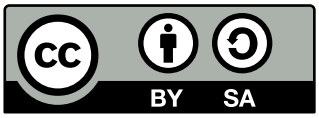
Mathematical Programming with Julia

An open-source approach to
Linear & Mixed Integer Programming
Version 1.0
Julia 1.7.2/JuMP 1.0

Richard Lusby & Thomas Stidsen

ISBN: 978-87-93458-25-3
Publisher: DTU Management, Technical University of Denmark, Denmark, 2022



The authors would like to acknowledge

Thomas would like to thank Charlotte

Richard would like to thank Angelika

1 Introduction

Models of many kinds, both physical and mathematical, have been used for centuries, mainly to gain an understanding of the larger system that is modelled. Mathematical models have been applied in natural sciences and technical sciences at least since Galileo Galilei and Isaac Newton. With the advent of the computer, even models that cannot be solved analytically may produce results. In this book, we only consider two specific types of mathematical models: Linear Programming (LP) and Mixed Integer Programming (MIP). Since the world war II, these two types of models have been crucial for planning, and they have been applied widely in many areas. Both model types require model constructors to define decision variables, an objective function, and one or more constraints defining feasibility. Building LP and MIP models, thus builds understanding of the problem. Having an LP or a MIP model enables optimization, which can provide an optimal solution to the model. Recent technological developments have made the two standard techniques of LP and MIP even more powerful and much more accessible. Optimization software, e.g., Gurobi, Cplex and HiGHS, has not only improved dramatically, but computer hardware has also become much faster, not to mention cheaper. Modern open-source programming languages and some of the solvers, HiGHS, have made the price right; they are free. With this book, we hope to make open-source modelling and solver software for mathematical programming more accessible to a wider audience. Therefore, this book is freely available.

In this book, we focus on **how** to create LP and MIP models. The models are first formulated in terms of mathematical equations, which are then implemented in the programming language Julia `www.julialang.org`, using the JuMP package `https://jump.dev/JuMP.jl/stable/`. The implementation of a mathematical model using the JuMP package closely resembles how the mathematical model would be written. This makes the translation from the formulation to the implementation very intuitive. The book is intended to primarily teach *mathematical programming modelling* and how to use Julia/JuMP for the implementation. It is *not* to teach Julia programming in general. For this reason, the assignments are kept small and simple. There are already many good resources and tutorials available for learning Julia which can be found on `www.julialang.org`, where links to instructive Julia videos are available.

The pedagogical philosophy behind this book is *learning by doing*. We sincerely believe that the best way to learn to formulate good mathematical programming models is through repeated practice on a large number of diverse examples. Therefore, read the

chapters **and** do the exercises. By solving a number of different modelling exercises of increasing complexity, you will learn to formulate mathematical programming models and implement these in Julia/JuMP. If you already have a good understanding of LP you can skip Chapter 3 and if you have a good understanding of MIP you can skip Chapter 5. These chapters are included to give a brief introduction to LP and MIP, and references to further reading on selected topics are given throughout the chapters.

1.1 How to use this book

We suggest that you first install Julia and the Julia packages JuMP and HiGHS, see Chapter 2. You should then read our brief introduction to LP in Chapter 3, but only sections 3.1, 3.2, and 3.3. At this point, you should be able to start working through the LP assignments, in order, in Chapter 4. Some of the problems in Chapter 4 will then require you to read the rest of Chapter 3. After this, you can proceed to the MIP introduction that in Chapter 5. This prepares you for the MIP exercises that follow in Chapter 6. These exercises should be completed in order. Chapter 7 is an auxiliary chapter that explains how to import data into Julia. Finally, we provide some suggestions for further reading in Chapter 8.

1.2 Book website

This book has a dedicated web-site `https://www.man.dtu.dk/MathProgrammingWithJulia` hosted by our university, the Technical University of Denmark. Here you can find:

- This book, both in it's newest version and all previous versions
- Data for some of the assignments
- A correct answer explanation for each assignment, as a pdf file
- Running Julia/JuMP for all assignments

1.3 Donation

This book is based on our work teaching mathematical programming, for which we are paid. Thus, we do not ask for any donations, but in this book we utilize three specific open-source projects you should certainly support if possible:

- Julia, open-source programming language from MIT, 2009. To donate go to: `https://github.com/sponsors/julialang`
- JuMP, open-source mathematical programming modelling Julia package. To donate go to: `https://numfocus.salsalabs.org/donate-to-jump/index.html`
- HiGHS, open-source LP model and MIP model solver. To donate go to: `https://github.com/sponsors/ERGO-Code`

Contents

2 Installation

This book focuses on Mathematical Programming modelling, and we will use Julia and JuMP as the vehicle to implement the models. This is necessary, since these models cannot be solved analytically. We must emphasize that this book is **not** about Julia programming. For this, there are a number of other resources available at `www.julialang.org`. JuMP is a software package which makes it easy to implement Mathematical Programming models in Julia. In this chapter, we will briefly describe how to install Julia, JuMP, selected solvers, and software editors.

2.1 Julia

Julia is a new programming language, initiated in 2009, which is "as fast as C and as easy as Python". Installing Julia is easy, simply download the correct version for your computer from `https://julialang.org/downloads/` and follow the instructions. In this version of the book we use Julia version 1.7.2. Julia supports Windows, Mac, Linux, and FreeBSD.

