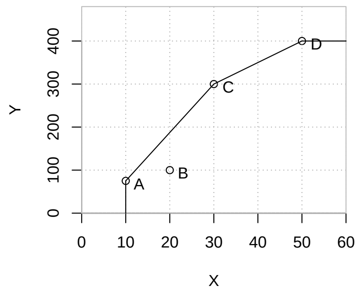
```r
RTS<-"VRS"
for (k in 1:ND) {

  result <- MIPModel() %>%
    add_variable(vlambda[j], j = 1:ND, type = "continuous",
                 lb = 0) %>%
    add_variable(vtheta, type = "continuous") %>%
    set_objective(vtheta, "min") %>%
    add_constraint(sum_expr(vlambda[j] * xdata[j,i],
                            j = 1:ND)
                   <= vtheta * xdata[k,i], i = 1:NX) %>%
    add_constraint(sum_expr(vlambda[j] * ydata[j,r],
                            j = 1:ND)
                   >= ydata[k,r], r = 1:NY)
    if (RTS=="VRS") {result <- add_constraint(result,
                     sum_expr(vlambda[j], j = 1:ND)
                     == 1)
                    }  #Returns to Scale
result <- solve_model(result, with_ROI (solver = "glpk"))

    results.efficiency[k] <-  get_solution(result, vtheta)
    results.lambda[k,] <- t(as.matrix(as.numeric(
              get_solution(result, vlambda[j])[,3] )))
}  # Repeat for each unit, k
Lambdanames <- list("$\\lambda_A$", "$\\lambda_B$",
                    "$\\lambda_C$", "$\\lambda_D$")
DMUnames <- list("A", "B", "C", "D")
dimnames(results.efficiency)<-list(DMUnames,"BCC-IO")
dimnames(results.lambda)<-list(DMUnames,Lambdanames)
pander (cbind(results.efficiency, results.lambda),
        caption="Input-Oriented VRS Envelopment Results")
```

Table 5.5: Input-Oriented VRS Envelopment Results

	BCC-IO	λ_A	λ_B	λ_C	λ_D
A	1	1	0	0	0
B	0.6111	0.8889	0	0.1111	0
C	1	0	0	1	0
D	1	0	0	0	1

Notice that the efficiencies have generally increased or stayed the same. Whereas earlier three out of four DMUs were inefficient, now three out of four are efficient. Much more could be said about returns to scale. One way of thinking of returns to scale is whether doubling the inputs should be expected to result in doubling the outputs that should be achieved. Another way to think of it is whether it is fair to think of scaling up or down an efficient significantly to set a performance target for a much bigger or smaller unit. For example, would it be *fair* to compare a small convenience store such as 7-11 to a CostCo store scaled down by a factor of a 100?

```
library(Benchmarking, quietly=TRUE)
dea.plot(x, y, RTS="vrs", ORIENTATION="in-out",
         txt=LETTERS[1:length(x)],
         add=FALSE, wx=NULL, wy=NULL, TRANSPOSE=FALSE,
         fex=1, GRID=TRUE, RANGE=FALSE, param=NULL)
```



In addition to Constant Returns to Scale (CRS) and Variable Returns to Scale (VRS), two other common approaches are Increasing Returns to Scale (IRS) and Decreasing Returns to Scale (DRS). Technically, IRS is sometimes more formally referred to as non-decreasing returns to scale. Similarly, DRS corresponds to non-increasing returns to scale.

Returns to Scale	Envelopment Constraint
CRS	No constraint needed
VRS	$\sum_{j=1}^{N^D} \lambda_j = 1$
IRS/NDRS	$\sum_{j=1}^{N^D} \lambda_j \geq 1$
DRS/NIRS	$\sum_{j=1}^{N^D} \lambda_j \leq 1$

Now, we will generalize this by allowing a parameter to set the returns to scale.

```
RTS<-"VRS"
for (k in 1:ND) {

  result <- MIPModel() %>%
    add_variable(vlambda[j], j = 1:ND, type = "continuous",
                 lb = 0) %>%
    add_variable(vtheta, type = "continuous") %>%
    set_objective(vtheta, "min") %>%
    add_constraint(sum_expr(vlambda[j] * xdata[j,i],
                            j = 1:ND)
                   <= vtheta * xdata[k,i], i = 1:NX) %>%
    add_constraint(sum_expr(vlambda[j] * ydata[j,r],
                            j = 1:ND)
                   >= ydata[k,r], r = 1:NY)
    if (RTS=="VRS") {result <-
      add_constraint(result, sum_expr(vlambda[j],
                                      j = 1:ND) == 1) }
    if (RTS=="IRS") {result <-
      add_constraint(result, sum_expr(vlambda[j],
                                      j = 1:ND) >= 1) }
    if (RTS=="DRS") {result <-
      add_constraint(result, sum_expr(vlambda[j],
                                      j = 1:ND) <= 1) }
result <- solve_model(result, with_ROI(solver = "glpk"))

    results.efficiency[k] <-  get_solution(result, vtheta)
    results.lambda[k,] <- t(as.matrix(as.numeric(
              get_solution(result, vlambda[j])[,3] )))
}
Lambdanames <- list("$\\lambda_A$", "$\\lambda_B$",
                    "$\\lambda_C$", "$\\lambda_D$")
DMUnames <- list("A", "B", "C", "D")
dimnames(results.efficiency)<-list(DMUnames,"BCC-IO")
dimnames(results.lambda)<-list(DMUnames,Lambdanames)
pander (cbind(results.efficiency, results.lambda),
        caption="Input-Oriented BCC Model with Option
        of Alternate Returns to Scale")
```

Table 5.7: Input-Oriented BCC Model with Option of Alternate Returns to Scale

	BCC-IO	λ_A	λ_B	λ_C	λ_D
A	1	1	0	0	0
B	0.6111	0.8889	0	0.1111	0
C	1	0	0	1	0
D	1	0	0	0	1

5.7 *Multiple Inputs and Multiple Ouptuts*

Using DEA for two-dimensional examples such as the one-input, one-output model is easy to draw and visualize but overkill and not generally very useful.

Let's create a richer dataset. For this example, we will use the dataset from Kenneth Baker's third edition of *Optimization Modeling with Spreadsheets*, pages 175-178, Example 5.3 titled "Hope Valley Health Care Association." In this example, a health care organization wants to benchmark six nursing homes against each other.

This data is a little more complicated so let's make sure that we understand the definitions. Look at the line for the inputs which we will call XBaker1. The list of 12 data values has the structure defined as two columns for two separate inputs by use of 'ncol=2'. This then means that there are 6 rows or DMUs in this dataset. The six units are given capital letters for names starting with A. The names of the inputs are hard coded as "x1" and"x2" to represent the *staff hours per day* and the *supplies per day* respectively.

```
  XBaker1 <- matrix(c(150, 400, 320, 520, 350, 320, .2,
                      0.7, 1.2, 2.0, 1.2, 0.7),
                    ncol=2,dimnames=list(LETTERS[1:6],
                                         c("x1", "x2")))
```

The two outputs of *reimbursed patient-days* and *privately paid patient-days* are named "y1" and "y2" but otherwise have the same structure as the inputs.

```
YBaker1 <- matrix(c(14000, 14000, 42000, 28000, 19000,
                    14000, 3500, 21000, 10500, 42000,
                    25000, 15000), ncol=2,
                  dimnames=list(LETTERS[1:6],c("y1", "y2")))
ND <- nrow(XBaker1); NX <- ncol(XBaker1);
NY <- ncol(YBaker1);
pander(cbind(XBaker1,YBaker1))
```

	x1	x2	y1	y2
A	150	0.2	14000	3500
B	400	0.7	14000	21000
C	320	1.2	42000	10500
D	520	2	28000	42000
E	350	1.2	19000	25000
F	320	0.7	14000	15000

Note that I'm naming the data sets based on their origin and then loading them into xdata and ydata for actual operation. This allows the model to be generalized. The actual returns to scale models are straightforward to implement.

```
xdata      <-XBaker1[1:ND,]  # Call it xdata
dim(xdata) <-c(ND,NX)  # structure data correctly
ydata      <-YBaker1[1:ND,]
dim(ydata) <-c(ND,NY)
# Need to remember to restructure the results matrices.
res.efficiency <- matrix(rep(-1.0, ND), nrow=ND, ncol=1)
res.lambda     <- matrix(rep(-1.0, ND^2), nrow=ND,ncol=ND)
res.xslack     <- matrix(rep(-1.0, ND*NX), nrow=ND,ncol=NX)
res.yslack     <- matrix(rep(-1.0, ND*NY), nrow=ND,ncol=NY)
res.vweight    <- matrix(rep(-1.0, ND*NX), nrow=ND,ncol=NX)
res.uweight    <- matrix(rep(-1.0, ND*NY), nrow=ND,ncol=NY)
DMUnames    <- list(c(LETTERS[1:ND]))
Xnames      <- lapply(list(rep("X",NX)),paste0,1:NX)
Ynames      <- lapply(list(rep("Y",NY)),paste0,1:NY)
Vnames      <- lapply(list(rep("v",NX)),paste0,1:NX)
Unames      <- lapply(list(rep("u",NY)),paste0,1:NY)
SXnames     <- lapply(list(rep("sx",NX)),paste0,1:NX)
SYnames     <- lapply(list(rep("sy",NY)),paste0,1:NY)
Lambdanames <- lapply(list(rep("$\\lambda_",ND)),
                      paste0,lapply(LETTERS[1:ND],
                                    paste0,"$"))

dimnames(xdata)              <- c(DMUnames,Xnames)
dimnames(ydata)              <- c(DMUnames,Ynames)
dimnames(res.efficiency) <- c(DMUnames,"CCR-IO")
dimnames(res.lambda)     <- c(DMUnames,Lambdanames)
dimnames(res.xslack)     <- c(DMUnames,SXnames)
dimnames(res.yslack)     <- c(DMUnames,SYnames)
dimnames(res.vweight)    <- c(DMUnames,Vnames)
dimnames(res.uweight)    <- c(DMUnames,Unames)
```

We are now ready to do the analysis. Note that Baker's analysis uses the multiplier model and then extracts the lambda values from the sensitivity report. This generally yields the same results. We will discuss the multiplier model in more detail later.

```
RTS<-"CRS"
for (k in 1:ND) {

  modBakerCCR <- MIPModel() %>%
    add_variable(vlambda[j], j = 1:ND, type = "continuous",
                 lb = 0) %>%
    add_variable(vtheta, type = "continuous") %>%
    set_objective(vtheta, "min") %>%
    add_constraint(sum_expr(vlambda[j] * xdata[j,i],
                            j = 1:ND)
                   <= vtheta * xdata[k,i], i = 1:NX) %>%
    add_constraint(sum_expr(vlambda[j] * ydata[j,r],
                            j = 1:ND)
                   >= ydata[k,r], r = 1:NY)
    if (RTS=="VRS") {result <- add_constraint(result,
                 sum_expr(vlambda[j],j = 1:ND) == 1) }
        #Returns to Scale
res <- solve_model(modBakerCCR, with_ROI(solver = "glpk"))

    res.efficiency[k] <-  get_solution(res, vtheta)
    res.lambda[k,] <- t(as.matrix(as.numeric(
                     get_solution(res, vlambda[j])[,3] )))
}
pander (cbind(res.efficiency, res.lambda),
        caption="Results from Baker's Example (CRS)")
```

Table 5.9: Results from Baker's Example (CRS) (continued below)

	CCR-IO	λ_A	λ_B	λ_C	λ_D
A	1	1	0	0	0
B	1	0	1	0	-2.427e-32
C	1	-2.151e-17	0	1	-4.66e-17
D	1	0	0	0	1
E	0.9775	0.2	0.08048	0	0.5383
F	0.8675	0.3429	0.395	0	0.1311

	λ_E	λ_F
A	0	0
B	0	0
C	0	0
D	0	0
E	0	0
F	0	0

```
BakerCCR.Res<-cbind(res.efficiency, res.lambda)
```

The efficiency scores match those reported in Baker, page 176, Figure 5.8. The columns to the right of the efficiency score are the lambdas. The lambdas for facility 5, (E), are given on page 177 of Baker and match the fifth row of our results.

In contrast, some values of λ warrant a more careful look, in particular, there may be very small positive or negative numbers. These numbers are due to computational issues in the linear programming solvers and should be interpreted as zero. Rounding the results can make the tables easier for interpretation by the user. The pander packages makes it easy to round numbers to varying levels of precision.

```
pander (cbind(res.efficiency, res.lambda), round = 6,
        caption="Results from Baker's Example (CRS)
        with Rounded Values")
```

Table 5.11: Results from Baker's Example (CRS) with Rounded Values (continued below)

	CCR-IO	λ_A	λ_B	λ_C	λ_D	λ_E	λ_F
A	1	1	0	0	0	0	0
B	1	0	1	0	0	0	0
C	1	0	0	1	0	0	0
D	1	0	0	0	1	0	0
E	0.9775	0.2	0.08048	0	0.5383	0	0
F	0.8675	0.3429	0.395	0	0.1311	0	0

The small, nearly zero, values that may appear in DEA calculations can cause some computational difficulty. For example, testing for zero lambda values could miss cases where the value is approximately but not exactly zero.

Let's compare the results from the constant and variable returns to scale cases.

```
RTS<-"VRS"
for (k in 1:ND) {

  modBakerBCC <- MIPModel() %>%
    add_variable(vlambda[j], j = 1:ND, type="continuous",
                 lb = 0) %>%
    add_variable(vtheta, type = "continuous") %>%
    set_objective(vtheta, "min") %>%
    add_constraint(sum_expr(vlambda[j] * xdata[j,i],
                            j = 1:ND)
                   <= vtheta * xdata[k,i], i = 1:NX) %>%
    add_constraint(sum_expr(vlambda[j] * ydata[j,r],
                            j = 1:ND)
                   >= ydata[k,r], r = 1:NY)
    if (RTS=="VRS") {modBakerBCC <-
      add_constraint(modBakerBCC,
                 sum_expr(vlambda[j],j = 1:ND) == 1) }
                 #Returns to Scale
res <- solve_model(modBakerBCC, with_ROI(solver = "glpk"))

    res.efficiency[k] <- get_solution(res, vtheta)
    res.lambda[k,] <- t(as.matrix(as.numeric(
                        get_solution(res, vlambda[j])[,3] )))
}
BakerBCC.Res<-cbind(res.efficiency, res.lambda)
colnames(BakerBCC.Res)[1] <- "BCC-IO"
```

```
pander(cbind(BakerCCR.Res[,1, drop=FALSE],
             BakerBCC.Res), round=6,
       caption=
         "Comparison of CRS vs. VRS Efficiency Scores")
```

Table 5.13: Comparison of CRS vs. VRS Efficiency Scores (continued below)

	CCR-IO	BCC-IO	λ_A	λ_B	λ_C
A	1	1	1	0	0
B	1	1	0	1	0

	CCR-IO	BCC-IO	λ_A	λ_B	λ_C
C	1	1	0	0	1
D	1	1	0	0	0
E	0.9775	1	0	0	0
F	0.8675	0.8963	0.4014	0.3423	0

	λ_D	λ_E	λ_F
A	0	0	0
B	0	0	0
C	0	0	0
D	1	0	0
E	0	1	0
F	0	0.2563	0

Hint: drop=FALSE in the subsetting of BakerCCR.Res enables retaining the original structure of the matrix, which allows the column name to remain. The default is drop=TRUE which simplifies the data object into a form that doesn't have a column name.

A few takeaways from these results:

- Switching from CRS to VRS will never hurt the efficiency of a DMU. From an optimization perspective, think of this as adding a constraint which will increase or leave unchanged the objective function value from a minimization linear program.
- Another way to view the impact of adding a returns to scale assumption is that this makes it *easier* for a DMU to find some way of being *best* or efficient. They can do this by now having the largest of any output or the smallest of input as well as many other ways.
- The number of DMUs that are used for setting performance targets for inefficient units is typically broadened. This is visible in the non-zero λ values for unit E
- At a simple level, the CCR (CRS) efficiency score can be thought of as a combination of how efficient someone is considering their operational and size efficiency. The BCC (VRS) efficiency is how efficient they are given their current operating size.

5.8 *Extracting Multiplier Weights from Envelopment Results*

Recently, Dirk Schumacher has added the option of extraction dual results from glpk-solved linear programs. As of April, 2018, it is in the developmental version of ompr which is installed using devtools.

```r
rduals <- as.matrix(get_row_duals(res))
res.vweight[k,] <- -1*rduals[1:NX]
   # Extract first NX row duals assuming first set of
   #    constraints for inputs
   # Multiply input weights by -1 to adjust
   #    for inequality direction
res.uweight[k,] <- rduals[(NX+1):(NX+NY)]
   # Extract next NY row duals, assuming output
   #    constraints follow inputs
pander(c(res.vweight[k,],res.uweight[k,]),
       caption="Input and output weights for last DMU")
```

v1	v2	u1	u2
0.001944	0.54	0	4.32e-05

These results are for just the last unit examined.

Let's now wrap this into the full series of linear programs so as to calculate the multiplier weights for each unit.

```r
RTS<-"CRS"
for (k in 1:ND) {

  modBakerCRS <- MIPModel() %>%
    add_variable(vlambda[j], j = 1:ND, type = "continuous",
                 lb = 0) %>%
    add_variable(vtheta, type = "continuous") %>%
    set_objective(vtheta, "min") %>%
    add_constraint(sum_expr(vlambda[j] * xdata[j,i],
                            j = 1:ND)
                   <= vtheta * xdata[k,i], i = 1:NX) %>%
    add_constraint(sum_expr(vlambda[j] * ydata[j,r],
                            j = 1:ND)
                   >= ydata[k,r], r = 1:NY)
    if (RTS=="VRS") {modBakerCRS <-
      add_constraint(modBakerCRS,
                 sum_expr(vlambda[j],j = 1:ND) == 1) }
resBakerCRS <- solve_model(modBakerCRS,
                           with_ROI(solver = "glpk"))

    res.efficiency[k] <-  get_solution(resBakerCRS, vtheta)
    res.lambda[k,] <- t(as.matrix(as.numeric(
                     get_solution(resBakerCRS,
                                          vlambda[j])[,3] )))
    rduals  <- as.matrix(get_row_duals(resBakerCRS))
    res.vweight[k,] <- -1*rduals[1:NX]
      # Extract first NX row duals
      #   Assumes first set of constraints for inputs
      # Multiply input weights by -1 to adjust for
      #   inequality direction
    res.uweight[k,] <- rduals[(NX+1):(NX+NY)]
      # Extract next NY row duals
      #   Assumes output constraints follow inputs
    }
pander (cbind(res.efficiency, res.vweight,
             res.uweight),round=6,
        caption="Results from Baker's Example")
```

Table 5.16: Results from Baker's Example

	CCR-IO	v1	v2	u1	u2
A	1	0.005172	1.121	5e-05	8.4e-05
B	1	0.001376	0.6422	0	4.8e-05
C	1	0.001724	0.3736	1.7e-05	2.8e-05
D	1	0.001923	0	0	2.4e-05
E	0.9775	0.001099	0.5128	1.2e-05	3e-05
F	0.8675	0.001546	0.7216	1.6e-05	4.3e-05

A few things should be noticed with respect the the multiplier weights:

- Multiplier weights can be quite small because in general, they are multiplied by inputs and the product is typically less than one. They may appear to round to zero but still be significant.
- Multiplier weights are not unique for strongly efficient DMUs. This is explored in more detail in the multiplier chapter.
- Multiplier weights are usually unique for inefficient DMUs and should match the results obtained using either the row duals of the envelopment model or those obtained directly from the multiplier model.

5.9 Slack Maximization

Situations can arise where units may appear to be radially efficient but can still find opportunities to improve one or more inputs or outputs.

This is defined as weakly efficient.

In order to accommodate this, we need to extend the simple radial model by adding variables to reflect nonradial slacks. We do this by converting the model's input and output constraints from inequalities into equalities by explicitly defining slack variables.

$$\begin{aligned} &\text{minimize } \theta \\ &\text{subject to } \sum_{j=1}^{N^D} x_{i,j}\lambda_j - \theta x_{i,k} + s_i^x = 0 \;\forall\; i \\ &\sum_{j=1}^{N^D} y_{r,j}\lambda_j - s_r^y = y_{r,k} \;\forall\; r \\ &\lambda_j, s_i^x, s_r^y \geq 0 \;\forall\; j, i, r \end{aligned}$$

Simply formulating the model with slacks is insufficient. We want to maximize these slacks after having found the best possible radial contraction (minimum value of θ.) This is done by adding a term summing the slacks to the objective function. Note that this sum of slacks is multiplied by ϵ which is a non-Archimedean infinitesimal.

The value of ϵ should be considered to be so small as to ensure that minimizing theta takes priority or maximizing the sum of slacks. Note also that maximizing the sum of slacks is denoted by minimizing the negative sum of slacks.

$$\begin{aligned} &\text{minimize } \theta - \epsilon(\sum_i s_i^x + \sum_r s_r^y) \\ &\text{subject to } \sum_{j=1}^{N^D} x_{i,j}\lambda_j - \theta x_{i,k} + s_i^x = 0 \;\forall\; i \\ &\sum_{j=1}^{N^D} y_{r,j}\lambda_j - s_r^y = y_{r,k} \;\forall\; r \\ &\lambda_j, s_i^x, s_r^y \geq 0 \;\forall\; j, i, r \end{aligned}$$

A common mistake in implementing DEA is to use a finite approximation for ϵ such as 10^{-6}. Any finite value can cause distortions in θ. For example, an application comparing companies using revenue and expenses might have inputs and outputs on the order of millions or billions. In this case, non-radial slacks could also be on the order of 10^6. Multiplying the two results in a value similar in magnitude to the maximum possible efficiency score ($10^{-6}10^6 = 1$) which would then potentially overwhelm the radial efficiency, θ, part of the objective function and lead to distorted results.

The proper way to implement slack maximization is to treat it as a preemptive goal programming problem. The primary goal is to minimize θ in a first phase linear program and the second goal, holding the level of θ fixed from the first phase, is to then maximize the sum of the slacks.

The first phase can take the form of any of our earlier linear programs without the ϵ and sum of slacks in the objective function. The second phase is the following where θ^* is the optimal value from phase one and θ is then held constant in the second phase. This is implemented by adding a constraint, $\theta = \theta^*$ to the second phase linear program.

$$
\begin{aligned}
\text{maximize } & \sum_{i} s^x_i + \sum_{r} s^y_r \\
\text{subject to } & \sum_{j=1}^{N^D} \lambda_j = 1 \\
& \sum_{j=1}^{N^D} x_{i,j}\lambda_j - \theta x_{i,k} + s^x_i = 0 \; \forall \; i \\
& \sum_{j=1}^{N^D} y_{r,j}\lambda_j - s^y_r = y_{r,k} \; \forall \; r \\
& \theta = \theta^* \\
& \lambda_j, s^x_i, s^y_r \geq 0 \; \forall \; j, i, r
\end{aligned}
$$

Implementing this algebraically is quite straightforward.

```r
RTS<-"CRS"
for (k in 1:ND) {

  LPSlack <- MIPModel() %>%
    add_variable(vlambda[j], j = 1:ND, type = "continuous",
                 lb = 0) %>%
    add_variable(vtheta, type = "continuous") %>%
    add_variable(xslack[i], i = 1:NX, type = "continuous",
                 lb=0) %>%
    add_variable(yslack[r], r = 1:NY, type = "continuous",
                 lb=0) %>%

    set_objective(vtheta, "min") %>%

    add_constraint(sum_expr(vlambda[j] * xdata[j,i] +
                            xslack[i], j = 1:ND)
```

```
                   - vtheta * xdata[k,i]==0, i = 1:NX) %>%

    add_constraint(sum_expr(vlambda[j] * ydata[j,r] -
                                yslack[r], j = 1:ND)
                    ==ydata[k,r], r = 1:NY)
    if (RTS=="VRS") {LPSlack<-
      add_constraint(LPSlack, sum_expr(vlambda[j],
                                         j = 1:ND) == 1) }
  #Returns to Scale
    result <- solve_model(LPSlack, with_ROI(solver = "glpk"))
    # The following are key steps to slack maximization
    phase1obj <-  get_solution(result, vtheta)
    # Get Phase 1 objective value
    add_constraint(LPSlack, vtheta==phase1obj)
        # Passing result from phase 1 to phase 2
    set_objective(LPSlack, sum_expr(
              xslack[i], i=1:NX)+sum_expr(
                yslack[r], r=1:NY), "max")
          # Modify the objective function for phase 2
    result <- solve_model(LPSlack, with_ROI(solver = "glpk"))

    res.efficiency[k] <-  get_solution(result, vtheta)
    res.lambda[k,] <- t(as.matrix(
                 as.numeric(get_solution(result,
                                           vlambda[j])[,3] )))
}
rownames(res.efficiency)<-c("A", "B", "C", "D", "E", "F")
#print (results.efficiency)
pander(cbind(res.efficiency,xdata,ydata,res.lambda),
       round=5, caption="Input-oriented Efficiency Results")
```

Table 5.17: Input-oriented Efficiency Results (continued below)

	CCR-IO	X1	X2	Y1	Y2	λ_A	λ_B
A	1	150	0.2	14000	3500	1	0
B	1	400	0.7	14000	21000	0	1
C	1	320	1.2	42000	10500	0	0
D	1	520	2	28000	42000	0	0
E	0.9775	350	1.2	19000	25000	0.2	0.08048
F	0.8675	320	0.7	14000	15000	0.3429	0.395

	λ_C	λ_D	λ_E	λ_F
A	0	0	0	0
B	0	0	0	0
C	1	0	0	0
D	0	1	0	0
E	0	0.5383	0	0
F	0	0.1311	0	0

5.10 Further Reading

This section covers the second chapter of the book and introduces DEA and R working through a model and looping through a series of analyses. The interested reader is referred to the book *DEA Using R* for more information on DEA. In particular, chapters covers topics such as:

- a matrix-oriented implementation of DEA,
- output-orientation (maximizing the production of outputs)
- multiplier model (weighting inputs and weighting models)
- an easy to understand application applied to over 100 years of data
- examples of previously published applications applied to R&D project evaluation.

5.11 Exercises

Exercise 5.1 (Adding DMU E-Graphically)**.**

Add a fifth unit, E, to the first example that produces 400 units output using 30 units of input. Graphically evaluate all five units for their efficiency scores and lambda values. Interpret the solution in terms of who is doing well, who is doing poorly, and who should be learning from whom.

Exercise 5.2 (Adding DMU E-Using R)**.**

Examine the new unit, E, using R. Interpret the solution in terms of who is doing well, who is doing poorly, and who should be learning from whom.

Exercise 5.3 (Looping)**.**

Wrap a for loop around the model to examine every unit. Discuss results.

Exercise 5.4 (Bigger Data).

Use a bigger data set and conduct an analysis & interpretation (more inputs, outputs, and units.)

Exercise 5.5 (Compare Results).

Check results against a DEA package (ex. DEAMultiplier, TFDEA, Benchmarking, nonparaeff).

Exercise 5.6 (Slacking).

Construct an example where Phase 2 increases positive slacks from Phase 1.

Exercise 5.7 (Graphics).

Create "cool" graphs or plots of results.

To pass the challenges, work on extending my RMarkdown file or using a similar script. You can use my *.rmd file as a starting point. Others might prefer to create a well documented R script instead of using RMarkdown. Others might even prefer using LaTeX. If you use RMarkdown or LaTeX, please use section headings to indicate each challenge solved. You can use other packages for graphics or data manipulation but don't use a DEA package.

Exercise 5.8 (A DEA Application).

Conduct a DEA study for an application that you are personally familiar with. (Pick one for which data is readily available but something that you are passionate about. It can have scrubbed or anonymized data. Examples might include your favorite sports league, team salaries, coaching salaries, wins, etc. or it can be USAF related.)

a) Describe the application.
b) Describe and justify the data including the inputs and outputs used as well as items explicitly not used.
c) Select an appropriate DEA model and conduct the analysis.
d) Discuss the results.

6 Mixed Integer Optimization

Up until this point, we have always assumed that variables could take on fractional (or non-integer) values. In other words, they were continuous variables. Sometimes the values conveniently turned out to be integer-valued, in other cases, we would brush off the non-integer values of the results.

Allowing for integer values opens up many more important areas of applications as we will see.

Let's look at two numerical examples. The first shows a case where integrality makes little difference, the second where it has a major impact.

Unfortunately, there is no such thing as a free lunch-adding integrality can make problems much more computationally demanding. We will go through an example of how many optimization routines do integer optimization to demonstrate why it can be more difficult algorithmically even if looks trivially easy from the perspective of the change in the ompr function.

6.1 Example of Integrality having Little Impact

Let's revisit the earlier example of producing chairs, desks, and tables. Where we add one extra hour of labor to the machining center. The resulting LP is shown below.

$$
\begin{aligned}
\text{Max } & 20Chairs + 14Desks + 16Tables \\
\text{subject to } & 6Chairs + 2Desks + 4Tables \leq 2000 \\
& 8Chairs + 6Desks + 4Tables \leq 2000 \\
& 6Chairs + 4Desks + 8Tables \leq 1441 \\
& 40Chairs + 25Desks + 16Tables \leq 9600 \\
& Tables \leq 200 \\
& Chairs,\ Desks,\ Tables \geq 0
\end{aligned}
$$

We used the following implementation.

```
BaseModelCDT <- MIPModel() %>%
  add_variable(Chairs, type = "continuous", lb = 0) %>%
  add_variable(Desks, type = "continuous",lb = 0) %>%
  add_variable(Tables, type = "continuous", lb = 0) %>%

  set_objective(20*Chairs + 14*Desks + 16*Tables, "max") %>%

  add_constraint(6*Chairs+2*Desks+4*Tables<=2000) %>%
                                    #fabrication
  add_constraint(8*Chairs+6*Desks+4*Tables<=2000) %>%
                                    #assembly
  add_constraint(6*Chairs+4*Desks+8*Tables<=1441) %>%
                                    #machining
  add_constraint(40*Chairs+25*Desks+16*Tables<= 9600) %>%
                                    #wood
  add_constraint(Tables <= 200) %>% #
  solve_model(with_ROI(solver = "glpk"))
  obj_val <- objective_value(BaseModelCDT)
  xchairs <- get_solution (BaseModelCDT, Chairs)
  xdesks  <- get_solution (BaseModelCDT, Desks)
  xtables <- get_solution (BaseModelCDT, Tables)
inc_mc_res  <- cbind(xchairs,xdesks,xtables,obj_val)
rownames(inc_mc_res) <- ""
pander(inc_mc_res,
       caption="Production Plan with Continuous Variables")
```

Table 6.1: Production Plan with Continuous Variables

xchairs	xdesks	xtables	obj_val
161.5	118	0	4882

In this case, the fractional value of *Chairs* would be only somewhat concerning. We wouldn't actually ship a half of a chair to a customer-at least I hope not. The difference between 162 and 163 is relatively small over a month of production. This is a difference of less than 1% and none of the numbers specified in the model appear to be reported to more than two significant digits or this level of precision. The result is that we could specify the answer as 162.5 and be satisfied that while it is our actual best guess or might reflect a chair half finished for the next month, it isn't a major concern.

For the sake of illustration, let's show how we would modify the model to be integers. If we had wanted to modify the problem to force the amount of each item to be produced to be an integer value, we can modify our formulation and the implementation with only a few small changes.

In the formulation, we can replace the non-negativity requirement with a restriction that the variables are drawn from the set of non-negative integers.

$$\begin{aligned}
\text{Max } & 20Chairs + 14Desks + 16Tables \\
\text{subject to } & 6Chairs + 2Desks + 4Tables \leq 2000 \\
& 8Chairs + 6Desks + 4Tables \leq 2000 \\
& 6Chairs + 4Desks + 8Tables \leq 1441 \\
& 40Chairs + 25Desks + 16Tables \leq 9600 \\
& Tables \leq 200 \\
& Chairs,\ Desks,\ Tables \in \{0, 1, 2, 3, ...\}
\end{aligned}$$

The ompr implementation has a similar, straightforward change. In the declaration of variables (add_variable function), we simply set the type to be integer rather than continuous.

```
IntModelCDT <- MIPModel() %>%
  add_variable(Chairs, type = "integer", lb = 0) %>%
  add_variable(Desks, type = "integer",lb = 0) %>%
  add_variable(Tables, type = "integer", lb = 0) %>%

  set_objective(20*Chairs+14*Desks+16*Tables,"max") %>%

  add_constraint(6*Chairs+2*Desks+4*Tables<=2000) %>%
                                      #fabrication
  add_constraint(8*Chairs+6*Desks+4*Tables<=2000) %>%
                                      #assembly
  add_constraint(6*Chairs+4*Desks+8*Tables<=1441) %>%
                                    #machining
  add_constraint(40*Chairs+25*Desks+16*Tables<= 9600) %>%
                                    #wood
  add_constraint(Tables <= 200) %>% #
  solve_model(with_ROI(solver = "glpk"))
  obj_val <- objective_value(IntModelCDT)
  xchairs <- get_solution (IntModelCDT, Chairs)
  xdesks  <- get_solution (IntModelCDT, Desks)
  xtables <- get_solution (IntModelCDT, Tables)
IntModelCDTres  <- cbind(xchairs,xdesks,xtables,obj_val)
rownames(IntModelCDTres) <- ""
pander(IntModelCDTres,
     caption="Production Plan with Integer Variables")
```

Table 6.2: Production Plan with Integer Variables

xchairs	xdesks	xtables	obj_val
160	120	0	4880

Notice that in this case, there was a small adjustment to the production plan and a small decrease to the optimal objective function value. This is not always the case, sometimes the result can be unchanged. In other cases, it may be changed in a very large way. In still others, the problem may in fact even become infeasible.

6.2 Example of Integality having a Large Impact

Let's take a look at another example. Acme makes two products, let's refer to them as product 1 and 2. Product 1 generates a profit of $2000 per product, requires one liter of surfactant for cleaning, and 2.5 kilograms of high grade steel.

In contrast, each unit of Product 2 produced generates a higher profit of $3000 per product, requires 3.0 liters of surfactant, and 1.0 kilograms of high grade steel. Acme has 8.25 liters of surfactant and 8.75 kilograms of high grade steel.

Only completed products are sellable, what should be Acme's production plan?

Formulating the optimization problem is similar to our earliest production planning problems. Let's define x_1 to be the amount of product 1 to produce and x_2 to be the amount of product 2 to produce.

$$\begin{split}
\text{Max } & 2000x_1 + 3000x_2 \\
\text{subject to } & 1.0x_1 + 3.0x_2 \leq 8.25 \\
& 2.5x_1 + 1.0x_2 \leq 8.75 \\
& x_1,\ x_2 \geq 0 \\
& x_1,\ x_2 \in \{0, 1, 2, 3, ...\}
\end{split}$$

The last constraint cannot be solved directly strictly using linear programming. It is instead referred to as an integer linear program or ILP.

Again, we could try solving it with and without the imposition of the integrality constraint.

```
LPRelax <- MIPModel() %>%
  add_variable(xx1, type = "continuous", lb = 0) %>%
  add_variable(xx2, type = "continuous",lb = 0) %>%
  set_objective(2*xx1 + 3*xx2, "max") %>%
  add_constraint(1.0*xx1 + 3.0*xx2 <= 8.25) %>% #Surfactant
  add_constraint(2.5*xx1 + 1.0*xx2  <= 8.75) %>% #Steel
  solve_model(with_ROI(solver = "glpk"))
  obj_val <- objective_value(LPRelax)
  x1 <- get_solution (LPRelax, 'xx1')
  x2  <- get_solution (LPRelax, 'xx2')
acme_res_lp  <- cbind(x1,x2,obj_val)
colnames(acme_res_lp) <- list("x1", "x2", "Profit")
rownames(acme_res_lp) <- "LP Solution"
```

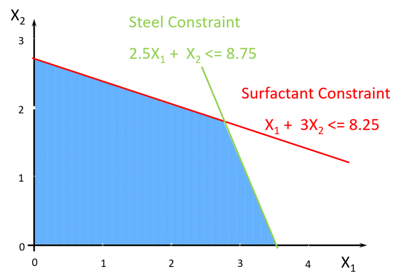
```
IPSoln <- MIPModel() %>%
  add_variable(xx1, type = "integer", lb = 0) %>%
  add_variable(xx2, type = "integer",lb = 0) %>%
  set_objective(2*xx1 + 3*xx2, "max") %>%
  add_constraint(1.0*xx1 + 3.0*xx2 <= 8.25) %>% #Surfactant
  add_constraint(2.5*xx1 + 1.0*xx2  <= 8.75) %>% #Steel
  solve_model(with_ROI(solver = "glpk"))
  obj_val <- objective_value(IPSoln)
  x1 <- get_solution (IPSoln, 'xx1')
  x2  <- get_solution (IPSoln, 'xx2')
acme_res_ip  <- cbind(x1,x2,obj_val)
colnames(acme_res_ip) <- list("x1", "x2", "Profit")
rownames(acme_res_ip) <- "IP Solution"
```

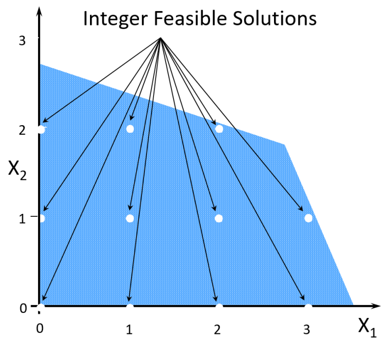
```
pander(rbind(acme_res_lp,acme_res_ip),
       caption="Production Plan-Continuous vs. Integer")
```

Table 6.3: Production Plan-Continuous vs. Integer

	x1	x2	Profit
LP Solution	2.769	1.827	11.02
IP Solution	2	2	10

Simply rounding down did not yield an optimal solution. Notice that production of product 1 went down while product 2 went up. The net impact was a profit reduction of almost *10%*.





6.3 *The Branch and Bound Algorithm*

The simplicity of making the change in the model from continuous to integer variables hides or understates the complexity of what needs to be done to computationally to solve the optimization problem. This small change can have major impacts on the computational effort needed to solve a problem. Solving linear programs with tens of thousands of continuous variables is very quick and easy computationally. Changing those same variables to general integers can make it *very* difficult. Why is this the case?

At its core, the basic problem is that the simplex method and linear programming solves a system of equations and unknowns (or variables). It is trivial to solve one equation with one unknown such as $5x = 7$. It is only slightly harder to solve a system of two equations and two unknowns. Doing three equations and three unknowns requires some careful algebra but is not too hard. In general, solving n equations and n unknowns is readily solvable. On the other hand, none of this algebraic work of solving for unknown variables can require that the variables take on integer values. Even with only one equation and one unknown, x will only be integer for certain combinations of numbers in the above equation. Specifically when the number on the right is a multiple of the number in front of x. It gets even less likely to have an all integer solution as the number of equations and unknowns increase.