2.2 JuMP

When Julia is installed, installing JuMP is easy: Start Julia in a command-line. In the command-line write: import Pkg; Pkg.add("JuMP"). Then the JuMP package will be automatically installed. This may take a few minutes.

2.3 Solvers

Given a Mathematical Programming model, either a LP or MIP solver is necessary. JuMP supports a number of solvers, both commercial and open-source:

1. HiGHS: **open-source** solver, for both LP models and MIP models
2. Gurobi: Commercial industry leading solver, for both LP models and MIP models

3. Cplex: Commercial industry leading solver, for both LP models and MIP models
4. Clp: **open-source** solver for solving LP models. Not further developed
5. GLPK: **open-source** solver for solving both LP models and MIP models. Not further developed
6. Cbc: **open-source** solver for solving MIP models. Not further developed
7. Xpress: Commercial industry leading solver, for both LP models and MIP models

In this book, we recommend to use either HiGHS, Gurobi, or CPLEX. HiGHS is an opensource solver which is just as easy to install as JuMP and can be used for free. All the models in this book can be solved with the best open-source solver, HiGHS, if the model is implemented correctly. The two commercial solvers Gurobi and CPLEX are state-of-art solvers and can be obtained for academic usage for free, but they are harder to install and can then only be used for teaching and research. But they are significantly faster for most MIP models.

There is a longer list of supported solvers shown in the JuMP documentation `https://jump.dev/JuMP.jl/stable/installation/` and the list keeps growing. We have picked what we consider the best solvers for use in this book above. Furthermore, a number of the supported solvers are for other types of mathematical models not considered in this book: QP (Quadradic Programming), NLP (Non-Linear Programming), (MI)NLP (Mixed Integer Non-Linear Programming), SOCP (Second-Order Cone Programming), SDP (Semidefinite programming), and several others.

2.4 Editors

The Julia/JuMP models are in principle simple ASCII text files, and any editor can be used to manipulate them, even Windows Notepad can be used. The models can then be executed from the command-line. There are, however, also integrated Julia programming environments:

- Visual Studio Code, `https://code.visualstudio.com`
- Atom: `https://atom.io`

Installing these environments, both of which are open-source, and connecting them to Julia can sometimes be tricky, depending on your computer setup.

2.5 Why Julia ?

The material in this book is part of the material used in our course "Mathematical Programming Modelling" (42112), taught at the Technical University of Denmark. This course about making LP models and MIP models has been taught at our university for more than 40 years, the last 18 years by us. For the entire period until 2018 the specialized Mathematical programming language GAMS `www.gams.com` was used. Since 2019 we have used Julia/JuMP. Using Julia/JuMP comes with a number of advantages:

- The whole system is open-source, when using the HiGHS open-source solver, i.e. free of charge to use.
- Integration with other programs is simpler, because these can simply be programmed in Julia.
- Like GAMS, multiple different solvers can be applied, and easily be switched between different solvers.

Our experience with Julia/JuMP since 2019 has been so good that we consider the days of dedicated Mathematical Programming languages like GAMS, AIMMS, AMPL, MPL to be over. Julia/JuMP is now the only programming language used in the courses at the Operations Research section, at the Department of Management Engineering. It is being used in all our 10+ courses on Operations Research for, e.g., meta-heuristics, Dantzig-Wolfe decomposition, Benders Decomposition and Network optimization.

3 Linear Programming

LP is a classic technique in Operations Research and was invented independently several times during World War II. With the invention of the Simplex algorithm, by George Dantzig Danzig (1947), the models could also be optimized, i.e. the best solution values of the decision variables given an objective and a set of constraints. The optimization algorithms for LP is not the topic of this book, in Chapter 8 we will give links to literature on this important topic. Here we focus on **modelling** problems with the LP model. We will now introduce LP models using the two first assignments from Chapter 4, "Incredible Chairs" and "Incredible Chairs II".

3.1 Incredible Chairs

The company Incredible Chairs produces two different types of chairs, A and B, where one unit (pallet of chairs) of Chair A can be sold for a profit of 4 and one unit of Chair B can be sold for a profit of 6. The chairs are produced on three production lines:

1. Line 1 requires 2 hours to produce one unit of Chair A, and at most 14 hours can be used due to other production on the line.

2. Line 2 requires 3 hours to produce one unit of Chair B, and at most 15 hours can be used.

3. Line 3 requires 4 hours to produce one unit of Chair A and 3 hours to produce one unit of chair B, and at most 36 hours can be used.

Suppose you are the production planner at the company, how will you plan the production of the chairs? The first important thing to identify is what you are in control of. In other words, what are the decisions you can make as production planner? In this case, you can decide **how many** of each type of chair to produce. Since these quantities are unknown and must be decided, we define two **decision variables**, $x_A \geq 0$ and $x_B \geq 0$, which will store the quantity of each type of chair. Notice that the variables have a domain, i.e. an interval of feasible values. Here, they are each constrained to take a positive value. The job for you as the production planner is to find the best possible

values for these two variables.

There are many possible solutions represented by the decision variables (x_A and x_B). As the production planner, you are, however, not interested in a random production plan, stated in the variables x_A and x_B, but you want the **best feasible** production plan. To compare the different solutions, an **objective function** $f(x_A, x_B)$ which quantifies the utility of the solution is used. For this example, the objective function measures the profit of the solution obtained by producing x_A and x_B chairs. The objective function can therefore be stated as:

$$\max f(x_A, x_B) = 4 \cdot x_A + 6 \cdot x_B$$

Not all solutions are possible; they have to be **feasible**. This means that the values assigned to the variables should satisfy the restrictions that are imposed by the production lines. In particular, for each production line, the planned production of chairs cannot exceed the resource limit available. On line 3, the sum of the required resources cannot exceed 36. Given the unknown values of the decision variables, we can now formulate an equation - a **constraint** - that limits the values of x_A and x_B to values values which are possible given the limitations on line 3:

$$4 \cdot x_A + 3 \cdot x_B \leq 36$$

Similarly, there are two other constraints imposed by lines 1&2. They are given below:

$$\begin{aligned} 2 \cdot x_A &\leq 14 \\ 3 \cdot x_B &\leq 15 \end{aligned}$$

The complete LP model is therefore:

$$\begin{array}{lrcrcr} \texttt{Maximize.} & 4 \cdot x_A & + & 6 \cdot x_B & & \\ \texttt{Subject to :} & & & & & \\ & 2 \cdot x_A & & & \leq & 14 \\ & & & 3 \cdot x_B & \leq & 15 \\ & 4 \cdot x_A & + & 3 \cdot x_B & \leq & 36 \\ & x_A & & & \geq & 0 \\ & & & x_B & \geq & 0 \end{array}$$

LP models are **linear**, i.e., the only allowed calculations in objective function and constraints are multiplications between constants and decision variables, and the addition and subtraction of variables.

One of the reasons for the popularity of LP models is that today there are a number of efficient mathematical optimization algorithms - *solvers* - which can be used to **solve** (i.e., find the optimal solution to) the LP model. Solvers optimize the mathematical model and in doing so find the best possible feasible values for the decision variables (here xA and xB). To solve the model, it must be loaded into a solver. In this book, we use the standard programming language Julia in combination with the specialized mathematical programming modelling package JuMP. The syntax is different to the above mathematical model but very similar. Below we list the compact version, in Julia/JuMP.

Compact IC model version:

```
1 using JuMP
2 using HiGHS
3 IC = Model(HiGHS.Optimizer)
4
5 @variable(IC,xA>=0)
6 @variable(IC,xB>=0)
7 @objective(IC, Max, 4*xA+6*xB)
8 @constraint(IC, 2*xA <= 14)
9 @constraint(IC,3*xB <= 15)
10 @constraint(IC,4*xA+3*xB <= 36)
11 print(IC)
12 optimize!(IC)
13 println("Termination status: $(termination_status(IC))")
14 if termination_status(IC) == MOI.OPTIMAL
15     println("Optimal objective value: $(objective_value(IC))")
16     println("xA: ",value(xA))
17     println("xB: ",value(xB))
18 else
19     println("No optimal solution available")
20 end
```

It should be clear that despite the different syntax there is a one-to-one correspondence between the LP model given above and the Julia/JuMP model on lines 5 to 10. Notice that the LP model assumes that chairs can be produced in any number, including fractional amounts. In fact, the optimal production plan is to produce 5.25 of chair A and 5 of chair B. If we impose the requirement that $x_A \in Z^+$, i.e. that only **integer** amounts of chairs can be produced, the model becomes a MIP model. This is described further

in Chapter 5.

When we execute the model in Julia, we get the following printout:

```
Max 4 xA + 6 xB
Subject to
 2 xA <= 14.0
 3 xB <= 15.0
 4 xA + 3 xB <= 36.0
 xA >= 0.0
 xB >= 0.0
Presolving model
1 rows, 2 cols, 2 nonzeros
1 rows, 2 cols, 2 nonzeros
Presolve : Reductions: rows 1(-2); columns 2(-0); elements 2(-2)
Solving the presolved LP
Using EKK dual simplex solver - serial
  Iteration        Objective     Infeasibilities num(sum)
          0     0.0000000000e+00 Ph1: 0(0) 0s
          1    -5.1000000000e+01 Pr: 0(0) 0s
Solving the original LP from the solution after postsolve
Model   status      : Optimal
Simplex   iterations: 1
Objective value     :  5.1000000000e+01
HiGHS run time      :          0.00
Termination status: OPTIMAL
Optimal objective value: 51.0
xA: 5.25
xB: 5.0
```

The model is output in first seven lines as per the instruction line 11 of the Julia/JuMP program. This can be beneficial when debugging a model, but is only ever relevant for relatively small models. The following fourteen lines of output comes from the HiGHS solver. The program then prints out that the optimal solution has been found (line 13 in the program) and then finishes by printing the optimal, maximal profit (51.0) (line 15), and the optimal values of the two decision variables, $xA = 5.25$ and $xB = 5.0$ (line 16 and line 17).

3.2 Incredible Chairs 2

What if the company were not producing two different chairs on three production lines, but say ten different chairs on five different production lines? If the same approach were used, the size of the LP model would increase, and it would be quite tedious to input using the direct approach above. In this case, we can formulate a more abstract model. Given a set of chairs C, indexed by c, and a set of production lines P, indexed by p, we can define a (one-dimensional) vector $Profit_c$, which states the profit for each chair $c \in C$, a (one-dimensional) production capacity vector $Capacity_p$, which states the production capacity of each production line $p \in P$, and a (two-dimensional) matrix $RecourseUsage_{p,c}$ that states how much of the available resource on production line p is needed for production of one unite of chair c.