Another way to visualize integer solutions for general systems of inequalities is to think of the two equation and two unknown (x and y) case. Finding a solution is the same as finding where two lines intersect in two dimensional space. The odds of getting lucky and having the two lines intersect at an integer solution is small. Even if we limit ourselves to x and y both being between 0 and 10, this means that there are $11^2 = 121$ possible integer valued points in this space such as (0,0), (0,3), and (10,10). In contrast, the set of all points, including non-integers, in this region includes all of the above points as well as (0,0.347) and (4.712, 7.891) among infinitely more. The result is that if you think of this as a target shooting, the probability of finding an integer solution by blind luck is $\frac{121}{\infty} \approx 0$.

What we need to do is an algorithm around the simplex method to accomplish this. A basic and common and approach often used for solving integer variable linear programming problems is called branch and bound. We start with a solution that has presumably continuous valued variables. We then branch (create subproblems) off of this result and add bounds (additional constraints) to rule out the fractional value of that variable.

The LP Relaxation

To solve this using branch and bound, we *relax* the integrality requirements and solve what is called the LP Relaxation. To do this, we simply remove the integrality requirement.

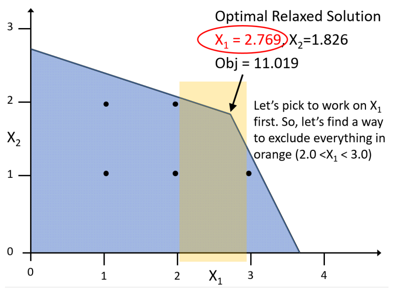
$$\begin{aligned} \text{Max } & 2000x_1 + 3000x_2 \\ \text{subject to } & 1.0x_1 + 3.0x_2 \leq 8.25 \\ & 2.5x_1 + 1.0x_2 \leq 8.75 \\ & x_1,\ x_2 \geq 0 \end{aligned}$$

This is a problem that we **can** solve using linear programming so let's go ahead do it! Well, we already did so let's just review the results for what is now called the LP Relaxation because we are *relaxing* the integrality requirements.

```
pander(acme_res_lp,
       caption="Acme's Production Plan based
       on the LP Relaxation")
```

Table 6.4: Acme's Production Plan based on the LP Relaxation

	x1	x2	Profit
LP Solution	2.769	1.827	11.02



Subproblem I

Alas, this production plan from the LP Relaxation is not feasible from the perspective of the original integer problem because it produces *2.769* of product 1 and *1.827* of product 2. If both of these variables had been integer we could have declared victory and been satisfied that we had easily (*luckily?*) found the optimal solution to the original integer programming problem so quickly.

Instead, we will need to proceed with the branch and bound process. Since both of the variables in the LP relaxation have fractional values, we need to start by choosing which variable to use for branching first. Algorithmic researchers would focus on how to best pick a branch but for our purposes to improve solution speed but it doesn't really matter for illustration so let's arbitrarily choose to branch on x_1.

For the branch and bound algorithm, we want to include two subproblems that exclude the *illegal* value of $x_1 = 2.769$ as well as everything else between 2 and 3. We say that we are *branching* on x_1 to create two subproblems. The first subproblem (I) has an added constraint of $x_1 \leq 2.0$ and the other subproblem (II) has an added constraint of $x_1 \geq 3.0$.

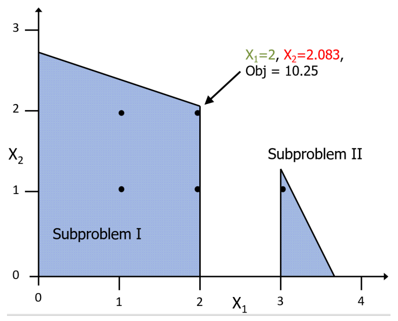
$$\begin{aligned}
\text{Max } & 2000x_1 + 3000x_2 \\
\text{subject to } & 1.0x_1 + 3.0x_2 \leq 8.25 \\
& 2.5x_1 + 1.0x_2 \leq 8.75 \\
& x_1 \leq 2.0 \\
& x_1,\ x_2 \geq 0
\end{aligned}$$

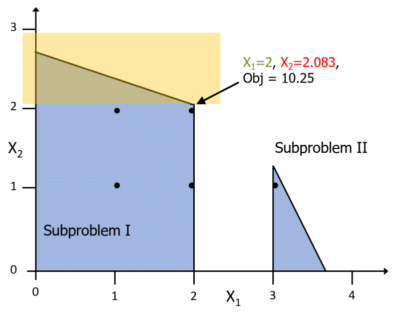
```r
LPSubI <- MIPModel() %>%
  add_variable(xx1, type = "continuous", lb = 0) %>%
  add_variable(xx2, type = "continuous",lb = 0) %>%
  set_objective(2*xx1 + 3*xx2, "max") %>%
  add_constraint(1.0*xx1 + 3.0*xx2 <= 8.25) %>% #Surfactant
  add_constraint(2.5*xx1 + 1.0*xx2  <= 8.75) %>% #Steel
  add_constraint(xx1 <= 2.0) %>% #Bound for Subproblem 1
  solve_model(with_ROI(solver = "glpk"))
  obj_val <- objective_value(LPSubI)
  x1 <- get_solution (LPSubI, 'xx1')
  x2  <- get_solution (LPSubI, 'xx2')
LPSubI_res  <- cbind(x1,x2,obj_val)
rownames(LPSubI_res) <- ""
pander(LPSubI_res,
       caption="Production Plan based on Subproblem I")
```

x1	x2	obj_val
2	2.083	10.25

Table: Production Plan based on Subproblem I

Looking over the results, we now get an integer value for x_1 but not for x_2. We repeat the same process by creating subproblems from subproblem I by branching off of x_2.





Subproblem III

Choosing which subproblem to examine next is one of the areas that large scale integer programming software and algorithms specialize and provide options. One way to think of it is to focus on searching down a branch and bound tree deeply or to search across the breadth of the tree. For this example, let's go deep (we'll return to subproblem II later.)

Since x_2 is now a non-integer solution, we will create branches with bounds (or constraints) on x_2 in the same manner as before. Subproblem III has an added constraint of $x_2 \leq 2.0$ and Subproblem IV has $x_2 \geq 3.0$.

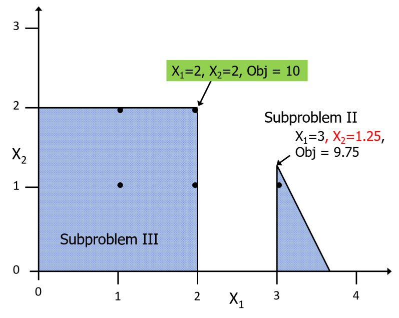
To simplify the implementation, I can simply change the upper and lower bounds on variables rather than adding separate variables.

```
LPSubIII <- MIPModel() %>%
  add_variable(xx1, type = "continuous", lb = 0, ub = 2) %>%
  add_variable(xx2, type = "continuous",lb = 0, ub = 2) %>%
  set_objective(2*xx1 + 3*xx2, "max") %>%
  add_constraint(1.0*xx1 + 3.0*xx2 <= 8.25) %>% #Surfactant
  add_constraint(2.5*xx1 + 1.0*xx2  <= 8.75) %>% #Steel
  solve_model(with_ROI(solver = "glpk"))
  obj_val <- objective_value(LPSubIII )
  x1 <- get_solution (LPSubIII , 'xx1')
  x2  <- get_solution (LPSubIII , 'xx2')
LPSubIII_res  <- cbind(x1,x2,obj_val)
rownames(LPSubIII_res) <- ""
pander(LPSubIII_res,
       caption="Production Plan based on Subproblem III")
```

Table 6.6: Production Plan based on Subproblem III

x1	x2	obj_val
2	2	10

This results in integer values for both x_1 and x_2 so it is feasible with respect to integrality in addition to the production constraints. It does generate less profit than the LP relaxation. While it is feasible, it doesn't prove that it is optimal though. We need to explore the other potential branches.



Subproblem IV

Next, let's look at Subproblem IV. This problem adds the bound of $x_2 \geq 3.0$ to subproblem I. Notice that in the MIPModel, the variable for x_1 has an upperbound (ub) of 2 in order to implement bounding constraint for subproblem I and the lower bound on x_2 of 3 in the variable declarations.

```r
LPSubIV <- MIPModel() %>%
  add_variable(xx1, type = "continuous", lb = 0, ub = 2) %>%
  add_variable(xx2, type = "continuous",lb = 3) %>%
  set_objective(2*xx1 + 3*xx2, "max") %>%
  add_constraint(1.0*xx1 + 3.0*xx2 <= 8.25) %>% #Surfactant
  add_constraint(2.5*xx1 + 1.0*xx2  <= 8.75) %>% #Steel
  solve_model(with_ROI(solver = "glpk"))
  obj_val <- objective_value(LPSubIV )
  x1 <- get_solution (LPSubIV , 'xx1')
  x2  <- get_solution (LPSubIV , 'xx2')
LPSubIV_res  <- cbind(x1,x2,obj_val)
LPSubIV$status
```

```
## [1] "infeasible"
```

```r
rownames(LPSubIV_res) <- ""
pander(LPSubIV_res,
       caption=
"Infeasible Production Plan based on Subproblem IV")
```

Table 6.7: Infeasible Production Plan based on Subproblem IV

x1	x2	obj_val
0	3	9

At first glance, this looks okay but if you look more closely at the value of x_2, and the surfactant constraint, $x_1 + 3x_2 \leq 9$, there isn't enough enough surfactant to make three units of the second product. This means that by inspection we can see that this production plan is infeasible! What is going on here?

We should check the status of the solver before examining the result.

```
LPSubIV
```

```
## Status: infeasible
## Objective value: 9
```

The returned status value of this solved LP indicates that this tell us that subproblem IV is *infeasible*. It still returned values that were used when it determined that the problem was infeasible which is why it gave the results in the previous table. From here on out, it is good to check the status. Note that it can be used as in-line evaluated expression to simply say was it feasible or not? Yes, it was infeasible.

Subproblem II

Now, we have searched down all branches or subproblems except for subproblem II. Let's go back and do that problem. We do that by simply adding one new bound to the LP Relaxation. That is $x_1 \geq 3.0$.

```
LPSubII <- MIPModel() %>%
  add_variable(xx1, type = "continuous", lb = 3) %>%
  add_variable(xx2, type = "continuous",lb = 0) %>%
  set_objective(2*xx1 + 3*xx2, "max") %>%
  add_constraint(1.0*xx1 + 3.0*xx2 <= 8.25) %>% #Surfactant
  add_constraint(2.5*xx1 + 1.0*xx2  <= 8.75) %>% #Steel
  solve_model(with_ROI(solver = "glpk"))
  obj_val <- objective_value(LPSubII )
  x1 <- get_solution (LPSubII , 'xx1')
  x2  <- get_solution (LPSubII , 'xx2')
LPSubII_res  <- cbind(x1,x2,obj_val)
cat("Status of Subproblem II:",LPSubII$status)
```

```
## Status of Subproblem II: optimal
```

```
rownames(LPSubII_res) <- ""
pander(LPSubII_res,
       caption="Production Plan based on LP Subproblem II")
```

Table 6.8: Production Plan based on LP Subproblem II

x1	x2	obj_val
3	1.25	9.75

At this point, our first reaction may be to breathe deeply and do the same branch and bound off of x_2. On the other hand, if we step back and take note that the objective function value is 9.75, while optimal for this subproblem, it is less than what we found from a feasible, fully integer-valued solution to subproblem III. Given that adding constraints can't improve an objective function value, we can cleverly trim all branches below subproblem II.

Given that we now no longer have any branches to explore, we can declare that we have found the optimal solution. The optimal solution can now be definitively stated to be what we found from Subproblem III.

```
pander(LPSubIII_res,
       caption="Acme's Optimal Production Plan
       based on Subproblem III")
```

Table 6.9: Acme's Optimal Production Plan based on Subproblem III

x1	x2	obj_val
2	2	10

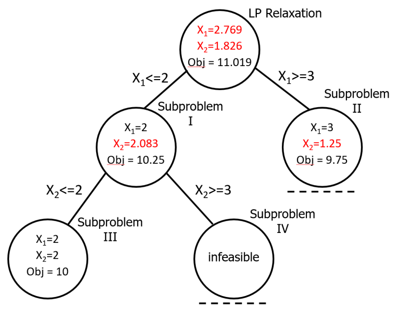
Before this, we could only say that it was feasible solution and candidate to be optimal since no better integer feasible solution had been found.

Let's summarize the results of the series of LPs solved.

```
LPSubIV_res <- c("-", "-", "Infeasible")
        # Make adjustment for misinterpretation of status
BandB <- rbind(format(round(acme_res_lp,  3), nsmall = 3),
               format(round(LPSubI_res,   3), nsmall = 3),
               format(round(LPSubIII_res,  3), nsmall = 3),
               LPSubIV_res,
               format(round(LPSubII_res,  3), nsmall = 3))
rownames(BandB) <- list("LP Relaxation", "Subproblem I",
                        "Subproblem III", "Subproblem IV",
                          "Subproblem II")
pander(BandB, caption="Branch and Bound’s Sequence of LPs")
```

Table 6.10: Branch and Bound’s Sequence of LPs

	x1	x2	Profit
LP Relaxation	2.769	1.827	11.019
Subproblem I	2.000	2.083	10.250
Subproblem III	2.000	2.000	10.000
Subproblem IV	-	-	Infeasible
Subproblem II	3.000	1.250	9.750



6.4 *Computational Complexity*

As for computational complexity, this was a small problem with only two integer variables but it still required solving the LP Relaxation and four separate LP subproblems for a total of five linear programs in all. As the number of variables and constraints increases, so do the number of LPs needed to typically find a solution using branch and bound. Small and medium size problems can generally be solved quickly but worst case scenarios have a combinatorial explosion.

Full Enumeration

Another approach to integer programming is full enumeration which means listing out all possible solutions and then determining if that solution is feasible and if so, what the objective function value is. Alas, this can result in a combinatorial explosion. For example, if a problem has 1,000 non-negative integer variables each of which can range from zero to nine, full enumeration would require listing 10^{1000} possible solutions. This is far larger than the number of grains of sand on Earth (~10^{19}) or the number of stars in the universe (~10^{19}). In fact this is much larger than the number of atoms in the universe (~10^{80}). Needless to say explicit enumeration for large problems is not an option.

The Branch and Bound algorithm has the benefit that it will find the optimal solution. Sometimes it will find it quickly, other times it may take a very long time.

Worst case scenarios for Branch and Bound may approach that of full enumeration but in practice performs much better. A variety of sophisticated algorithmic extensions have been added by researchers over the years but solving large scale integer programming problems can still be quite difficult.

One option to deal with long solving times is to set early termination conditions such as running for a certain number of seconds. If it is terminated early, it may report the best feasible solution found so far and a precise bound as to how much better an as yet unfound solution might be based on the best remaining open branch. This difference between the best feasible solution found so far and the theoretically possible best solution to be found gives a gap that an analyst can set. This acceptable gap is sometimes referred to as a suboptimality tolerance factor. In other words, what percentage of suboptimality is the analyst willing to accept for getting a solution more quickly. For example, a suboptimality tolerance factor of 10% would tell the software to terminate the branch and bound algorithm if a feasible solution is found and it is certain that regardless of how much more time is spent solving, it is impossible to improve upon this solution by more than 10%.

In practice, even small suboptimality tolerance factors like 1% can allow big problems to be solved quickly and are often well within the margin of error for model data. In other cases, organizations may be interested in finding *any* feasible solution to large, vexing problems.

Note that our Acme example only specified two digits of accuracy for resource usage and may have only one digit of accuracy for profitability per product.

Note that in our earlier example, if we had a wide enough suboptimality tolerance, say 30% and followed the branch for $x_1 \geq 3.0$ first rather than $x_1 \leq 2.0$, we might have terminated with a suboptimal solution.

6.5 Binary Variables and Logical Relations

Binary variables are a special case of general integer variables. Rather than variables taking on values such as 46 and 973, acceptable values are limited to just 0 and 1. This greatly reduces the possible search space for branch and bound since you know that in any branch, you will never branch on a single variable more than once. On the other hand, with a large number of variables, the search space can still be very large. If the previous case with 1000 variables were binary, a full enumeration list would rely on a list of 2^{1000} or one followed by about 300 zeros) possible solutions. While better than the case of integers, it is still vastly more than the number of atoms in the universe.

The important thing is is that binary variables give us a lot of rich opportunities to model complex, real world situations.

Examples include assigning people to projects, go-no go decisions on projects, and much more.

Let's explore the impact of binary restrictions with another example. The world famous semiconductor company, Outtel has a choice of five major R&D projects:

1) Develop processor architecture for autonomous vehicles (project A) 2) Next generation microprocessor architecture (project B) 3) Next generation process technology (project C) 4) New fabrication facility (project D) 5) New interconnect technology (project E)

The key data is described as follows. NPV is Net Present Value of the project factoring in costs. *Inv* is the capital expenditures or investment required for each project. The company has $40 Billion in capital available to fund a portfolio of projects. Let's set up the data as matrices in R.

```
I <- matrix(c(12,24,20,8, 16), ncol=1,
            dimnames=list(LETTERS[1:5],"Inv"))
N <- matrix(c(2.0, 3.6, 3.2, 1.6, 2.8), ncol=1,
            dimnames=list(LETTERS[1:5],"NPV"))
pander (cbind(I,N),
        caption="Data for the Outtel Example")
```

Table 6.11: Data for the Outtel Example

	Inv	NPV
A	12	2
B	24	3.6
C	20	3.2
D	8	1.6
E	16	2.8

These projects all carry their own expenses and engineering support. There are limits to both the capital investment and engineering resources available. To start, consider all of these project to be independent.

Exercise 6.1 (Formulate the model mathematically)**.**

Create a mathematical model for the basic model. To start, let's assume that partial projects are acceptable and that they can be repeated. (Hint: You don't need binary variables yet.)

Exercise 6.2 (Create and solve the model).

Formulate and solve a basic, naive model that allows project to repeated and partially completed. Implement and solve your model. What is the optimal decision? Is this result surprising? Does this make sense in terms of the application?

Exercise 6.3 (No Project is Repeated).

Now, let's explore one aspect of moving towards binary variables. What constraint or constraints are needed to ensure no project is done more than once while allowing partial projects to still be funded and incur both proportionate costs and benefits. How does this solution compare to that of above?

Exercise 6.4 (No Partial Projects).

What change is needed to prevent partial funding of projects. How does this solution compare to that of above? Can you envision a situation where it might make sense to have partially funded projects? Could this be similar to corporate joint ventures where investments and benefits are shared across multiple entities?

Now that effectively have binary constraints, let's explore the impact of logical restrictions with a series of separate scenarios. In each of these cases, make sure that you implement the relationship as a linear relationship. This means that you cannot use a function such as an "If-then", "OR", "Exclusive OR", or other relationship. Furthermore, you can't multiply variables together or divide variables into variables. In each of the following sections create a linear relationship that models this situation.

Exercise 6.5 (Chip Architects).

Let's assume that projects A and B require the focused effort of the Outtel's top chip architects and Outtel has decided therefore that it is not possible to do both projects. Therefore, they want to ensure that at most one of the two projects can be pursued. What needs to be added to enforce this situation?

Exercise 6.6 (Interconnects Need Something to Connect).

Instead of the constraint on chip architects, let's consider the situation of the interconnect technology. Assume that project E on interconnect technology would only provide strong benefit if a new architecture is also developed. In other words, E can only be done if A or B is done. Note that if both A and B are done, E will certainly have a use!