This gives the following LP model:

$$\begin{array}{lrll} \texttt{Maximize.} & \sum_{c \in C} Profit_c \cdot x_c & & \\ \texttt{Subject to :} & & & \\ & \sum_{c \in C} RecourseUsage_{p,c} \cdot x_c & \leq & Capacity_p \quad \forall p \in P \\ & x_c & \geq & 0 \quad \forall c \in C \end{array}$$

The above model is a very **generic** (chair) production model for profit maximization for any number of chairs and any number of production lines. In fact, any (simple) profit maximizing production problem with a number of products and production resources can be modelled in this way. In the remainder of this book, we will consider two types of variables: **direct variables** and **abstract variables**. The above model has 10 $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10})$ direct variables, but only 1 abstract variable (x_c). Correspondingly, we will consider two types of constraints, **direct constraints** and **abstract constraints**. The above model has 5 direct constraints but only 1 abstract constraint. In general, we measure the **complexity** of a model in the number of abstract variables and constraints, and the **size** of the model is measured in the number of direct variables and direct constraints.

The corresponding Julia/JuMP model can now be written as:

```
IC = Model(HiGHS.Optimizer)

@variable(IC,x[c=1:C] >=0 )

@objective(IC, Max, sum( Profit[c]*x[c] for c=1:C)

@constraint(IC, [p=1:P],
sum( RecourseUsage[p,c] * x[c] for c=1:C) <= Capacity[p])
```

```
9
10 optimize!(IC)
```

Notice that the program actually becomes smaller. Also there is a one-to-one correspondence between the individual elements.

- The definition of the variable $x_c \geq \ \forall\ c$ is now given on one line, line 3:

```
@variable(IC,x[c=1:C] >=0 )
```

- The $\sum$ operator, over an index of a set, is used both in the objective function and in the constraint:
  - The objective function, line 5, $\sum_{c \in C} Profit_c \cdot x_c$ is written as:

```
@objective(IC, Max, sum( Profit[c]*x[c] for c=1:C))
```

  - The constraint: $\sum_{c \in C} RecourseUsage_{p,c} \cdot x_c \leq Capacity_p \forall p \in P$ is written in 7 and 8 as:

```
@constraint(IC, [p=1:P],
sum( RecourseUsage[p,c] * x[c] for c=1:C) <= Capacity[p])
```

In line 1, the data-structure in which the variables, objective function, and constraints are contained is constructed, named IC. Here the solver needed to solve the model is also defined.

```
IC = Model(HiGHS.Optimizer)
```

When the model is considered to be complete, it is transferred to the solver, which finds the optimal solution to the model. This happens in line 10:

```
optimize!(IC)
```

When the optimize! command is given, the full model is transferred to the solver, HiGHS in this case.

The Model construct, and the different model structures @variable, @objective, and optimize! are part of the JuMP package.

Setting up the data-structures Profit, Capacity, and RecourseUsage in Julia is described in Chapter 7. Below we present full Julia/JuMP program with the correct numbers and data-sets for the Incredible Chairs 2 assignment. This format is used in all of the assignments in the book.

Standard Incredible Chairs 2 model version:

```
#************************************************************************
# Incrediblex Chairs 2 assignment, Simple LP
using JuMP
using HiGHS
#************************************************************************

#************************************************************************
# Data
Chairs=["A", "B", "C", "D", "E", "F", "G", "H", "I", "J"]
C=length(Chairs)
ProductionLines=[1, 2, 3, 4, 5]
P=length(ProductionLines)

Profit=[6, 5, 9, 5, 6, 3, 4, 7, 4, 3] # profit for each chair
Capacity=[47, 19, 36, 13, 46] # production line capacity
RecourseUsage=[ # RecourseUsage[p,c] res use for chair c on prod. line p
6  4 2  3 1 10 2 9  3 5;
5  6 1  1 7  2 9 1  8 6;
8 10 7  2 9  6 9 6  5 6;
8  4 8 10 5  4 1 5  3 5;
1  4 7  2 4  1 2 3 10 1]
#************************************************************************

#************************************************************************
# Model
IC2 = Model(HiGHS.Optimizer)

@variable(IC2,x[c=1:C]>=0) # x[c] is the number of chairs produced

# Objective is to maximize profig
@objective(IC2, Max, sum( Profit[c]*x[c] for c=1:C ) )

# Constraints ensuring production line capacity is not exceeded
@constraint(IC2, [p=1:P],
            sum( RecourseUsage[p,c]*x[c] for c=1:C) <= Capacity[p]
            )
#************************************************************************
```

```
38
39 ##########################################################################
40 # Solve
41 optimize!(IC2)
42 println("Termination status: $(termination_status(IC2))")
43 ##########################################################################
44
45 ##########################################################################
46 if termination_status(IC2) == MOI.OPTIMAL
47     println("Optimal objective value: \$(objective_value(IC2))")
48     for c=1:C
49         if value(x[c])>0.001 # only print the chairs actually produced
50             println("No of chairs of type ", Chairs[c], " produced: ",
                ↪  value(x[c]))
51         end
52     end
53 else
54     # model is in-feasible or un-bounded
55     println("No optimal solution available")
56 end
57 ##########################################################################
```

The whole Julia/JuMP program consists of 5 overall parts, divided by the comment lines of asterisk. # is used for comments. The program parts are:

- The initial section, 1-5. Here the program is named and the Using keyword is used to load the different packages.
- The data section, 7-22. Here the data needed for the model is defined. For at description of how to define data, Chapter 7
- The model section, 24-37. Here the decision variables, objective function and constraints are defined.
- The solve section, 39-43. Here the model is transferred to the solver and solved.
- The result section, 45-57. The results from the solver are printed.

3.3 Good modelling practices and debugging

There are a number of recommended practices in the above example:

- Use comments extensively; it makes the programs more readable. The character # makes all text afterwards on that line a comment, which is shown in green above.
- Given a **set** of items, define them as a one-dimensional vector (Chairs line 9) **and** define the length of the vector, as an upper-case letter (C line 10).
- Choose your index names carefully! As an example, c for chairs and C for the number of chairs. This is particularly important since there is no type-checking of sets in Julia/JumP; The instruction RecourseUsage[c,p] is a valid instruction but will lead to an error, it should have been RecourseUsage[p,c].

3.3.1 Debugging

Errors can be one source of frustration when implementing mathematical models. It is important to have strategies to resolve them when they do occur. To provide an example of how one might resolve an error, we deliberately introduce a programming error in the above program and change line 41 to:

```
optimize!(ICC)
```

When calling Julia, we get the following error report:

```
ERROR: LoadError: UndefVarError: ICC not defined
Stacktrace:
 [1] top-level scope
   @ ~/Desktop/IncredibleChairs.jl:41
in expression starting at /Users/thomasstidsen/Desktop/IncredibleChairs.jl:37
```

Error messages may not be very descriptive at times, and very often the best clue is to look for the line number of the error (here 41). The programs that are considered in this book are all of a relatively limited size, usually less than 200 lines of code. However, constraints that conflict with each other can lead to errors which are very difficult to spot.

Assuming that there are no Julia programming errors (as above), the objective of the model can be:

- An optimal and correct solution. We check if an optimal solution is found in line 46. If yes, he solution here is printed out 45-57. Notice use of the two JuMP functions:

1. objective_value(IC2)) : After optimization, if there is an optimal result, we can obtain the optimal value using this function on the model.
2. value(x[c]) : After optimization, if there is an optimal result, we can obtain the optimal value, i.e. the optimal number of chairs which should be produced, using this function

- An optimal but incorrect solution: In all of the assignments in this, the optimal objective value is given. If your model does **not** give this value, then there is definitely an error in your model. If your model **does** get the same optimal value, then your model **may** be correct, but this is not a guarantee.
- Unbounded: If the test in line 46 evaluates to false, the model may be unbounded. This means that the objective tends to $+\infty$ (for a maximization problem) or $-\infty$ (for a minimization problem). In both cases, there is an error in your model (this is not a lost opportunity to earn an infinite amount of money). Often this error comes down to two possibilities: You have forgotten to set the domain of your variables, e.g. $x_i \geq 0$ or you have inadvertently omitted a constraint that limits values of one or more of your decision variables. In the above example, if we add an extra chair, chair no. 11, and define the resource usage to be 0 for all production lines (e.g. it could be produced on a new line we forget to include), and there is a positive profit value, i.e. Profig[11]>0, **then** the solver will try to produce an infinite number of chairs of type 11.
- Infeasible: This means that there is no feasible solution to the model. This can happen; however, in this book all assignments have a feasible, optimal solution. In the above example, if just one of the capacity values take a negative value e.g. Capacity[1]=-47, the whole model becomes infeasible. The simplest way to debug this case is to insert sets of constraints one-by-one to see at which point feasibility of the problem is lost.

3.4 Conditional sums and constraints

Sometimes it is necessary to make conditional constraints or to use sum over a subset of elements in the objective function and/or constraints. This is easy to implement in Julia/JuMP:

- **Conditional constraint:** $\sum_k x_{i,j,k} <= 1 \ \forall \ i, j | i \neq j$

```
@constraint(M, [i=1:I,j=1:J;i!=j], sum( x[i,j,k] for k=1:K) <= 1)
```

  Notice that here the ";" defines the beginning of the condition.

- **Conditional sum:** $\sum_{i,j|i \neq j} x_{i,j}$

```
sum( x[i,j] for i=1:I,j=1:J if i !=j )
```

Notice thate here "if" defines the beginning of the condition.

This kind of conditional statement is necessary in some of the assignments.

3.5 Balance constraints

A very common constraint type used in LP and MIP models is the so-called **balance constraint**. Such a constraint connects two different sets of variables and ensures that the value of their respective sums is exactly the same, i.e., balanced.

Three examples of balanced constraints that you will encounter throughout the exercises include the following:

- *Product flows*: A storage facility where products are first transported from factories into the storage facility and which are then subsequently transported from the storage facility to the customers. Since there is no production at the storage facility and nor is it possible to store products long term, the quantity of product is delivered from the different factories to a storage facility should match precisely the quantity of product delivered from the storage facility to the customers.