Exercise 6.7 (Manufacturing).

Outtel knows that staying aggressive in manufacturing is important in this competitive industry but that it is too risky to do too much at once from manufacturing perspective. The result is that Outtel want to ensure that exactly one major manufacturing initiative (C or D) must be done. In other words, project C or project D must be done but not both.

Exercise 6.8 (Solving Each Case).

Solve for the base case and then show the results of just each constraint at once. Combine the results into a single pander table. Discuss the results.

Exercise 6.9 (Full Enumeration?).

When projects can be neither partially funded nor repeated, how many candidate solutions would there be by full enumeration? Can you list them out?

Exercise 6.10 (Revisiting Dirk's Course Planning).

Take a look at Dirk Schumacher's vignette for "Course Planning". `https://dirkschumacher.github.io/ompr/articles/problem-course-assignment.html`

Dirk is the author of ompr and has a nice collection of vignettes.

Consider the application of senior design capstone projects where a class of 32 students is going to be divided up to work on 6 projects. Outside sponsors pitch topics and then each student of the 32 students gives a score of 0 to 10 to indicate how interested they are in the topic. Using the student project preferences, you want to assign students to the projects that they are most interested in.

You may use Dirk's vignette as a starting point to build your own model for assigning students to projects based on their preferences. Projects can have no more than 6 people on a single capstone project and every student needs to be assigned to a project.

Student preferences can be loaded from the comma separated file as shown below or typed in.

```
#studentpref<-read.csv(file="HW5data.csv")
#pander (studentpref,
#      caption="Student to Project Preferences")
```

Build and solve an appropriate model. Be sure to describe your formulation and discuss the results.

Can this problem ever be infeasible if the number of students is less than 6 times the number of projects? Why or why not?

6.6 Fixed Charge Models

Fixed charge models are a special case of integer programming models where situations where product cost has a fixed cost component and a marginal (per unit) cost component. A common example of this is when a machine must be setup before any production can occur.

Fixed Charge Example-Introduction

For example let's assume that desk, table, and bookcase production would each require fixtures for production with varying cost, say $800, $600, and $500 respectively. How would this affect your decision to produce items?

This example is based on a writeup from Thanh Thuy Nguyen in the Fall 2018 ETM 540 class.

One option is to pay for manufacturing setups of all three products. At $1900, this eliminates the potential option of choosing to make only two or three products.

Now, We could introduce a decision variable to represent the "go-no go" decision for each product. If we now solve for this case, let's examine the solution.

What we can see is that we are producing a mix of the products, we don't have any fixtures for producing these products.

What we need to do is connect the decision variables of how much of a product to make and whether we make any of that product.

We do that by connecting the two decision variables for each of the products. A common example of this kind of situation is a fixed charge problem. Let's explore an example of that.

Widget Inc. is re-evaluating their product production mix. As the plant manager, you are responsible for determining what products the company should manufacture. Since the company is leasing equipment, there are setup costs for each product. You need to determine the mix that will maximize profit for the company.

The profit for each Widget is shown below, as well as the setup costs (if you decide to produce any of that that Widget). The materials you have sourced allow you to only produce each Widget up to its capacity.

Product	Profit	Setup Cost	Capacity
Widget 1	\$15	\$1000	500
Widget 2	\$10	\$1500	1000
Widget 3	\$25	\$2000	750

To produce each Widget, the hours required at each step of the manufacturing process are shown below. Also shown are the availability (in hours) of the equipment at each step.

Production	Widget 1	Widget 2	Widget 3	Available
Pre-process	2	1	4	1000
Machining	1	5	2	2000
Assembly	2	1	1	1000
Quality Assurance	3	2	1	1500

In order to develop our optimization model, we determine the objective function (goal), decision variables (decisions) and constraints (limitations).

Objective Function:
Maximize profit

Decision Variables:

- W_1 = Number of Widget 1 to produce
- W_2 = Number of Widget 2 to produce
- W_3 = Number of Widget 3 to produce
- $Y_1 = 1$ if you choose to produce Widget 1; $Y_1 = 0$ otherwise
- $Y_2 = 1$ if you choose to produce Widget 2; $Y_2 = 0$ otherwise
- $Y_3 = 1$ if you choose to produce Widget 3; $Y_3 = 0$ otherwise

Constraints:

1) Using no more than 1000 hours of Pre-process
2) Using no more than 2000 hours of Machining
3) Using no more than 1000 hours of Assembly
4) Using no more than 1500 hours of Quality Assurance
5) Producing no more than 500 of Widget 1
6) Producing no more than 1000 of Widget 2
7) Producing no more than 750 of Widget 3

Our base model:

$$
\begin{aligned}
\text{Max } & 15W_1 + 10W_2 + 25W_3 - 1000Y_1 - 1500Y_2 - 2000Y_3 \\
\text{subject to } & 2W_1 + 1W_2 + 4W_3 \leq 1000 \\
& 1W_1 + 5W_2 + 2W_3 \leq 2000 \\
& 2W_1 + 1W_2 + 1W_3 \leq 1000 \\
& 3W_1 + 2W_2 + 1W_3 \leq 1500 \\
& W_1 \leq 500 \\
& W_2 \leq 1000 \\
& W_3 \leq 750 \\
& W_1, W_2, W_3 \geq 0 \\
& Y_1, Y_2, Y_3 \in \{0, 1\}
\end{aligned}
$$

Let's examine the results if we run this model.

For Fixed Charge problems, we need to link our Widgets to our decision on whether or not we produce the Widget (and its associated setup costs). We do this what is called the "Big M" method.

```
fc_base_model <- MIPModel() %>%
 add_variable(W1, type = "integer", lb = 0) %>% #Widget 1
 add_variable(W2, type = "integer", lb = 0) %>% #Widget 2
 add_variable(W3, type = "integer", lb = 0) %>% #Widget 3
 add_variable(Y1, type = "binary") %>%
    #Binary Decision for Widget 1
 add_variable(Y2, type = "binary") %>%
    #Binary Decision for Widget 2
 add_variable(Y3, type = "binary") %>%
    #Binary Decision for Widget 3

 set_objective(15*W1 + 10*W2 + 25*W3
              - 1000*Y1 - 1500*Y2 - 2000*Y3, "max") %>%
 add_constraint(2*W1 + 1*W2 + 4*W3 <= 1000) %>%
             # Pre-process availability
 add_constraint(1*W1 + 5*W2 + 2*W3 <= 2000) %>%
             # Machining
 add_constraint(2*W1 + 1*W2 + 1*W3 <= 1000) %>%
             # Assembly
 add_constraint(3*W1 + 2*W2 + 1*W3 <= 1500) %>%
             # Quality Assurance

 add_constraint(W1 <= 500)  %>%  # Widget 1 capacity
 add_constraint(W2 <= 1000) %>%  # Widget 2 capacity
```

```r
 add_constraint(W3 <= 750)       # Widget 3 capacity

fc_base_res <- solve_model(fc_base_model,
                           with_ROI(solver = "glpk"))
fc_base_res
```

```
## Status: optimal
## Objective value: 8160
```

Let’s now examine the results in more detail.

```r
fc_base_summary <-
  cbind(fc_base_res$objective_value,
        t(as.matrix(fc_base_res$solution)))
colnames(fc_base_summary)<-
  c("Obj Func Value", "W1","W2","W3",
    "Y1","Y2","Y3")
pander(fc_base_summary, caption =
         "Base Case Solution for Fixed Charge Problem")
```

Table 6.14: Base Case Solution for Fixed Charge Problem

Obj Func Value	W1	W2	W3	Y1	Y2	Y3
8160	266	332	34	0	0	0

We are producing all three products since W_1, W_2, and W_3 are all positive but $Y_1 = Y_2 = Y_3 = 0$. This is a nice, high profit situation because we are not paying for the production setups.

If you think about it, this may not be surprising that the optimization model chooses to not “pay” the fixed charge for production. This is a “penalty” in the objective function. What we need is a way of connecting the amount to produce of a widget and the decision to produce any of that widget.

What we need is a way to connect, associate, or dare I say “link” the W_1 and Y_1. In fact, this connection is called a *linking constraint* and is quite common in mixed integer programming problems.

6.7 Linking Constraints with “Big M”

I like to think of the linking constraint as the following: $W_1 \leq M * Y_1$. The value M is a big value that is so large that it does not prematurely limit the production for widget 1. Whatever value of M is used, W_1 can never exceed that value.

Our linking constraints force our new values to 0 or 1. If $W_1 > 0$, then this constraint forces the associated Y_1 to be equal to 1. If $W_1 = 0$, then this constraint allows Y_1 to be either 0 or 1. However, our objective function will cause Solver to avoid paying a setup by setting $Y_1 = 0$.

It might be tempting to select a very large number such as a billion 10^{10} but picking excessively large numbers, can result in poor computational performance. As we've discussed large, real-world optimization problems are hard enough, let's not make it any harder. Albert Einstein once said "A model should be as simple as possible, but no simpler." Essentially, a value for M should be as small as possible, but no smaller.

Since M serves to impose an upper bound on W_1, this might suggest how we can use this information to pick an appropriate value for M and that we can do so for each product separately, using a separate value M_i for each widget.

Let's examine the constraints for the production plan in the situation gives all resources to the production of Widget 1 and nothing to Widget 2 and Widget 3.

More formally, using the non-negative lower bounds on widget production, we can set $W_2 = W_3 = 0$ and substitute into the constraints of the optimization model. The constraints now look like the following:

$$\begin{aligned}
2W_1 + 1*0 + 4*0 &\leq 1000 \\
1W_1 + 5*0 + 2*0 &\leq 2000 \\
2W_1 + 1*0 + 1*0 &\leq 1000 \\
3W_1 + 2*0 + 1*0 &\leq 1500 \\
W_1 &\leq 500 \\
0 &\leq 1000 \\
0 &\leq 750
\end{aligned}$$

This can be further simplified as the following.

$$\begin{aligned} 2W_1 &\leq 1000 \\ 1W_1 &\leq 2000 \\ 2W_1 &\leq 1000 \\ 3W_1 &\leq 1500 \\ W_1 &\leq 500 \end{aligned}$$

It can be further simplified as the following:

$$\begin{aligned} W_1 \leq \frac{1000}{2} &= 500 \\ W_1 &\leq 2000 \\ W_1 \leq \frac{1000}{2} &= 500 \\ W_1 \leq \frac{1500}{3} &= 500 \\ W_1 &\leq 500 \end{aligned}$$

Since all constraints need to be satisfied, we can represent this to mean that Widget 1 manufacturing could never be higher than the smallest (most restrictive) of these constraints.

Therefore,

$$W_1 \leq min\{\frac{1000}{2}, \frac{2000}{1}, \frac{1000}{2}, \frac{1500}{3}, 500\}$$

or

$$W_1 \leq min\{500, 2000, 500, 500, 500\} = 500$$

Therefore, we know that Widget 1 production can never exceed 500 units regardless of the amount of W2, W3, and the objective function. We can then safely set the Big M value for product 1 to be 500. In other words, $M_1 = 500$.

We can follow the same process for setting Big M values for widget 2. We start by setting $W_2 = W_3 = 0$. We know that W_2 must be no larger than the most restrict constraint. I'll skip rewriting the constraints and jump a little ahead.

$$W_2 \leq min\{\frac{1000}{2}, \frac{2000}{5}, \frac{1000}{2}, \frac{1500}{2}, 1000\} = 400$$

Therefore, we can safely use a Big M value for W_2 of $M_2 = 400$.

Again we can follow the same process for W3 to find a small Big M value.

$$W_3 \leq min\{\frac{1000}{4}, \frac{2000}{2}, \frac{1000}{2}, \frac{1500}{1}, 750\} = 250$$

Our updated model with "Big M":

$$\begin{aligned}
\text{Max } & 15W_1 + 10W_2 + 25W_3 - 1000Y_1 - 1500Y_2 - 2000Y_3 \\
\text{subject to } & 2W_1 + 1W_2 + 4W_3 \leq 1000 \\
& 1W_1 + 5W_2 + 2W_3 \leq 2000 \\
& 2W_1 + 1W_2 + 1W_3 \leq 1000 \\
& 3W_1 + 2W_2 + 1W_3 \leq 1500 \\
& W_1 \leq 500 \\
& W_2 \leq 1000 \\
& W_3 \leq 750 \\
& W_1 - 500Y_1 \leq 0 \\
& W_2 - 400Y_2 \leq 0 \\
& W_3 - 250Y_3 \leq 0 \\
& W_1, W_2, W_3 \geq 0 \\
& Y_1, Y_2, Y_3 \in \{0, 1\}
\end{aligned}$$

6.8 Fixed Charge Implementation

Given that we have already created a model without the Big M constraints, we can simply add the constraints to the model. We'll skip the piping operator and just add

```
fc_bigM_model <- add_constraint(fc_base_model,
                                W1 - 500*Y1 <= 0) %>%
              # W1's Big M linking constraint
 add_constraint(W2 - 400*Y2 <= 0) %>%
              # W2's Big M linking constraint
 add_constraint(W3 - 250*Y3 <= 0)
              # W3's Big M linking constraint

fc_bigM_res <- solve_model(fc_bigM_model,
                           with_ROI(solver = "glpk"))
fc_bigM_res
```

```
## Status: optimal
## Objective value: 6500
```

Our model was able to find an optimal solution with an objective value of 6500. Our optimal production plan is shown below. As we can see, our model returned a production plan with only one model of widget being produced.

Interesting—the objective function value has gone down significantly.

Let's look this over in more detail and compare it to the results that we had when we avoided paying for setups.

```r
fc_M_summary <- cbind(fc_bigM_res$objective_value,
                        t(as.matrix(fc_bigM_res$solution)))
colnames(fc_M_summary)<-c("Obj Func Value", "W1","W2","W3",
                            "Y1","Y2","Y3")
fc_combined_res <- rbind (fc_base_summary, fc_M_summary)
rownames(fc_combined_res)<-c("Base Case",
                            "With Fixed Charge")
pander(fc_combined_res,
       caption = "Fixed Charge Problem")
```

Table 6.15: Fixed Charge Problem

	Obj Func Value	W1	W2	W3	Y1	Y2	Y3
Base Case	8160	266	332	34	0	0	0
With Fixed Charge	6500	500	0	0	1	0	0

Notice that the the high setup costs have caused us to focus our production planning decisions. Rather than spreading ourselves across three different product lines, we are producing as many of widget 1 as we can.

6.9 *Model Results and Interpretation*

We can also calculate our resource usage.

```r
fc_bigM_res.W1 <- get_solution(fc_bigM_res, W1)
        # Extract solution value for decision variable, W1
fc_bigM_res.W2 <- get_solution(fc_bigM_res, W2)
        # Extract solution value for decision variable, W2
fc_bigM_res.W3 <- get_solution(fc_bigM_res, W3)
        # Extract solution value for decision variable, W3
fc_bigM_res.df <- data.frame(c(fc_bigM_res.W1,
                              fc_bigM_res.W2,
                              fc_bigM_res.W3))
fc_bigM_res.r1 <- t(data.frame(c(fc_bigM_res.W1*2,
                                  fc_bigM_res.W2*1,
                                  fc_bigM_res.W3*4)))
                #multiply results with hours used
rownames(fc_bigM_res.r1) <- "Pre-process"
fc_bigM_res.r2 <- t(data.frame(c(fc_bigM_res.W1*1,
                                  fc_bigM_res.W2*5,
                                  fc_bigM_res.W3*2)))
                #multiply results with hours used
rownames(fc_bigM_res.r2) <- "Machining"
fc_bigM_res.r3 <- t(data.frame(c(fc_bigM_res.W1*2,
                                  fc_bigM_res.W2*1,
                                  fc_bigM_res.W3*1)))
                #multiply results with hours used
rownames(fc_bigM_res.r3) <- "Assembly"
fc_bigM_res.r4 <- t(data.frame(c(fc_bigM_res.W1*3,
                                  fc_bigM_res.W2*2,
                                  fc_bigM_res.W3*1)))
                #multiply results with hours used
rownames(fc_bigM_res.r4) <- "Quality Assurance"
fc_bigM_res.tot <- data.frame(c(sum(fc_bigM_res.r1),
                                 sum(fc_bigM_res.r2),
                                 sum(fc_bigM_res.r3),
                                 sum(fc_bigM_res.r4)))
                #sum each step
colnames(fc_bigM_res.tot) <- "Total Used"
fc_bigM_res.avail <- data.frame(
  c("1000","2000","1000","1500"))
                #print available hours for each step
colnames(fc_bigM_res.avail) <- "Available"
pander(cbind(rbind(fc_bigM_res.r1,fc_bigM_res.r2,
                   fc_bigM_res.r3,fc_bigM_res.r4),
                   fc_bigM_res.tot,
                   fc_bigM_res.avail),
       caption = "Manufacturing Resource Usage")
```

Table 6.16: Manufacturing Resource Usage

	W1	W2	W3	Total Used	Available
Pre-process	1000	0	0	1000	1000
Machining	500	0	0	500	2000
Assembly	1000	0	0	1000	1000

	W1	W2	W3	Total Used	Available
Quality Assurance	1500	0	0	1500	1500

As can be seen, we used all of our Pre-processing hours, Assembly hours, and Quality Assurance hours. There is a significant about of time available in Machining though.

Further analysis could examine alternatives such as redesigning widget 2 and 3 to be less resource instensive in production to see at what point we would choose to produce them.

```
##
## Attaching package: 'magrittr'

## The following object is masked from 'package:tidyr':
##
##     extract
```

7
More Integer Programming Models

7.1 Overview

This chapter consists of a collection of rich MILP models that can be used for inspiration on products. Some of these cases are expansions of Dirk Schumacher's ompr vignettes. These make for excellent resouces demonstrating a variety of features such as creation of simulated data and visualization of results. I strongly recommend reading the original. These vignettes can be viewed from the package documentation, Dirk's github web site, or downloading his github repository.

- All text from Dirk Schumacher's articles are set in block quotes.
- All code chunks are from Dirk Schumacher's articles unless stated otherwise.
- The LaTeX formulations are based on Dirk Schumacher's LaTeX with some modifications to support LaTeX environments.

7.2 Revisiting the Warehouse Location Problem

Let's start by reviewing Dirk's original description of the warehouse location problem.

> In this article we will look at the Warehouse Location Problem. Given a set of customers and set of locations to build warehouses, the task is to decide where to build warehouses and from what warehouses goods should be shipped to which customer.

> Thus there are two decisions that need to made at once: where and if to build warehouses and the assignment of customers to warehouses. This simple setting also implies that at least one warehouse must be built and that any warehouse is big enough to serve all customers.

As a practical example: you run the logistics for an NGO and want to regularly distribute goods to people in need. You identified a set of possible locations to set up your distribution hubs, but you are not sure where to build them. Then such a model might help. In practice however you might need to incorporate additional constraints into the model.

Let's start by defining the decision variables. Each possible location of a warehouse, j can have a warehouse be built or not be built. We will use $y_j = 1$ to indicate that warehouse j is built. Conversely, $y_j = 0$ indicates that we are not building a warehouse at location j. Since a warehouse is either built or not built, a binary variable is appropriate for y_j.

Similarly, the variable $x_{i,j}$ is the decision of assigning customer i to warehouse j. It also needs to be a binary variable since partial assignments are not allowed. Therefore, $x_{3,7} = 1$ means that customer *3* is assigned to warehouse j and we would expect $x_{3,8} = 0$ since a customer can't be assigned to two warehouses.