- *Inventory management of a certain product*: For a given time horizon, and an initial storage level of the product, the quantity of product on hand at the end of the time horizon must be the initial storage level adjusted by any new production (increasing the quantity on hand) and the amount of product that is sold or released within the time horizon (decreasing the quantity on hand).

- *Flows within a network*: As an example, consider intersections in a road network. The cars driving into any intersections must leave it again.

A good example that you will meet in a later assignment is the inventory management case. This requires one to relate a given timeslot (index) with its previous timeslot (index) and induces boundary certain boundary cases. Consider what is the timeslot before the the first timeslot. In Julia, the in-line if statement can be used. `(A ? B : C)` is an if statement. It evaluates whether `A` is true or false. If `A` is true, then it uses `B` and if `A` is false, then it uses `C`. Therfore, a statement like `(t>1 ? <normal case> : <what to do if this is first time-slot>)` can be used.

3.6 Relaxation and restriction of linear programs

Consider an LP model. By changing its set of constraints, one can either **relax** the model or **constrain** the model further.

Relaxation: If one (or more) of the constraints is removed from the model, then the model has been *relaxed*. The resulting model is termed a *relaxation*. Relaxations lead to optimal objective values that are at least as good as the original model (i.e. an increased value if maximizing and a decreased value if minimizing). The reason for this is that all of the previous solutions are still feasible, but there might also be new solutions that are now feasible due to the omitted constraints.

Restriction: If one (or more) constraint(s) is added to the model, then the model has fewer feasible solutions. Constraining a model more leads to (possibly) worse optimal objective values (i.e., a decreased value if maximizing and an increased value if minimizing). The reason for this is that some of the previously feasible solutions may now be infeasible given the newly added constraints.

3.7 Solver parameters

It is possible to transfer parameters to the solver via Julia/JuMP. This is often not so important for LP models since these are often solved quickly. When MIP models are solved, however, it may be interesting to set, e.g., a time limit or an acceptable optimality gap. Below we provide examples of how to set the values of some useful solver parameters. In the examples, M refers to the model name.

- Set a time limit for the solver. This may mean that the solver is forced to terminate after a specific time. The solver may not find the optimal solution or even find a feasible solution. Below, we demonstrate how the the time limit is set to 300 sec.

  ```
  set_time_limit_sec(M, 300.0)
  ```

- The printout from to solver suppressed with the set_silent function:

  ```
  set_silent(M)
  ```

- Set an acceptable gap, i.e. an acceptable relative gap between the best possible objective value and the current best value. This option becomes critical when working with MIP. This parameter depends on which solver is used:

- Setting a gap of 10 % in HiGHS:

  ```
  set_optimizer_attribute(M, "mip_rel_gap", 0.10)
  ```

- Setting a gap of 10 % in Gurobi:

  ```
  set_optimizer_attribute(M, "MIPGap", 0.10)
  ```

- Setting a gap of 10 % in Cplex:

  ```
  set_optimizer_attribute(M, "CPXPARAM_TimeLimit, 0.10)
  ```

3.8 Julia and JuMP structure

It is important to understand the differences and the relationships between Julia, JuMP, and the solver. In this section, we will briefly describe this.

3.8.1 Julia

Julia is a fully-fledged programming language that is used in many different application areas. In this book, we will only use a small subset of the Julia constructs.

3.8.2 JuMP

JuMP is a package for Julia and is specialized for creating Mathematical Programming models. In this book, we will only consider LP models and MIP models. JuMP can, however, be used for a number of other model types. JuMP extends Julia with some new constructs, created as macros. Consider the following example taken from Incredible Chairs, see Chapter 3.1.

- The model statement:

  ```
  IC = Model(HiGHS.Optimizer)
  ```

  The model statement is used to define model.

- The variable statement:

  ```
  @variable(IC,xA>=0)
  ```

  The variable statement is used to define a variable for a given model, in this case IC.

- The objective statement:

  ```
  @objective(IC, Max, 4*xA+6*xB)
  ```

  The objective statement is used to define an objective function for a given model, in this case IC.

- The constraint statement:

  ```
  @constraint(IC, 4*xA+3*xB <= 36)}
  ```

  The constraint statement is used to define one (or more constraints) for a mathematical programming model, in this case IC.

- The optimize statement:

  ```
  optimize!(IC)
  ```

  The optimize statement is used to transfer the model to the **solver**.

- The value statement:

  ```
  value(xA)
  ```

  This function is used to retrieve the value of a JuMP variable in the solution found by the solver. This function can only be used **after** the optimize! statement.

There are a number of important details:

- The first statement is always the model statement.
- The order of variable, constraint and objective statements does not matter. In this book, we will always first write the variables, then the objective function and finally the constraints
- There can be multiple variables, of different names, and multiple different constraints, but only one objective function. If several objective statements are issued, only the last before the optimize statement is used.
- The optimize statement builds the model in a solver-readable form and transfer the control to the solver.
- The variables of a JuMP model do **not** hold a value until after the optimize statement, The solver assigns the values, which can then be retrieved using the value statement. For this reason, the JuMP variables cannot be used in Julia constructs prior to the optimize statement.

3.8.3 Solver

One of the major advantages of using JuMP is that one can easily toggle between different solvers for the same model. The only changes necessary are in using statement at the top of the program and also in the model construction statement. JuMP supports a number of solvers for both LP models and MIP models, see the list of solvers we recommend in Section 2.3.