Now, that we have a handle on the variables, let's move on to the constraints. Each customer must be assigned to one and only one warehouse. For illustration, this means that one of the variables $x_{1,j}$ must be set to one and the others are zero. To enforce this constraint, we can simply add the x variables for warehouse 1 for each of the warehouses. We could do this with $x_{1,1}+x_{1,2}+\ldots+x_{1,m}$ and requiring it to be one. We could rewrite this using a summation as $\sum\limits_{j=1}^{m} x_{1,j} = 1$. That constraint is limited to just customer 1 though. We don't want to write this constraint out separately for each customer so we can generalize this by repeating it for all n customers as $\sum\limits_{j=1}^{m} x_{i,j} = 1,\ i = 1, \ldots, n$.

It would not work to assign a customer to an unbuilt warehouse. For example, it would not work to have customer 23 assigned to warehouse 7 if warehouse 7 was not built. In other words, the combination of $x_{23,7} = 1$ and $y_7 = 0$ would be a big problem and should not happen. We need to connect the decision variable $x_{23,7}$ and y_7. One way to do that is creating a constraint, $x_{23,7} \leq y_7$ which explicitly blocks assigning customer 23 to warehouse 7 unless warehouse 7 is operating. This can be generalized as $x_{i,j} \leq y_j,\ i = 1, \ldots, n,\ j = 1, \ldots, m$.

Our objective is to minimize cost. We have a cost assumed for each customer to warehouse assigment. This might be related to distance. Let's refer to this cost as $\text{transportcost}_{i,j}$. The cost based on the transportation from warehouse to customer is then $\sum\limits_{i=1}^{n}\sum\limits_{j=1}^{m} \text{transportcost}_{i,j} \cdot x_{i,j}$. Note that we could concisely abbreviate it as something like $C^T_{i,j}$. In this case, using a capital C to indicate cost and a superscript T to indicate transportation cost.

We also have a cost factor for each warehouse that we choose to build/operate. Again, if we define fixedcost_j as the fixed cost for building warehouse j, the cost of our decisions is simply, $\sum\limits_{j=1}^{m} \text{fixedcost}_j \cdot y_j$.

Let's combine all this together into the formulation.

$$\begin{array}{lll}
\min & \sum\limits_{i=1}^{n}\sum\limits_{j=1}^{m} \text{transportcost}_{i,j} \cdot x_{i,j} + \sum\limits_{j=1}^{m} \text{fixedcost}_j \cdot y_j & \\
\text{subject to} & \sum\limits_{j=1}^{m} x_{i,j} = 1 & i = 1, \ldots, n \\
& x_{i,j} \leq y_j, & i = 1, \ldots, n, j = 1, \ldots, m \\
& x_{i,j} \in \{0,1\} & i = 1, \ldots, n, j = 1, \ldots, m \\
& y_j \in \{0,1\} & j = 1, \ldots, m
\end{array}$$

Implementing the Model

Rather than typing in fixed data or loading it from data file, Dirk simply generated a collection of random data for testing purposes. This highlights the advantage of R in that we have a rich collection of numerical tools available that we can leverage.

The first thing we need is the data. In this article we will simply generate artificial data.

We assume the **customers** are located in a grid with Euclidian distances. Let's explain the functions used for this data generation. The first command sets the random number seed. In general, computers don't actually generate random numbers, they generate what is called pseudo random numbers according to a particular algorithm in a sequence. The starting point will set a sequence that appears to be random and behaves in a relatively random way. Much more can be said but this is setting as the first random number, 1234. It could just as easily have been 4321 or any other integer.

Let's unpack this further.

The grid size is set to 1000. You can think of this as a map for a square region that is 1000 kilometers in both horizontal (East-West) and vertical (North-South) directions for visualization purposes. We are then randomly placing customers on the map.

Next we set the number of customers to be 100 and the number of warehouses to be 20. The size of this problem can have a tremendous impact on solving speed.[1]

[1] In particular, I recommend using a much smaller problem size (fewer customers and warehouses) if you are using a slow computer or a shared resource such as a server or cloud-based service like RStudio.cloud. Integer programming problems can quickly consume tremendous amounts of computer processing time which can have budgetary impacts or other problems.

```
set.seed(1234)      # Set random number seed
grid_size <- 1000   # Horizontal and vertical axis ranges
n <- 100 ; m <- 20 # Set number of customers and warehouses
        # Select 100 and 20 for local computer runs
#n <- 10 ; m <- 3  # Safer problem size for cloud services
        # Text discussion is for the larger size
```

Next we create the customer data as a dataframe. The runif function generates a uniform random number between 0.0 and 1.0 such as perhaps 0.327216. This would be multiplied by our grid size multipler resulting in 327.216. This is then rounded to 327.

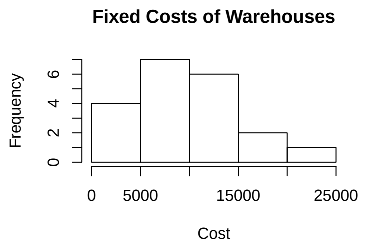
```
customer_locations <- data.frame(
  id = 1:n,
     # Create column with ID numbers from 1 to 100
  x = round(runif(n) * grid_size),
     # Create 100 x-coordinate values in next column
  y = round(runif(n) * grid_size)
     # Create 100 y-coordinate values in last column
)
```

The **warehouses** are also randomly placed on the grid. The fixed cost for the warehouses are randomly generated as well with mean cost of 10,000. Notice that in this case, the fixedcost is generated using a normal random variable function, with a mean of 10 times the grid size or 10,000 and a standard deviation of 5000.

```
warehouse_locations <- data.frame(
  id = 1:m,
  x = round(runif(m) * grid_size),
  y = round(runif(m) * grid_size)
)
fixedcost <- round(rnorm(m, mean = grid_size * 10,
                         sd = grid_size * 5))
```

The fixed costs to set up a warehouse are the following:

```r
hist(fixedcost, main="Fixed Costs of Warehouses",
     xlab="Cost")
```



Note that the distribution of fixed costs is pretty broad. In fact, using a normal distribution means that is unbounded in both directions so it is possible for a warehouse to have a negative construction cost. While a negative construction cost might seem impossible, perhaps this would be an example of local tax incentives at work.

For transportation cost, we will assume that simple Euclidean distance is a good measure of cost. Just a reminder, Euclidean distance between two points is $\sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2}$. Of course in terms of cost, we could scale it or add a startup cost to each trip but for the sake of this example, we don't need to worry about it.

Her we define a function to calculate the distance between customer i and warehouse j.

```r
transportcost <- function(i, j) {
  customer <- customer_locations[i, ]
  warehouse <- warehouse_locations[j, ]
  round(sqrt((customer$x - warehouse$x)^2 +
               (customer$y - warehouse$y)^2))
}
transportcost(1, 3)
```

```
## [1] 302
```

We could look at a sample of the customer location data.

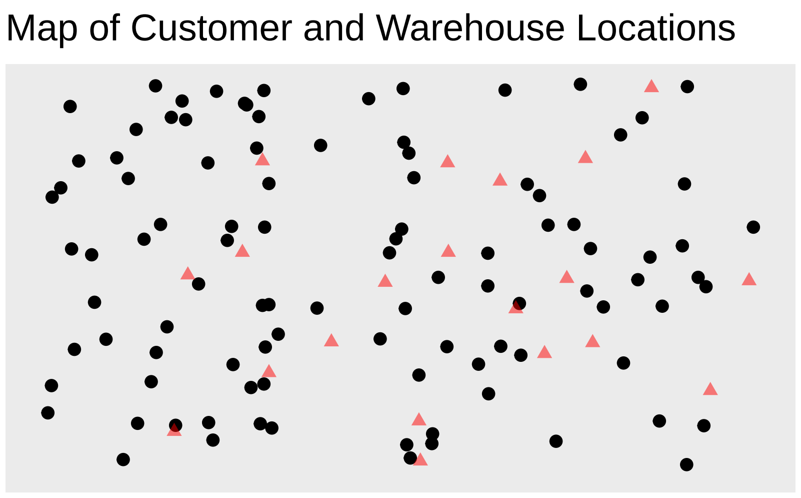
```r
pander(head(customer_locations),
       caption="Locations of First Six Customers")
```

Table 7.1: Locations of First Six Customers

id	x	y
1	114	35
2	622	565
3	609	280
4	623	204
5	861	134
6	640	326

A table of the randomly generated locations is not terribly helpful for understanding the locations of customers or warehouses though. Let's instead build a map by plotting both customers and warehouses together. Black dots are customers and red dots are possible warehouse locations.

```
p <- ggplot(customer_locations, aes(x, y)) +
  geom_point() +
  geom_point(data = warehouse_locations, color = "red",
             alpha = 0.5, shape = 17) +
  scale_x_continuous(limits = c(0, grid_size)) +
  scale_y_continuous(limits = c(0, grid_size)) +
  theme(axis.title = element_blank(),
        axis.ticks = element_blank(),
        axis.text = element_blank(),
        panel.grid = element_blank())
p + ggtitle("Map of Customer and Warehouse Locations")
```



Black dots are customers. Light red triangles show potential warehouse locations.

Note that I have modified the variables names for x and y to be `Vx` and `Vy` in the code chunk to remove confusion over x and y representing location or variables. The prefix of `V` is meant to suggest that it is a variable.

Note that I have modified the variables names for x and y to be `Vx` and `Vy` in the code chunk to remove confusion over x and y representing location or variables. The prefix of `V` is meant to suggest that it is a variable.

```
model <- MIPModel() %>%
  # 1 iff i gets assigned to warehouse j
  add_variable(Vx[i, j], i=1:n, j=1:m, type="binary") %>%
  # 1 iff warehouse j is built
  add_variable(Vy[j], j = 1:m, type = "binary") %>%

  # maximize the preferences
  set_objective(sum_expr(transportcost(i, j) * Vx[i, j],
                         i = 1:n, j = 1:m) +
                  sum_expr(fixedcost[j] * Vy[j], j = 1:m),
                "min") %>%

  # every customer needs to be assigned to a warehouse
  add_constraint(sum_expr(Vx[i, j], j=1:m)==1, i=1:n) %>%

  # if a customer is assigned to a warehouse,
  #    then this warehouse must be built
  add_constraint(Vx[i,j] <= Vy[j], i = 1:n, j = 1:m)
model
```

```
## Mixed integer linear optimization problem
## Variables:
##   Continuous: 0
##   Integer: 0
##   Binary: 2020
## Model sense: minimize
## Constraints: 2100
```

The number of x (or `Vx`) variables is $m \cdot n = 20 \cdot 100$ or 2000 and the number of y (or `Vy`) variables is m or 20, the total number of variables is 2020 which matches the model summary. A similar calculation can be done on the constraints. The number of variables and constraints make this a non-trivial sized MIP problem. Fortunately, solving still turns out to be pretty easy on a personal computer but be cautious of solving this on a cloud instance.

Solving the Warehouse Location Problem

We are now ready to work on solving the model. We will start with using `glpk`.

The solver can generate a lot of additional information. The default is `verbose = FALSE` but let's see the extra information that we get with a verbose solution.

So far, you are unlikely to have noticed an optimization problem cause a visible delay. If you run the following code chunk to solve this model, you may see the first distinct pause. As I said, this is a non-trivial MIP problem! It may take only a few seconds or perhaps a few minutes depending upon your computer. This also means that you must be careful in running MIP problems on cloud based services. A particularly big MIP running on Amazon Web Services might create a sizeable bill. Running it on a shared service like RStudio.cloud's service, currently being provided for free while in Alpha should not be used for such computationally intense purposes.

In summary, non-trivial MIP models should be run locally on a computer unless you are certain that your computational loads will not cause a problem for yourself or others.

```
result <- solve_model(model, with_ROI(solver = "glpk",
                                       verbose = TRUE))
```

```
## <SOLVER MSG>  ----
## GLPK Simplex Optimizer, v4.47
## 2100 rows, 2020 columns, 6000 non-zeros
##       0: obj =  0.000000000e+000  infeas = 1.000e+002 (100)
## *   101: obj =  5.223300000e+004  infeas = 0.000e+000 (100)
## *   500: obj =  5.223300000e+004  infeas = 0.000e+000 (0)
## *  1000: obj =  3.413000000e+004  infeas = 0.000e+000 (0)
## *  1075: obj =  3.322200000e+004  infeas = 0.000e+000 (0)
## OPTIMAL SOLUTION FOUND
## GLPK Integer Optimizer, v4.47
## 2100 rows, 2020 columns, 6000 non-zeros
## 2020 integer variables, all of which are binary
## Integer optimization begins...
## +  1075: mip =     not found yet >=              -inf        (1; 0)
## +  1075: >>>>>  3.322200000e+004 >=  3.322200000e+004   0.0% (1; 0)
## +  1075: mip =  3.322200000e+004 >=     tree is empty   0.0% (0; 1)
## INTEGER OPTIMAL SOLUTION FOUND
## <!SOLVER MSG> ----
```

We can summarize the information by simply extracting a the objective function value. This can be done with inline R code. For example, we can say the following.

We solved the problem with an objective value of 33222.

With 2020 variables, it isn't always terribly useful to just list out all of the variables.

At this point, we are really interested in the small minority of non-zero variables. Only 5% of the x variables are not zero.

Challenge: Do you know why about 5% of the decision variables are non-zero?)

We can do some processing to extract the non-zero x (or `Vx`) variables from those with zero values. In order to do so, Dirk uses the `dplyr` package. This package provides a tremendous number of functions for data management. To illustrate how this is done, let's review line by line how this code chunk works.

```
matching <- result %>%
  get_solution(Vx[i,j]) %>%
  filter(value > .9) %>%
  select(i, j)
```

The next command `matching <- result %>%` does a few things. It uses the piping operator `%>%` to pass the `result` object into being used by the following command and then everything that gets done later in this piped sequence will be assigned to `matching`.

The command `get_solution(Vx[i,j]) %>%` extracts solution values for the x variables (`Vx`) from the solved model object `result` piped in from the previous stage. This will give us a data object consisting of 2000 variables values, 95% of them being zero.

The command `filter(value > .9) %>%` takes the previous command's 2000 variable values and only keeps the ones for which the value is larger than 0.9. Since these are binary variables, they should all be exactly 1.0 but just in case there is any numerical anomaly where a value is passed as a value close to one, such as 0.9999999, it is best to test with some tolerance. This command then only passes to the next command the 100 non-zero values.

The last command in this code chunk `select(i, j)` says to only select (or retain) the columns named `i` and `j`. Notice that this does not pass the actual value of zero or one - we know it is one! Also, note that it does not include the piping operator `%>%` which means that it is the end of this piped sequence.

Let's review what the two versions of the results look like.

```
pander(head( get_solution(result, Vx[i,j])),
       caption = "Raw results for Vx Variable.")
```

Table 7.2: Raw results for Vx Variable.

variable	i	j	value
Vx	1	1	0
Vx	2	1	0

variable	i	j	value
Vx	3	1	0
Vx	4	1	0
Vx	5	1	0
Vx	6	1	0

Notice that most of the results are simply zero indicating that the first six customers are not assigned to warehouse 1. It is common in many binary programming cases for the vast majority of variables to have a value of zero.

Notice that in the table of raw results, the first six entries (obtained using the `head` function) did not show any customers assigned to a warehouse. This isn't surprising given that only one in twenty values are non-zero.

```
pander(head( matching),
       caption=
   "Subset of Customers (i) Assigned to Warehouses (j)")
```

Table 7.3: Subset of Customers (i) Assigned to Warehouses (j)

i	j
13	7
17	7
20	7
21	7
22	7
23	7

Example of some of the processed results for variable x from matching.

The processed results table shows that the `matching` object simply lists six customers and to which warehouse it is assigned. It has cut out the 95% of rows that would be zero.

This clean and simple listing from `matching` is interesting and will then be used for adding lines to the earlier `ggplot` model.

The last step is to add the assignments to the previous plot we generated. Dirk used one single code chunk to do more processing of results, focused on warehouses, and then create the final plot. I'll break up the steps into separate code chunks just for the sake of illustrating how they work and showing the intermediate products. This is a useful chance to see how to manage results.

```
plot_assignment <- matching %>%
  inner_join(customer_locations, by = c("i" = "id")) %>%
  inner_join(warehouse_locations, by = c("j" = "id"))
```

```r
pander (head(plot_assignment),
        caption=
      "XY coordinates to draw Customer-Warehouse Routes")
```

Table 7.4: XY coordinates to draw Customer-Warehouse Routes

i	j	x.x	y.x	x.y	y.y
13	7	283	950	308	802
17	7	286	946	308	802
20	7	232	797	308	802
21	7	317	744	308	802
22	7	303	916	308	802
23	7	159	995	308	802

Notice that this gives the beginning and ending coordinates of what will soon be lines to show the connections between customers and warehouses.

Now, let's calcuate how many customers each warehouse serves.

```r
customer_count <- matching %>%
  # Process matching and save result to customer_count
  group_by(j) %>%          # Group by warehouse
  summarise(n = n()) %>%  # Summarize count
  rename(id = j)          # Rename column from j to id
pander (customer_count, caption=
  "Count of customers assigned to each operating warehouse")
```

Table 7.5: Count of customers assigned to each operating warehouse

id	n
7	32
8	29
15	20
18	19

We can see that warehouse 17 has the most customers but that the two warehouses are pretty evenly balanced.

```
plot_warehouses <- warehouse_locations %>%
  mutate(costs = fixedcost) %>%
  inner_join(customer_count, by = "id") %>%
  filter(id %in% unique(matching$j))
pander (plot_warehouses, caption =
          "Table of information about warehouses used")
```

Table 7.6: Table of information about warehouses used

id	x	y	costs	n
7	308	802	6843	32
8	404	337	2434	29
15	567	567	1749	20
18	701	307	259	19

The `plot_warehouses` data frame[2] is built using `dplyr` functions to create a summary of the important information about the warehouses used: where they are located, costs, and number of customers served.

[2] Note that the data type can sometimes be a source of problems for R users. You can use the command `class(plot_warehouses)` to confirm that it is a `data.frame`.