All models in this book can be solved in reasonable time using the opensource HiGHS solver. The two leading solver commercial solvers Gurobi and Cplex are much faster, in particular for MIP models, but they are relatively expensive. Academics and students can usually get access to an academic license for the solvers, but this requires a separate installation. The opensource solvers are, however, very simple to install in Julia, using the package system.

3.8.4 Structure

We are now ready to briefly describe the interactions of a model in Julia/JuMP with the solver, see Figure 3.1. The lowest level, i.e., the green part, corresponds to the Julia code part. When the program is executed, the initial part and any data will be initialized in the Julia program. This only affects the Julia part. The model is then built using the JuMP package. This is essentially an interface between the Julia program and the solvers (of which three are indicated in the figure). The JuMP package provides the model function and the @variable, @objective, and @constraint commands. These commands are used to build the LP model or MIP model. When the optimize statement is executed, the constructed JuMP model is transferred to the solver, which will then start to solve the model. When the solver terminates, the optimal objective value and corresponding solution is available via JuMP and can be queried from the Julia part using the respective functions objective_value function() and the value().

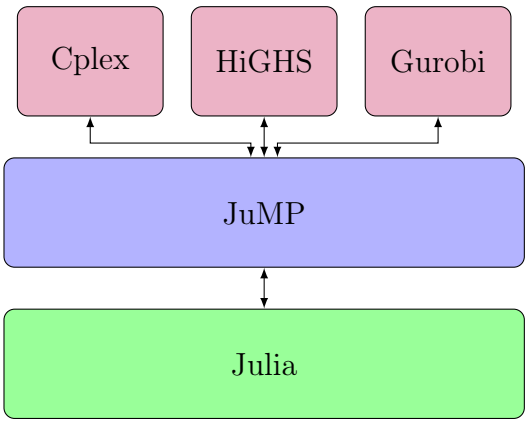


Figure 3.1: Julia/JuMP and Solver Structure

An important point is that the values of the JuMP variables, defined with the @variable statement, **cannot be used as variables in the Julia program !**. This is because, the value is set **only** by the solver and it can afterwards only be queried, not altered, using the value() function. We illustrate this with two deliberate errors in the Julia/JuMP program below (both will lead to errors in the compilation):

- In line 10, we attempt to make an in-line conditional statement in a constraint using the value of the JuMP variable xA. This is wrong for two reasons:
  - The xA JuMP variable does not have a value before the optimize!() statement.
  - Even if it had a value, the value() function is needed to extract this value
- In line 13, the value of the xA JuMP variable is used in an if statement, which is an error for the same reasons.

In line 21, the value() statement is used correctly to obtain the optimal value of the xA JuMP variable.

```
1 using JuMP
2 using HiGHS
3 IC = Model(HiGHS.Optimizer)
4
5 @variable(IC,xA>=0)
6 @variable(IC,xB>=0)
7 @objective(IC, Max, 4*xA+6*xB)
8 @constraint(IC, 2*xA <= 14)
9 @constraint(IC,3*xB <= 15)
```

```
@constraint(IC,4*xA+3*xB <= 36 + $(xA>2 ? 10 : 0 ) ) # <<<<<< error
print(IC)

if xA>3 # <<<<<< error
    println("High production of chair A")
end

optimize!(IC)
println("Termination status: $(termination_status(IC))")
if termination_status(IC) == MOI.OPTIMAL
    println("Optimal objective value: $(objective_value(IC))")
    println("xA: ",value(xA)) # <<<<<< correct
    println("xB: ",value(xB))
else
    println("No optimal solution available")
end
```

4 Linear Programming problems

This chapter is the first exercise chapter of the book. All the assignments in this chapter focus on LP. We categorize the assignments into groups of (somewhat) related problems and attempt to gradually increase the complexity of the models. The first two assignments, Incredible Chairs and Incredible Chairs 2, have already been described in detail in Section 3.1.

4.1 Supply and demand optimization

The following problems all involve either profit optimization with resource restrictions or cost minimization with demand requirements.

4.1.1 Incredible Chairs

The company Incredible Chairs produces two different types of chairs, A and B, where one Chair A can be sold for a profit of 4 and one Chair B can be sold for a profit of 6. The chairs are produced on three production lines:

1. Line 1 requires 2 hour to produce one unit of Chair A, and at most 14 hours can be used due to other production on the line.
2. Line 2 requires 3 hours to produce one unit of Chair B, and at most 15 hours can be used.
3. Line 3 requires 4 hours to produce one unit of Chair A and 3 hours to produce one unit of chair 2, and at most 36 hours can be used.

The objective is to maximize the total profit, given the above mentioned profit limitations. This leads to the following LP:

$$
\begin{array}{lrcrcr}
\texttt{Maximize.} & 4 \cdot x_A & + & 6 \cdot x_B & & \\
\texttt{Subject to :} & & & & & \\
 & 2 \cdot x_A & & & \leq & 14 \\
 & & & 3 \cdot x_B & \leq & 15 \\
 & 4 \cdot x_A & + & 3 \cdot x_B & \leq & 36 \\
 & x_A & & & \geq & 0 \\
 & & & x_B & \geq & 0
\end{array}
$$

On the next page, the corresponding model is given in Julia.