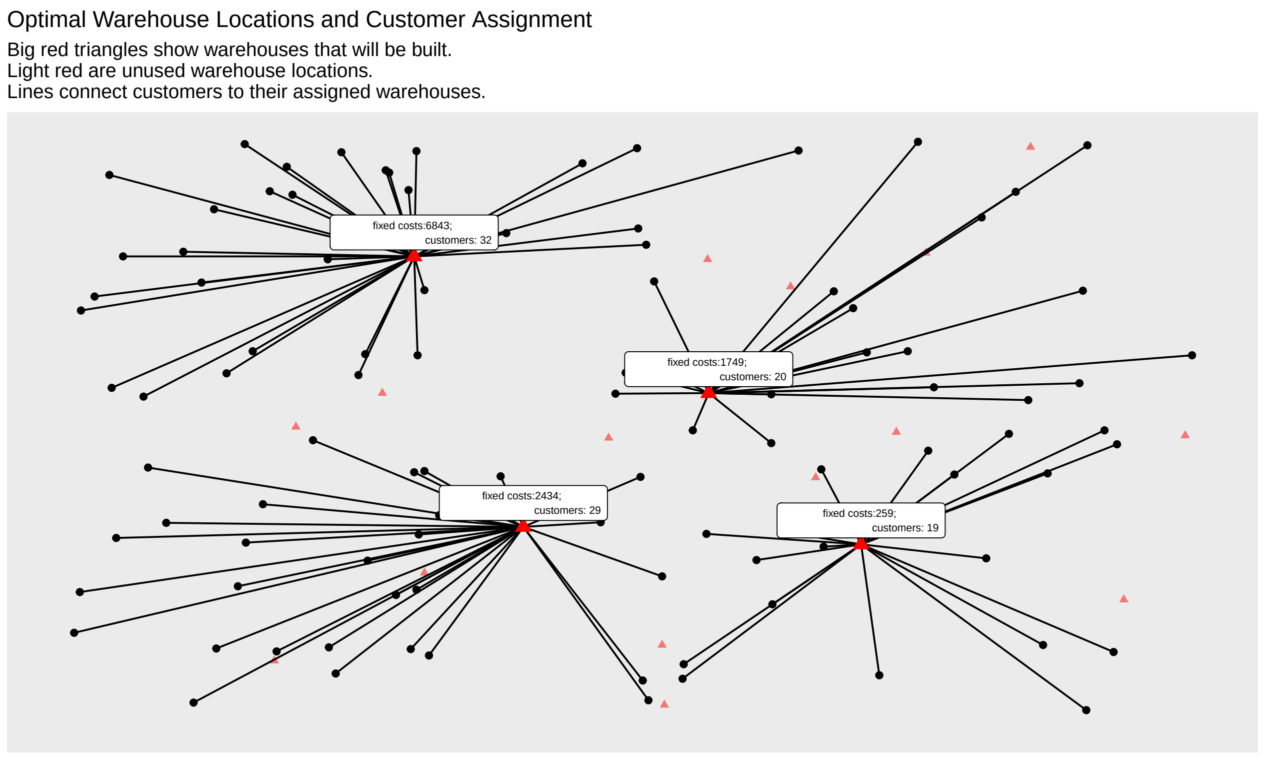
```
p +
  geom_segment(data=plot_assignment,
             aes(x=x.y, y=y.y, xend=x.x, yend=y.x)) +
  geom_point(data  = plot_warehouses,
            color = "red", size=3, shape=17) +
  ggrepel::geom_label_repel(data  = plot_warehouses,
                           aes(label = paste0(
                             "fixed costs:",costs, ";
                             customers: ", n)),
                           size = 2, nudge_y = 20) +
  ggtitle(paste0(
  "Optimal Warehouse Locations and Customer Assignment"),
"Big red triangles show warehouses that will be built.
Light red are unused warehouse locations.
Lines connect customers to their assigned warehouses.")
```

The total fixed costs for setting up the 4 warehouses is:

```
sum(fixedcost[unique(matching$j)])
```

```
## [1] 11285
```

The above function cleverly uses the warehouse ID column from `matching` (`matching$j`), makes the list unique to get rid of duplicates.



Recall that warehouses will be listed 100 times since every customer has a warehouse listed in matching. Next, it uses these ID numbers to extract fixedcost values and adds them up. Note that when you see a command such as this in R, it can often work to run them one at time in the console to see how they work. To demonstrate this, see how each statement builds upon the previous.

```
matching$j
```

```
##   [1]  7  7  7  7  7  7  7  7  7  7  7  7  7
##  [14]  7  7  7  7  7  7  7  7  7  7  7  7  7
##  [27]  7  7  7  7  7  7  8  8  8  8  8  8  8
##  [40]  8  8  8  8  8  8  8  8  8  8  8  8  8
##  [53]  8  8  8  8  8  8  8  8  8 15 15 15 15
##  [66] 15 15 15 15 15 15 15 15 15 15 15 15 15
##  [79] 15 15 15 18 18 18 18 18 18 18 18 18 18
##  [92] 18 18 18 18 18 18 18 18 18
```

```
   # List the warehouses used by customers
unique(matching$j)
```

```
## [1]  7  8 15 18
```

```
    # Eliminate duplicate warehouse IDs
fixedcost[unique(matching$j)]
```

```
## [1] 6843 2434 1749  259
```

```
    #Find fixed costs of warehouses used
sum(fixedcost[unique(matching$j)])
```

```
## [1] 11285
```

```
    # Add fixed costs of all warehouses used
```

Discussion

This warehouse customer assignment problem was discussed in detail for a variety of reasons.

- It demonstrates a large, classic, and common industrial application of optimization.
- The application uses simulated data to feed into an optimization model.
- This model uses both single and double subscripted variables as well techniques for coordinating them.
- Introduces the use of `ggplot` to visualize the data and results.
- Demonstrates data munging which is a major time sink for analysts which can also occur in large optimizaton models so this application shows how to use `dplyr` to process data and results.

This analysis could be extended in a variety of ways to explore other questions.

- What is the statistical distribution of the optimal number of warehouses used when simulated a 1000 times?
- How often would it be optimal to five or more warehouses?
- Does decreasing variation in fixed costs affect the number of warehouses used?
- Does decreasing the mean cost of warehouses affect the number of warehouses used?

The model can also be extended in a variety of ways.

- To allow for additional application characteristics such as warehouse capacity and customer demand.
- Using actual cities in a region for locations rather than randomly generated data and a map.

The same concepts can also be applied to other applications.

7.3 Solving MIPs with Different Solvers

Up until this point, we have used the `glpk` solver. One of the big benefits of `glpk` though is that it lets separate the process of formulating the optimization model from that of solving it and thereby let's us easily switch to other solvers. Now that we have a nontrivial optimization model, the Warehouse Location Model, let's use it.

The emphasis in this section is not model building but comparing the use of different solvers. Since we have the model already defined as an object, simply named `model`, we can easily pass it to the different solvers for optimizing. There is a large collection of other solvers that can be used with R.[3]

Let's use the `tictoc` package for examining computation time.

[3] This topic is not covered in Dirk's vignette or blog post.

We'll demonstrate the performance for illustration purposes. Again, do not run the following code chunks in this section on a cloud-based service such as Rstudio.cloud, unless you are certain that imposing a non-trivial computational load is acceptable. *Performance of* ***`glpk`***

We have been using `glpk` for all of our earlier examples. For the sake of comparison, let's start off with it too. We will use the `tictoc` package to help us collect data on timing. Note that timing can vary even if the same problem is solved on the same computer twice in rapid succession.

```r
library(tictoc)    # Package used for timing R functions

tic("glpk")         # Start the timer...

result_glpk <- solve_model(
  model, with_ROI(solver = "glpk", verbose=TRUE))
```

```
## <SOLVER MSG>  ----
## GLPK Simplex Optimizer, v4.47
## 2100 rows, 2020 columns, 6000 non-zeros
##       0: obj =  0.000000000e+000  infeas = 1.000e+002 (100)
## *   101: obj =  5.223300000e+004  infeas = 0.000e+000 (100)
## *   500: obj =  5.223300000e+004  infeas = 0.000e+000 (0)
## *  1000: obj =  3.413000000e+004  infeas = 0.000e+000 (0)
## *  1075: obj =  3.322200000e+004  infeas = 0.000e+000 (0)
## OPTIMAL SOLUTION FOUND
## GLPK Integer Optimizer, v4.47
## 2100 rows, 2020 columns, 6000 non-zeros
## 2020 integer variables, all of which are binary
## Integer optimization begins...
## +  1075: mip =     not found yet >=              -inf        (1; 0)
## +  1075: >>>>>  3.322200000e+004 >=  3.322200000e+004   0.0% (1; 0)
## +  1075: mip =  3.322200000e+004 >=     tree is empty   0.0% (0; 1)
## INTEGER OPTIMAL SOLUTION FOUND
## <!SOLVER MSG> ----
```

```
glpktime <- toc()  # End the timer and save results
```

```
## glpk: 10.12 sec elapsed
```

```
print(solver_status(result_glpk))
```

```
## [1] "optimal"
```

```
glpktime1 <- c("glpk", glpktime$toc - glpktime$tic,
               result_glpk$status,
               result_glpk$objective_value)
```

The `glpk` solver worked. We'll take the results and combine them with other shortly *Performance of* ***symphony***

Let's move on to testing the `symphony` solver.

```
tic("symphony")

result_symphony <-  solve_model(
  model, with_ROI(solver = "symphony", verbosity = -1))

symphonytime <- toc()
```

```
## symphony: 10.41 sec elapsed
```

```
print(solver_status(result_symphony))
```

```
## [1] "optimal"
```

```
symphonytime1 <- c("symphony",
                   symphonytime$toc - symphonytime$tic,
                   result_symphony$status,
                   result_symphony$objective_value)
```

Again, `symphony` successfully solved the optimization problem.

Several items should be noted from the above code and output. One item is that parameters can be passed directly to solvers. This is why `verbose = TRUE` was used for `glpk` but `symphony` uses `verbosity = -1`. Differing levels of verbosity gives much more granularity than a simple TRUE/FALSE. Setting `verbosity = 0` will give rich detail that an analyst trying to improve solution speed may find useful or for debugging why a model did not work but explaining the report is beyond the scope of this text.

It should be noted that other options can be passed to the symphony solver though such as `time_limit`, `gap_limit`, and `first_feasible`.

The `time_limit` option has a default value of -1 meaning no time limit. It should be an integer to indicate the number of seconds to run before stopping.

The `node_limit` option has a default value of -1 meaning no limit on the number of nodes (in an MIP problem, the number of linear programs). It should be an integer to indicate the number of nodes examined before stopping.

The last option, `first_feasible`, has a default value of `FALSE`. If it is set to `TRUE`, then symphony will stop when it has found a first solution that satisfies all the constraints (including integrality) rather than continuing on to prove optimality.

Let's examine the impact of just looking at the first feasible solution found by passing `first_feasible=TRUE` to see what happens.

```r
tic("symphony")

result_symphony_ff <- solve_model(
  model, with_ROI(solver = "symphony",
                  verbosity = -1, first_feasible=TRUE))

symphonytimeff <- toc()
```

```
## symphony: 10.19 sec elapsed
```

```r
print(solver_status(result_symphony_ff))
```

```
## [1] "infeasible"
```

```r
symphonytimeff <- c("symphony first feas.",
                    symphonytimeff$toc - symphonytimeff$tic,
                   result_symphony_ff$status,
                   result_symphony_ff$objective_value)
```

Several interesing and important things should be noted here. First, `ompr` interprets the status message from the solver as not being solved to optimality to indicate that the problem is not feasible. This is a known issue in `ompr` and `ompr.ROI`as of version 0.8.0.0. It highlights that "infeasible" status from the `ompr` should be thought of as meaning that an optimal solution was not found for *some* reason such as being told to terminate after the first feasible solution was found, time limit reached, node limit reached, the MIP was infeasible, the MIP was unbounded, or some other issue.

A second thing to note that in my randomly generated instance, the very first feasible solution it found, happens to have the same objective function value as the optimal solution found to optimality earlier by `glpk` and `symphony`. This is similar to the very simple, two variable integer programming problem that we examined using a branch and bound tree in the previous chapter. Symphony just doesn't have confirmation yet that this first feasible solution is truly optimal yet.

Thirdly, the time on my computer sometimes took less time to solve to optimality than for when it stopped *early* with just the initial feasible solution. This demonstrates the variability of solving time. Also, it might be that the code is more tightly optimized for cases of fully solving the LP than for early termination.

Performance of `lpsolve`

The `lpsolve` package has a long history too and is widely used. Let's test it and see how it performs.

```r
tic("lpsolve")

result_lpsolve <-  solve_model(
  model, with_ROI(solver = "lpsolve", verbose=TRUE))
```

```
## <SOLVER MSG>  ----
##
## Model name:  '' - run #1
## Objective:   Minimize(R0)
##
## SUBMITTED
## Model size:     2100 constraints,    2020 variables,         6000 non-zeros.
## Sets:                                   0 GUB,                   0 SOS.
##
## Using DUAL simplex for phase 1 and PRIMAL simplex for phase 2.
## The primal and dual simplex pricing strategy set to 'Devex'.
##
## Optimized BLAS was successfully loaded for bfp_LUSOL.
##
## Relaxed solution               33222 after        699 iter is B&B base.
##
```

```
## Warning in solve.lpExtPtr(om): printing of
## extremely long output is truncated
```

```
## Warning in solve.lpExtPtr(om): printing of
## extremely long output is truncated
```

```
##
## Excellent numeric accuracy ||*|| = 5.77316e-015
##
##  MEMO: lp_solve version 5.5.2.0 for 64 bit OS, with 64 bit LPSREAL variables.
```

```
## Warning in solve.lpExtPtr(om): printing of
## extremely long output is truncated
```

```
##      There were 2 refactorizations, 0 triggered by time and 0 by density.
##       ... on average 328.0 major pivots per refactorization.
##      The largest [LUSOL v2.2.1.0] fact(B) had 3473 NZ entries, 1.0x largest basis.
##      The maximum B&B level was 1, 0.0x MIP order, 1 at the optimal solution.
##      The constraint matrix inf-norm is 1, with a dynamic range of 1.
##      Time to load data was 0.376 seconds, presolve used 0.003 seconds,
##       ... 0.062 seconds in simplex solver, in total 0.441 seconds.
## <!SOLVER MSG> ----
```

```r
lpsolvetime <- toc()
```

```
## lpsolve: 10.75 sec elapsed
```

```r
print(solver_status(result_lpsolve))
```

```
## [1] "optimal"
```

```r
lpsolvetime1 <- c("lpsolve",
                  lpsolvetime$toc - lpsolvetime$tic,
                   result_lpsolve$status,
                   result_lpsolve$objective_value)
```

We can see that `lpsolve` was also successful.

7.4 Comparing Results across Solvers

Now let's compare how they each did.

```
timeresults <- rbind(glpktime1, symphonytime1,
                     symphonytimeff, lpsolvetime1)

colnames (timeresults) <- c("Solver", "Time (Sec)",
                            "Status",
                            "Obj. Func. Value")

rownames (timeresults) <- NULL

panderOptions('round', 2)

pander(timeresults,
    caption="Comparison of Results from Warehouse
       Customer Assignment Across Solvers", digits=2)
```

Table 7.7: Comparison of Results from Warehouse Customer Assignment Across Solvers

Solver	Time (Sec)	Status	Obj. Func. Value
glpk	10.12	optimal	33222
symphony	10.49	optimal	33222
symphony first feas.	10.19	infeasible	33222
lpsolve	11.88	optimal	33222

The most important thing to note is that each solver states that it found an optimal solution with the same objective function value. While they may vary in how it is achieved if there are multiple optima (in other words, different values for decision variables x and y), this means that they all found a correct solution. It would be a big issue if they differed. Causes for difference could be timing out, satisfying a suboptimality tolerance factor condition, or some error.

Note that time will vary significantly from computer to computer and may also depend upon the other tasks and the amount of memory available. Performance will change for different solver settings and for different problems. Also, performance time will change for different random instances—simply changing the random number seed will create diferent run-time results.

The ROI package lists 19 different solvers with ROI plugins. Three commercial solvers, C-Plex, GUROBI, and MOSEK are worth highlighting. These programs may be substantially faster on large MIP problems and may make versions available for academic use. Others are for quadratic optimization or special classes of optimization problems.

After running these chunks dozens of times and finding the results to be quite stable across the various solvers tested, I decided to make use of the code chunk option `cache=TRUE`. Caching means that it will save the results unless things are changed. If it determines that things have changed, it will rerun that chunk. This can save a lot of time in knitting the chapter or rebuilding the book. This is an excellent application area for demonstrating the benefits of using the chunk option for caching. The current application requires 10-15 seconds to do each full size customer warehouse optimization run. Caching just the four runs can save about a minute of knitting time. RStudio and knitr do a sophisticated check for changes which can invalidate the previously cached results and will then rerun the code chunk and save the results for future cached knits as appropriate. More demanding needs can use the granularity of numerical options for caching as compared to simply TRUE/FALSE and additional options. These might be required in detailed performance testing or numerical performance comparisons.

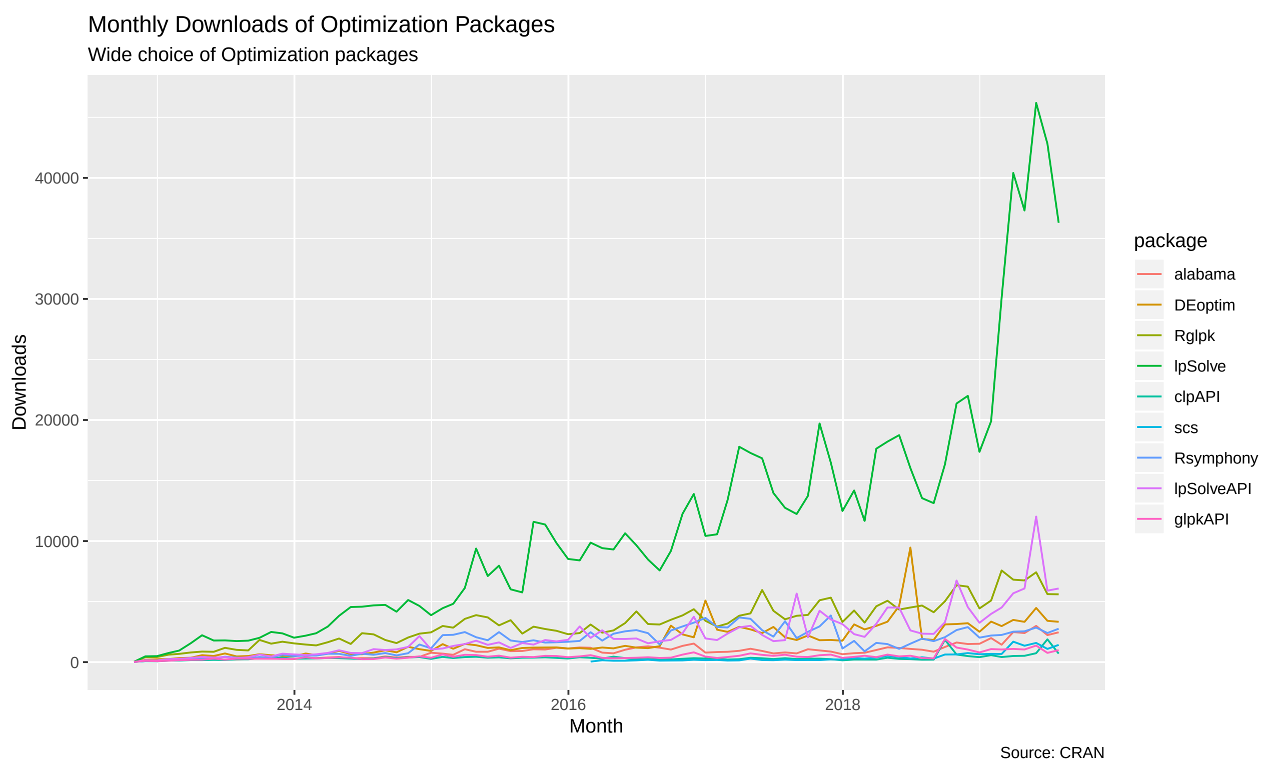
Significant computer time can be used for large models but a variety of questions can be examined such as the following.

- How is problem size related to solving time?
- Are there systematic differences in speed or solving success for different solvers? (ex. does Gurobi run faster?)
- How do various modeling parameters affect solution speed?

7.5 Popularity of LP Solvers

Let's take another look at comparing solvers by examining their popularity. More precisely, let's examine the download logs of CRAN for a variety of linear programming solvers.

Many of these packages are being used directly in other packages. While we can't see which packages are being specifically used most



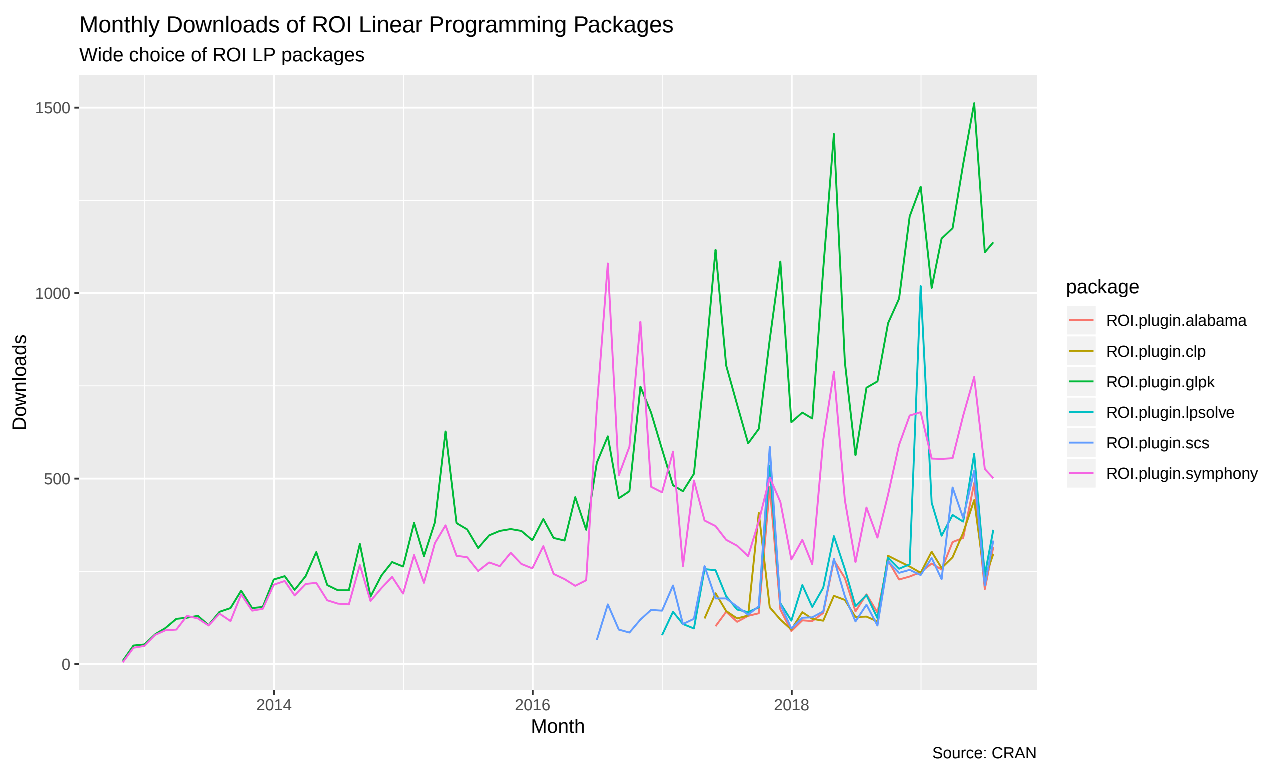
often for `ompr`, we can see which ROI plugins are downloaded most often which is probably a good proxy.

We will focus our attention on the ROI packages that emphasize linear programming. This means dropping packages such as `ROI.plugin.nloptr`, `ROI.plugin.qpOASES`, and `ROI.plugin.quadprog`. Also, proprietary packages that are not available on CRAN such as GUROBI and XPress, are not included.

It is interesting to see that the symphony and glpk ROI plugins were relatively equal in downloads until about 2015 when glpk started to open a lead in usage. This of course does not mean that glpk is better than the others but is interesting to note. The spikes in downloads could be driven by other packages that use this particular solver, visibility from vignettes, courses, other publications, or package updates.

Let's close this with examining the popularity of `ompr` itself.

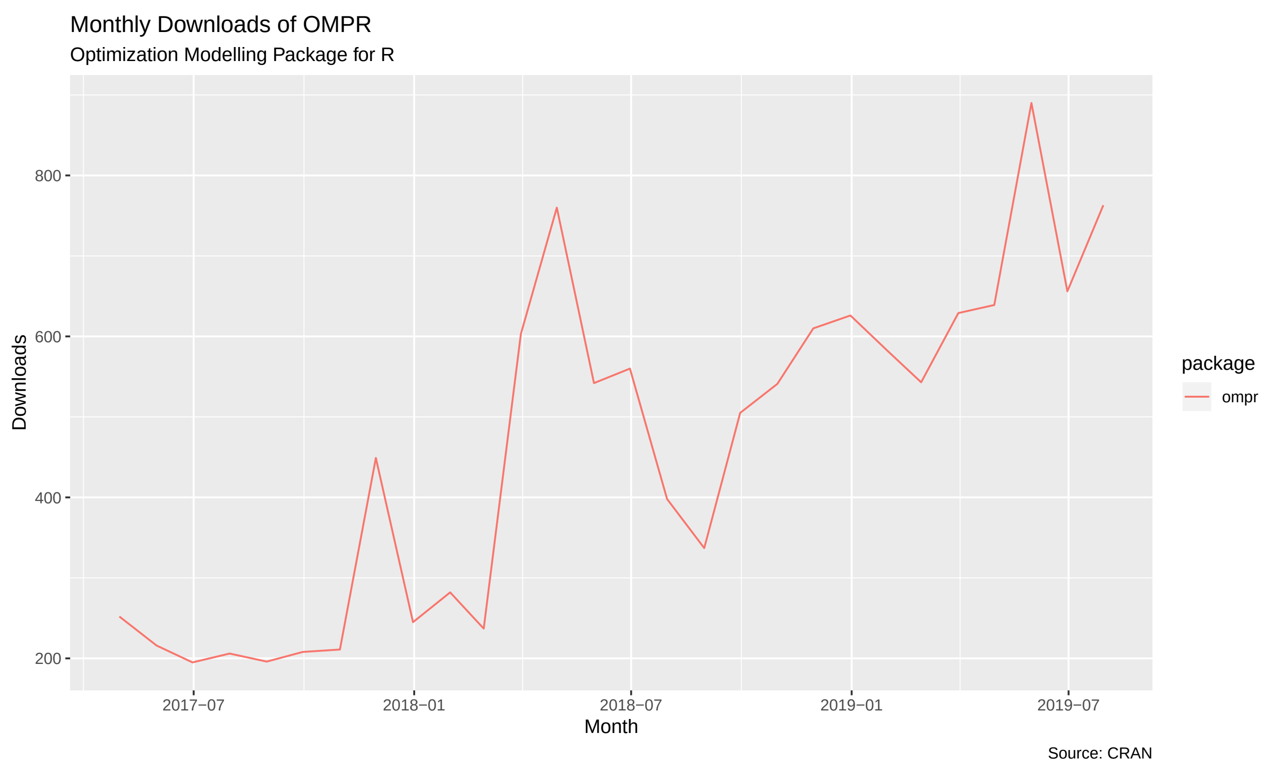
```r
pkg_dl_data <- cran_stats(packages = c("ompr"))

ggplot(pkg_dl_data, aes(end, downloads, group=package,
```



```
                                   color=package)) +
    geom_line()  +
    labs(title = "Monthly Downloads of OMPR",
       subtitle = "Optimization Modelling Package for R",
       caption = "Source: CRAN",
       x = "Month", y = "Downloads")
```

7.6 Solving Sudoku Puzzles using Optimization

Would you like to play a game? — Joshua (Wargames-1983)



				9				
							4	
			6		5			
						9		
	9	8	5					
			1	6			7	8
1		5	9		2			
			3					
9					4		3	

After spending a significant amount of time and CPU cycles on a serious topic of warehouse optimization, let's take a break with a more lighthearted application of the classic Sudoku puzzle game. Feel free to work on this Sudoku puzzle. For those unfamiliar with Sudkoku, the core idea is to use every digit from 1-9 once in each column, once in each row, and once in each 3x3 cluster.

The optimization approach will give you the opportunity to solve this or any other Sudoku model. This section is based on an article by Dirk Schumacher and helper functions from the 'sudokuAlt' package from Bill Venables, one of the Godfathers of R. Let's start with Dirk's explanation, with permission, of the model.

> In this vignette we will solve Sudoku puzzles using MILP. Sudoku in its most popular form is a constraint satisfaction problem and by setting the objective function to 0 you transform the optimization problem into a pure constraint satistication problem. In this document we will consider Sudokus in a 9x9 grid with 3x3 sub-matrices.
>
> Of course you can formulate an objective function as well that directs the solver towards solutions maximizing a certain linear function.
>
> The idea is to introduce a binary variable x with three indexes i, j, k

that is 1 if and only if the number k is in cell i, j.

The basic optimization formulation is interesting for a few reasons.

First, there isn't a clear objective function. We are simply trying to find a a set of values that solves the problem without violating any of the rules of Sudoku. In other words, any feasible solution would be acceptable. We could minimize $x_{1,1,1}$, the sum of all decision variables (which would be 81, corresponding to the number of cells in the Sudoku grid.) We could also just tell it to maximize a particular number, say 42, or minimize another number like 0, which is exactly what Dirk did.

We need a constraint to ensure that each cell (i, j) contains a value.

$$\sum_{k=1}^{9} x_{i,j,k} = 1,\ \forall i, j$$

We then need to ensure that in each row, i, each digit k only appears once.

$$\sum_{i=1}^{n} x_{i,j,k} = 1,\ \forall j, k$$

We then need to ensure that in each column, j, each digit only appears once.

$$\sum_{j=1}^{n} x_{i,j,k} = 1,\ \forall i, k$$

Next, we need to create a constraint for each 3x3 grouping so that it contains each digit only once. This is more complicated. Let's consider the top left 3x3 cluster. For each digit k, we need to ensure that only one cell has it.

$$x_{1,1,k} + x_{1,2,k} + x_{1,3,k} + x_{2,1,k} + x_{2,2,k} + x_{2,3,k} + x_{3,1,k} + x_{3,2,k} + x_{3,3,k} = 1,\ \forall\ k$$

Let's generalize this by using summations.

$$\sum_{i=1}^{3}\sum_{j=1}^{3} x_{i,j,k} = 1,\ k = 1, \ldots, 9$$

Unfortunately this set of constraints only covers one cluster.

If we are clever, we may note that the top left corner of each cluster is a row or column value of 1, 4, or 7.

We could extend this constraint to handle each set of clusters. We'll use a counter across for each cluster by row, using r. Similarly, we'll use c for cluster.

$$\sum_{i=1+3r}^{3+3r}\ \sum_{j=1+3c}^{3+3c} x_{i,j,k} = 1,\ k = 1, \ldots, 9\ r = 0, 1, 2,\ c = 0, 1, 2,$$

We can now pull everything together into a single formulation. Again, remember that we could use *any* objective function since we are simply trying to find a feasible solution. We will just picking maximizing the number zero since ompr defaults to maximization as an objective function. We could also tell a solver such as `symphony` to just find a first feasible solution and terminate.

$$\begin{aligned}
\max\ \ & 0 \\
\text{subject to}\ \ & \sum_{k=1}^{9} x_{i,j,k} = 1 & i = 1, \ldots, n,\ j = 1, \ldots, n \\
& \sum_{i=1}^{n} x_{i,j,k} = 1 & j = 1, \ldots, n,\ k = 1, \ldots, 9 \\
& \sum_{j=1}^{n} x_{i,j,k} = 1 & i = 1, \ldots, n,\ k = 1, \ldots, 9 \\
& \sum_{i=1+r}^{3+r} \sum_{j=1+c}^{3+c} x_{i,j,k} = 1, & k = 1, \ldots, 9\ r = 0, 3, 6,\ c = 0, 3, 6, \\
& x_{i,j,k} \in \{0, 1\} & i = 1, \ldots, n,\ j = 1, \ldots, n,\ k = 1, \ldots, 9
\end{aligned}$$

7.7 *Implementing the Sudoku model in ompr*

We are now ready to implement the model. As always, clearly defining variables and constraints is important. A triple subscripted variable often makes for a tricky model. Also, that last constraint may take careful examination.

```r
n <- 9
Sudoku_model <- MIPModel() %>%

  # The number k stored in position i,j
  add_variable(Vx[i, j, k], i = 1:n, j = 1:n, k = 1:9,
               type = "binary") %>%

  # no objective
  set_objective(0) %>%

  # only one number can be assigned per cell
  add_constraint(sum_expr(Vx[i, j, k], k = 1:9) == 1,
                 i = 1:n, j = 1:n) %>%

  # each number is exactly once in a row
  add_constraint(sum_expr(Vx[i, j, k], j = 1:n) == 1,
                 i = 1:n, k = 1:9) %>%
  # each number is exactly once in a column
  add_constraint(sum_expr(Vx[i, j, k], i = 1:n) == 1,
               j = 1:n, k = 1:9) %>%

  # each 3x3 square must have all numbers
  add_constraint(sum_expr(Vx[i, j, k], i = 1:3 + r,
                          j = 1:3 + c) == 1,
               r = seq(0, n - 3, 3),
               c = seq(0, n - 3, 3), k = 1:9)
Sudoku_model
```

```
## Mixed integer linear optimization problem
## Variables:
##   Continuous: 0
##   Integer: 0
##   Binary: 729
## Model sense: maximize
## Constraints: 324
```

We will use `glpk` to solve the above model. Note that we haven't fixed any numbers to specific values. That means that the solver will find a valid sudoku without any prior hints.

I've made a couple of minor changes to Dirk's code chunks. I replaced the `x` variable with `Vx` to avoid R name space collisions with previously defined enviromental variables for `x`. Secondly, I switch the `sx` and `sy` to `r` and `c` to represent moving over by rows and columns.

```
Sudoku_result <- solve_model(
  Sudoku_model, with_ROI(
    solver = "glpk", verbose = TRUE))
```

```
## <SOLVER MSG>  ----
## GLPK Simplex Optimizer, v4.47
## 324 rows, 729 columns, 2916 non-zeros
##       0: obj =  0.000000000e+000  infeas = 3.240e+002 (324)
##     500: obj =  0.000000000e+000  infeas = 2.000e+001 (79)
## *   520: obj =  0.000000000e+000  infeas = 4.644e-014 (75)
## OPTIMAL SOLUTION FOUND
## GLPK Integer Optimizer, v4.47
## 324 rows, 729 columns, 2916 non-zeros
## 729 integer variables, all of which are binary
## Integer optimization begins...
## +   520: mip =     not found yet <=              +inf        (1; 0)
## +  1414: >>>>>  0.000000000e+000 <=  0.000000000e+000   0.0% (43; 0)
## +  1414: mip =  0.000000000e+000 <=     tree is empty   0.0% (0; 85)
## INTEGER OPTIMAL SOLUTION FOUND
## <!SOLVER MSG> ----
```

```
# the following dplyr statement plots a 9x9 matrix
Solution <- Sudoku_result %>%
  get_solution(Vx[i,j,k]) %>%
  filter(value > 0) %>%
  select(i, j, k) %>%
  tidyr::spread(j, k) %>%
  select(-i)
```

If you want to solve a specific sudoku you can fix certain cells to specific values. For example here we solve a sudoku that has the sequence from 1 to 9 in the first 3x3 matrix fixed.

```
Sudoku_model_specific <- Sudoku_model %>%
  add_constraint(Vx[1, 1, 1] == 1) %>%
     # Set digit in top left to the number 1
  add_constraint(Vx[1, 2, 2] == 1) %>%
     # Set digit in row 1, column 2 to the number 2
  add_constraint(Vx[1, 3, 3] == 1) %>%
     # Set digit in row 1, column 3 to the number 3
  add_constraint(Vx[2, 1, 4] == 1) %>%
     # Set digit in row 2, column 1 to the number 4
  add_constraint(Vx[2, 2, 5] == 1) %>%  # etc....
  add_constraint(Vx[2, 3, 6] == 1) %>%
  add_constraint(Vx[3, 1, 7] == 1) %>%
  add_constraint(Vx[3, 2, 8] == 1) %>%
  add_constraint(Vx[3, 3, 9] == 1)
Sudoku_result_specific <- solve_model(
  Sudoku_model_specific, with_ROI(
    solver = "glpk", verbose = TRUE))
```

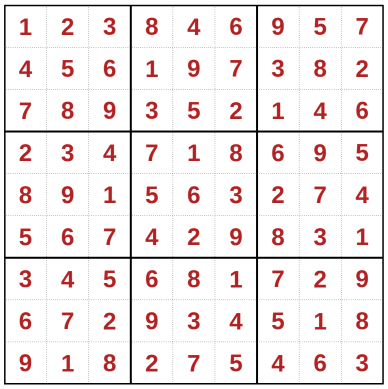
```
## <SOLVER MSG>  ----
## GLPK Simplex Optimizer, v4.47
## 333 rows, 729 columns, 2925 non-zeros
##       0: obj =  0.000000000e+000  infeas = 3.330e+002 (333)
## *   477: obj =  0.000000000e+000  infeas = 2.015e-015 (84)
## OPTIMAL SOLUTION FOUND
## GLPK Integer Optimizer, v4.47
## 333 rows, 729 columns, 2925 non-zeros
## 729 integer variables, all of which are binary
## Integer optimization begins...
## +   477: mip =     not found yet <=              +inf        (1; 0)
## +   980: >>>>>  0.000000000e+000 <=  0.000000000e+000   0.0% (31; 0)
## +   980: mip =  0.000000000e+000 <=     tree is empty   0.0% (0; 61)
## INTEGER OPTIMAL SOLUTION FOUND
## <!SOLVER MSG> ----
```

```
Solution_specific <- Sudoku_result_specific %>%
  get_solution(Vx[i,j,k]) %>%
  filter(value > 0) %>%
  select(i, j, k) %>%
  tidyr::spread(j, k) %>%
  select(-i)
Solution_specific   # Display solution to Sudoku Puzzle
```

```
##   1 2 3 4 5 6 7 8 9
## 1 1 2 3 8 4 6 9 5 7
## 2 4 5 6 1 9 7 3 8 2
## 3 7 8 9 3 5 2 1 4 6
## 4 2 3 4 7 1 8 6 9 5
## 5 8 9 1 5 6 3 2 7 4
## 6 5 6 7 4 2 9 8 3 1
## 7 3 4 5 6 8 1 7 2 9
## 8 6 7 2 9 3 4 5 1 8
## 9 9 1 8 2 7 5 4 6 3
```

Now, let's try printing it nicely using the sudokuAlt package from Bill Venables, available from CRAN.

```
plot(as.sudoku(as.matrix(Solution_specific, colGame="grey")))
```



1	2	3	8	4	6	9	5	7
4	5	6	1	9	7	3	8	2
7	8	9	3	5	2	1	4	6
2	3	4	7	1	8	6	9	5
8	9	1	5	6	3	2	7	4
5	6	7	4	2	9	8	3	1
3	4	5	6	8	1	7	2	9
6	7	2	9	3	4	5	1	8
9	1	8	2	7	5	4	6	3

Have at it. Any Sudoku puzzle can be solved using this model as long as there is a feasible solution. Note that if there are two or more solutions to the puzzle based on the hints, this model will find one of them and does not indicate whether there might be other solutions.

Feel free to time the solution using the tictoc package.

The intention here is not to imply that optimization is the only way or the best way to solve Sudoku problems. There are algorithmic approaches for solving Sudoku that can be more computationally efficient than using a general purpose integer programming system and in fact people have implemented such algorithms in R and other languages. The purpose of this example was to show how a nontraditional problem can be framed and implemented.

7.8 Conclusion

In this chapter, we covered some problems with medium-sized integer programming problems. The same approaches and models can be scaled up to larger problems.

Improvements in software and computer hardware have opened up optimization to a variety of large scale problems. Integrating this into R has made it straightforward to connect together a variety of tools including simulation and data visualization.

7.9 Exercises

Even moderate sized integer programming problems can result in significant computational loads. While these problems will not represent a heavy load for a desktop or laptop computer, they be intensive for netbooks or cloud-based services. If you are using a cloud-based service such as RStudio.cloud, a server, or third party service such as Amazon Web Services, be sure to understand the implications of heavy computational loads. Possible results of excessive use of third party computer time include throttling of access similar to cell phone users exceeding monthly data caps, high "overage" charges, a warning from a systems administrator, or even account termination.

Exercise 7.1 (Warehouse-Max Customers-Small)**.**

Run the analysis for the smaller warehouse optimization but with just 10 customers and 3 warehouses. Extend the warehouse customer assignment problem to have each warehouse that is built be assigned a maximum of 4 customers. Compare the solution to that obtained earlier. Ensure that your data for customers and warehouses is the same for the two cases.

This problem size should be cloud-friendly but ensure that you are solving the approppropriate sized problem before running.

Exercise 7.2 (Warehouse-Max and Min Customers-Small)**.**

Run the analysis for the smaller warehouse optimization but with just 10 customers and 3 warehouses. Extend and solve the warehouse customer assignment problem to have each warehouse that is built be assigned a maximum of 4 customers and a minimum of 2 customers. Show plots of both solutions. Compare the solution to that obtained earlier using tables as appropriate. Ensure that your data for customers and warehouses is the same for the two cases.

This problem size should be cloud-friendly but ensure that you are solving the appropropriate sized problem before running.

Exercise 7.3 (Warehouse-Simulating Customers-Moderate)**.**

Run the analysis for the smaller warehouse optimization with a max of 10 customers and 3 warehouses. Simulate it 20 times and plot the results in terms of number of warehouses used and the total cost. Discuss the results.

This problem size should not be be run on a cloud or remote server without full understanding of load implications. Note that this will be somewhat computationally intensive and is best done on a local computer rather than the cloud. Doing it on a personal computer may require on the order of five minutes of processing time.

Exercise 7.4 (Warehouse-Max Customers-Big)**.**

Extend the warehouse customer assignment problem to have each warehouse that is built be assigned a maximum of 40 customers. Compare the solution to that obtained earlier. Ensure that your data for customers and warehouses is the same for the two cases.

This problem size should not be be run on a cloud or remote server without full understanding of load implications.

Exercise 7.5 (Warehouse-Max and Min Customers-Big)**.**

Extend the warehouse customer assignment problem to have each warehouse that is built be assigned a maximum of 40 customers and a minimum of 15 customers. Compare the solution to that obtained earlier. Ensure that your data for customers and warehouses is the same for the two cases.

This problem size should not be be run on a cloud or remote server without full understanding of load implications.

Exercise 7.6 (Warehouse-Simulating Customers - Big)**.**

Run the analysis for the warehouse optimization model 20 times and plot the results in terms of number of warehouses used and the total cost. Discuss and interpret the results.

Note that this will be **computationally intensive** and should not be done on the cloud. Doing it on a personal computer may require on the order of five minutes of processing time.

Exercise 7.7 (Sudoku-Bottom Row)**.**

Solve the Sudoku puzzle from the beginning of this section using R.

Exercise 7.8 (Sudoku-Bottom Row)**.**

Solve a Sudoku model using optimization where the bottom row is 9 numbered down to 1.

Exercise 7.9 (Sudoku-Right Column)**.**

Solve a Sudoku model using optimization where the right most column is 9 numbered down to 1.

Exercise 7.10 (Sudoku-Center Cluster)**.**

Solve a Sudoku model using optimization where the center 3x3 cluster is made up of the numbers from 9 to 1. (First row is 9, 8, 7; second row is 6, 5, 4; bottom row is 3, 2, 1)

Exercise 7.11 (Find a Sudoku to Solve)**.**

Find a Sudoku puzzle from a newspaper or elsewhere and solve it using optimization.

				9				
							4	
			6		5			
						9		
	9	8	5					
			1	6			7	8
1		5	9		2			
			3					
9					4		3	

Exercise 7.12 (Solve a Specific Sudoku)**.**

Solve the Sudoku problem from the beginning of this section.

Exercise 7.13 (Sudoku-Multiple Optima).

Challenge: Sudoku puzzles are often created with the intention of having a single solution since the approaches people use to solve them are based on incrementally reasoning out what values each cell must contain. The result is that the ambiguity of multiple optima may cause problems for people to solve. While the optimization model can solve these without difficulty, it is only going to find one, arbitrary solution. Simply resolving is likely to give you the same solution even when there are multiple optima. Consider variations and apply them to find multiple solutions to a problem. For an example of one with multiple optimal solutions, the most extreme case would be starting with a blank board.

Exercise 7.14 (Sudoku-Infeasible)**.**

Challenge: A Sudoku puzzle with only a small numbers shown can be infeasible even though it does not contain any obvious violation of repeated digits. People building Sudoku puzzles need to ensure that every puzzle is solveable or risk having frustrated customers. Add a digit one at a time to an existing Sudoku puzzle board that does not immediately violate the repeated digits requirement, test for a valid solution, and repeat, until the Sudoku game board becomes infeasible.

Exercise 7.15 (Sudoku-4x4).

Ultra-Challenge: Solver the 16 letter Sudoku puzzle using R. It consists of 4x4 grids and uses the letters starting with A in place of numbers.

G									E				L		
			K				I							C	O
		E								C	O		B	J	
		F	M	K			J	L	B	D				A	P
		D		L											
		K	N						A	L			G		E
						J	N		K						
	I							N		O		L			
K												J			
								J					K		
		J					K						D		
P		N	I	J			L								
				F		K		C		N					
C							M								
	D			A	C					K		N			M
		P		E		D			G						

8 Goal Programming and Multiple Objectives

Up until this point, we assumed that there would be a single, clear objective function. Often we have more complex situations where there are multiple conflicting objectives. In our earlier production planning case, we might have additional objectives besides maximizing profit such as minimizing environmental waste or longer term strategic positioning. In the case of our capital budgeting problem, we can envision a range of additional considerations beyond simple unexpected net present value maximization.

Let's look at another type of problem, staff scheduling. The Information Technology Security Institute or ITSI needs to maintain 24x7 staffing to respond to crises in addition to regular workloads. The result is that there is a varying demand over time. For the sake of simplicity, let's treat the schedule as a series of 21 eight hour shifts.

ITSI has a variety of staff with different skill sets, pay rates, preferences, and scheduling requests. Jane is the Director of ITSI and wants to find a good schedule of workers.

```r
Req <- matrix(c(10,35,30,
                15,30,30),ncol=1,dimnames=list(LETTERS[1:6],"x"))
```

Idea, extend Dirk's class schedule assignment example to include secondary objectives.

MORE TO BE ADDED

A
An Introduction to Summation Notation

A.1 Basic Summation Notation

In linear programming, we often need to sum sets of numbers, such as:
1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 + 11 + 12 + 13 + 14 + 15
+ 16 + 17 + 18 + 19 + 20 + 21 + 22 + 23 + 2 + 25 + 26 + 27 + 28
+ 29 + 30 + 31 + 32 + 33 + 34 + 35 + 36 + 37 + 38 + 39 + 40 +
41 + 42 + 43 + 44 + 45 + 46 + 47 + 48 + 49 + 50

Summation notation is a mathematical shorthand used to represent repeated addition of algebraic expressions involving one or more variables. The Greek capital letter sigma, $\sum$, is the symbol used to show that we wish to make calculations using summation notation. Since summation notation includes at least one variable–let's limit this initial examination to one variable only–the variable is displayed below the $\sum$ symbol. The summation notation below tells us that we will be finding the summation of a variable expression involving x.

$$\sum_{x}$$

The starting value of the variable may be given below $\sum$ as well. In this case, the summation notation below tells us that the initial value of x will be equal to 1.

$$\sum_{x=1}$$

In some cases, we may also wish to designate an ending value of the variable, which we can include above the $\sum$. The summation notation below also tells us that the final value of x will be equal to 5.

$$\sum_{x=1}^{5}$$

In all dealings with summation notation, variables will only take integer values, beginning and ending at any values provided within the summation notation. Note that some summations may use an "ending" value of ∞, which would involve the summation of an infinite number of values.

Let's look at a basic summation problem.

$$\sum_{x=1}^{5} 2x$$

The summation above means that we will take the x values, starting at $x = 1$, and multiply the value of x by 2. We will continue to do this for each integer value of x until we have reached our ending value of 5. Then we will sum all of our results (five of them, in this case) to produce one final value for the summation.

$$\sum_{x=1}^{5} 2x$$

$$\begin{aligned} &= 2*1+2*2+2*3+2*4+2*5 \\ &= 2+4+6+8+10 \\ &= 30 \end{aligned}$$

Summation can be calculated over a variety of algebraic expressions. Another, perhaps more challenging, example is shown below.

$$\sum_{x=0}^{3} x^2 - 4x + 1$$

$$\begin{aligned} &= (0^2 - 4*0+1) + (1^2 - 4*1+1) + (2^2 - 4*2+1) + (3^2 - 4*3+1) \\ &= 1 + (-2) + (-3) + (-2) \\ &= -6 \end{aligned}$$

A.2 Notation in LaTeX

LaTex is a common typesetting used to express mathematical notation. The summation notation symbols thus far in this text have been written using LaTex. Inline LaTeX commands can be added by including a $ symbol on either side of the LaTeX command. To create entire LaTeX code chunks, include two $$ symbols before and after the chunk of LaTeX.

A.3 Sums

To display the $\sum$ symbol in LaTex, use the command `\sum_{lower}^{upper}`.

$$\sum_{lower}^{upper}$$

The sum expression can be added using the command:

$$\sum_{t=0}^{n} \frac{CF_t}{(1+r)^t}$$

Sum limits can be written to appear above and below the operator.

A.4 Products

The multiplication or product over a sequence is denoted as the uppercase Greek letter pi, $\prod$. The product command can be added using the expression `\prod_{x = a}^{b} f(x)`

$$\prod_{x=a}^{b} f(x)$$

The lower and upper limits of summation after the `\sum` are both optional. Products and integrals work similarly, only with `\prod` and `\int`:

Delimiters, like parentheses or braces, can automatically re-size to match what they are surrounding. This is done by using `\left` and `\right`:

$$\left(\sum_{i=1}^{n} i\right)^2 = \left(\frac{n(n-1)}{2}\right)^2 = \frac{n^2(n-1)^2}{4}$$

A.5 Summary of Mathematical Notations

Below are some common mathematical functions that are often used.

1. x = y $1.x = y$

2. x < y

$$2.x < y$$

3. x > y

$$3.x > y$$

4. x `\le` y

$$4.x \leq y$$

5. x `\ge` y

$$5.x \geq y$$

6. `x^{n}`

$$6.x^n$$

7. `x_{n}`

$$7.x_n$$

8. `overline{x}`

$$8.\overline{x}$$

9. `hat{x}`

$$9.\hat{x}$$

10. `tilde{x}`

$$10.\tilde{x}$$

11. `frac{a}{b}`

$$11.\frac{a}{b}$$

12. `binom{n}{k}`

$$12.\binom{n}{k}$$

13. `x_{1} + x_{2} + \cdots + x_{n}`

$$13.x_1 + x_2 + \cdots + x_n$$

14. `|A|`

$$14.|A|$$

15. `x \in A`

$$15.x \in A$$

16. `x \subset B`

$$16.x \subset B$$

17. `x \subseteq B`

$$17. x \subseteq B$$

18. `A \cup B`

$$18. A \cup B$$

19. `{1, 2, 3\}`

$$19. \{1, 2, 3\}$$

20. `\sin(x)`

$$20. \sin(x)$$

21. `\log(x)`

$$21. \log(x)$$

22. `\int_{a}^{b}`

$$22. \int_{a}^{b}$$

23. `\left(\int_{a}^{b} f(x) \; dx\right)`

$$23. \left(\int_{a}^{b} f(x) \; dx\right)$$

24. `\left. F(x) \right|_{a}^{b}`

$$24. \left. F(x) \right|_{a}^{b}$$

25. `\sum_{x = a}^{b} f(x)`

$$25. \sum_{x = a}^{b} f(x)$$

26. `\prod_{x = a}^{b} f(x)`

$$26. \prod_{x = a}^{b} f(x)$$

A.6 Sequences and Summation Notation

Often, especially in the context of optimization, summation notation is used to find the sum of a sequence of terms. The summation below represents a summing of the first five values of of a sequence of the variable x. The location of the values within the sequence are given by an index value, i in this case.

$$\sum_{i=1}^{5} x_i = x_1 + x_2 + x_3 + x_4 + x_5$$

Coefficient values may also be included in a summation, as shown below.

$$\sum_{i=1}^{5} (10-i)x_i = 9x_1+8x_2+7x_3+6x_4+5x_5$$

A.7 Applications of Summation

Sequence applications of summation notation can be very practical in that we can extract real-world data values given in an array or a matrix.

For example, let's imagine that the itemized cost for the production of a product from start to finish is that which is given in the table below.

```r
##Load Library
library(pander,quietly = TRUE)

#Load Data Table
M1<-matrix(c(11,12,13,14,15), ncol=1)
rownames(M1)<-list("Design", "Materials",
                   "Production", "Packaging", "Distribution")
colnames(M1)<-list("Product 1")
pander(M1,
       caption="Table 1: Itemized Production Costs for Product 1")
```

Table A.1: Table 1: Itemized Production Costs for Product 1

	Product 1
Design	11
Materials	12
Production	13
Packaging	14
Distribution	15

We might wish to determine the total cost to produce the product from start to finish. We can extract the data from our cost matrix and use summation to find the total cost.

$$\begin{aligned}
&\sum_{i=1}^{5} x_i \\
&= x_1 + x_2 + x_3 + x_4 + x_5 \\
&= 11 + 12 + 13 + 14 + 15 \\
&= 65
\end{aligned}$$

Let's say we now have additional products being produced.

```
#Load Data Table 2
M2<-matrix(c(11,12,13,14,15,21,22,23,24,25,31,32,33,34,35), ncol=3)
rownames(M2)<-list("Design", "Materials", "Production", "Packaging", "Distribution")
colnames(M2)<-list("Product 1", "Product 2", "Product 3")
pander(M2, caption="Table 2: Itemized Production Costs for Three Products")
```

Table A.2: Table 2: Itemized Production Costs for Three Products

	Product 1	Product 2	Product 3
Design	11	21	31
Materials	12	22	32
Production	13	23	33
Packaging	14	24	34
Distribution	15	25	35

We might wish to determine the cost to produce each of the three products from start to finish. We could show this with the following summation notation.

$$\sum_{i=1}^{5} x_{i,j} \ \ \forall \ j$$

This notation indicates that we are summing the cost values in the i rows for each product in column j. Note that the symbol $\forall$ shown in the summation above translates to the phrase"for all." The summation expression above can be interpreted as "the sum of all values of $x_{i,j}$, starting with an initial value of $i = 1$, *for all* values of j." The expression will result in j summations.

$$\begin{aligned}
&\sum_{i=1}^{5} x_{i,j} \quad \forall\, j \\
&= x_{1,1} + x_{2,1} + x_{3,1} + x_{4,1} + x_{5,1} \\
&= 11 + 12 + 13 + 14 + 15 \\
&= 65 && \text{Cost for Product 1}
\end{aligned}$$

AND

$$\begin{aligned}
&= x_{1,2} + x_{2,2} + x_{3,2} + x_{4,2} + x_{5,2} \\
&= 21 + 22 + 23 + 24 + 25 \\
&= 115 && \text{Cost for Product 2}
\end{aligned}$$

AND

$$\begin{aligned}
&= x_{1,3} + x_{2,3} + x_{3,3} + x_{4,3} + x_{5,3} \\
&= 31 + 32 + 33 + 34 + 35 \\
&= 165 && \text{Cost for Product 3}
\end{aligned}$$

We can see that the summation expression resulted in three summation values since j, the number of products, is equal to three. These summation values are 65, 115, and 165, representing the total cost from start to finish to produce Product 1, Product 2, and Product 3 respectively.

A.8 Double Summation

For some projects or models, we may need to add one summation into another. This procedure is called "double summation." Consider the following double summation expression: `\sum_{i=1}^3\sum_{j=1}^4 (i+j)`

$$\sum_{i=1}^{3}\sum_{j=1}^{4}(i+j)$$

Note that the expression contains two $\sum$ symbols, indicating a double summation. The double summation would expand as shown below.

$$\begin{aligned}
&\sum_{i=1}^{3}\sum_{j=1}^{4}(i+j) \\
&\qquad = (1+1) + (2+1) + (3+1) \\
&\qquad + (1+2) + (2+2) + (3+2) \\
&\qquad + (1+3) + (2+3) + (3+3) \\
&\qquad + (1+4) + (2+4) + (3+4)
\end{aligned}$$

A.9 Applications of Double Summation

Consider a transportation application using double summation in which we want to ship a given amount of product X from location i to location j, denoted $X_{i,j}$. The summation notation for this application is shown below.

$$\sum_{i=1}^{n}\sum_{j=1}^{m} X_{i,j}$$

Expanding on the previous summation, we may also want to include shipping costs C from location i to location j, denoted $C_{i,j}$. Combining the amount of product with the related shipping costs would result in the double summation expression shown below.

$$\sum_{i=1}^{n}\sum_{j=1}^{m} C_{i,j}X_{i,j}$$

A.10 Exercises

1. Write all terms and calculate the summation for each exercise.

$$A. \quad \sum_{x=1}^{4} x+3$$

$$B. \quad \sum_{x=0}^{5} 8x-1$$

2. Write as a sum of all terms in the sequence.

$$\sum_{x=1}^{6} (2i)x$$

3. Write a summation to represent the total cost associated with producing three items of Product 1. Use the values from Table 1 to evaluate your summation expression.

4. Write all terms and calculate the summation for each exercise.

$$A. \quad \sum_{i=1}^{3}\sum_{j=1}^{4} (i*j)$$

$$B. \quad \sum_{i=1}^{5}\sum_{j=1}^{2} (3i-j)$$

5. A company compensates its employees with an annual salary and an annual bonus.

A. Write an expression using summation notation to represent the total annual compensation i (including salary and bonus) for each job title j.

B. Write a double summation expression to represent the total amount the company pays annually to compensate all its employees if each job title has n_j employees.
(Need possible sample expression for A. and B.)

A.11 References

http://www.columbia.edu/itc/sipa/math/summation.html
https://www.overleaf.com/learn/latex/Integrals,_sums_and_limits#Sums_and_products
http://www.u.arizona.edu/~kuchi/Courses/MAT167/Files/LH_LEC.0130.Intro.SummationNotation.pdf
https://www.bates.edu/mathematics/resources/what-is-latex/
http://math.hws.edu/gassert/LaTeX_Guide_Title.pdf http://www.statpower.net/Content/310/R%20Stuff/SampleMarkdown.html
http://www.calvin.edu/~rpruim/courses/m343/F12/RStudio/LatexExamples.html http://pages.stat.wisc.edu/~jgillett/371/RStudio/RMarkdown.pdf
https://www.academia.edu/21823064/Creating_Dynamic_Mathematical_Documents_in_RStudio_by_Unifying_Computing_with_R_and_Typesetting_with_LaTeX

Adler, Joseph. 2012. *R in a Nutshell.* Sebastopol, CA: O'Reilly.

Davis, Christopher W. H. 2015. *Agile Metrics in Action: How to Measure and Improve Team Performance.* Manning Press.

Ismay, Chester, and Albert Y. Kim. n.d. *An Introduction to Statistical and Data Sciences via R.* Accessed December 28, 2018.

Kabacoff, Robert. 2011. *R in Action: Data Analysis and Graphics with R.* Shelter Island, NY; London: Manning ; Pearson Education [distributor].