Compact Julia/JuMP program/model:

```
using JuMP
using HiGHS
IC = Model(HiGHS.Optimizer)

@variable(IC,xA>=0)
@variable(IC,xB>=0)
@objective(IC, Max, 4*xA+6*xB)
@constraint(IC, 2*xA <= 14)
@constraint(IC,3*xB <= 15)
@constraint(IC,4*xA+3*xB <= 36)
print(IC)
optimize!(IC)
println("Termination status: $(termination_status(IC))")
if termination_status(IC) == MOI.OPTIMAL
    println("Optimal objective value: $(objective_value(IC))")
    println("xA: ",value(xA))
    println("xB: ",value(xB))
else
    println("No optimal solution available")
end
```

Assignment 1.1

Downloaded the above compact IC model from the book website `https://www.man.dtu.dk/MathProgrammingWithJulia`, or simply copy-paste from the pdf file. If you copy-paste you will probably have to clean up the code. Run the code and check the results.

Optimal objective value for Incredible Chairs Problem: 51.0 with $x_A = 5.25$ *and* $x_B = 5.0$
The standard format for the models in this book is a bit more expansive and is given below.

```
#*************************************************************************
# Incredible Chairs, Simple LP
using JuMP
using HiGHS
#*************************************************************************

#*************************************************************************
```

```
# PARAMETERS
Chairs = ["A", "B"]
C=length(Chairs)
ProductionLines = ["1","2","3"]
P=length(ProductionLines)

Profit = [4,6]
Capacity = [14,15,36]
RecourseUsage =
    [
        2 0;
        0 3;
        4 3
    ]
#*************************************************************************

#*************************************************************************
# Model
IC = Model(HiGHS.Optimizer)

@variable(IC,x[1:C]>=0)

@objective(IC, Max, sum( Profit[c]*x[c] for c=1:C))

@constraint(IC, [p=1:P], sum( RecourseUsage[p,c]*x[c] for c=1:C) <= Capacity[p])
#*************************************************************************

#*************************************************************************
# Solve
solution = optimize!(IC)
println("Termination status: $(termination_status(IC))")
#*************************************************************************

#*************************************************************************
# Solution
if termination_status(IC) == MOI.OPTIMAL
    println("Optimal objective value: $(objective_value(IC))")
    for c=1:C
        println("x[", Chairs[c], "] : ",value(x[c]))
    end
else
    println("No optimal solution available")
```

```
end
#**************************************************************************
```

Assignment 1.2

Downloaded the above compact IC model from the book website `https://www.man.dtu.dk/MathProgrammingWithJulia`, or simply copy-paste from the pdf file. If you copy-paste you will probably have to clean up the code. Run the code and check the results.

*The model is **identical** with the previous one, so the optimal objective value for Incredible Chairs Problem: 51.0 with* $x_A = 5.25$ *and* $x_B = 5.0$

NOTICE: There is no problem in copying the IncredibleChairs.jl standard program for the next assignments, BUT be care full. Remember to update all the data in the program. In particular make sure to name the sets, cardinality values and index names correctly. Here the sets are Chairs and ProductionLines. The cardinality values C and T. And the indexes are c and t.

4.1.2 Incredible Chairs 2

Incredible Chairs has decided to extend their line of products to 8 other chairs and now the model should include all the 5 production lines. Below we give the data, using the same names as in Section 3.1. You can read about how to define data in Julia in Chapter 7.

The profit for each chair is given in the table below.

	Chair types									
	A	B	C	D	E	F	G	H	I	J
Profit (€)	6	5	9	5	6	3	4	7	4	3

Table 4.1: Profit for each chair sold (€)

The capacity for each of the production lines is given in the table below.

	Production lines				
	1	2	3	4	5
Production line capacity	47	19	36	13	46

Table 4.2: Production line capacity

The resources needed for each chair on each line is given in the table below.

	Chair types									
	A	B	C	D	E	F	G	H	I	J
1	6	4	2	3	1	10	2	9	3	5
2	5	6	1	1	7	2	9	1	8	6
3	8	10	7	2	9	6	9	6	5	6
4	8	4	8	10	5	4	1	5	3	5
5	1	4	7	2	4	1	2	3	10	1

Table 4.3: Resource usage for each chair on each production line

Assignment 1.3

Formulate an LP to maximize the profit of the chair production

Optimal objective value for Incredible Chairs 2: 23.04 €.

4.1.3 Class Jobs

Your little sister and four of her friends from primary school have volunteered to do a number of different community jobs (like picking up trash etc.). The five children $c \in C$ have five jobs $j \in J$ to choose between. To make the work more attractive, each child is asked to rank their preference for each job on a scale from 1 to 5 (with five being best). This information is given in Table 4.4, i.e. $Wish_{j,c} \in \{1, 2, 3, 4, 5\}$. You can read about how to define data in Julia in Chapter 7.

	c_1	c_2	c_3	c_4	c_5
j_1	1	3	2	5	5
j_2	5	2	1	1	2
j_3	1	5	1	1	1
j_4	4	5	4	4	4
j_5	3	5	3	5	3

Table 4.4: Job preferences

Assignment 1.4

Formulate an LP that can be used to find the best assignment of the jobs to the children. The best assignment is the one that maximizes the total preference score. Each child can only do one job, and each job can only be done by one child.

Optimal solution for Class Jobs: 24

Is the solution **integer**? In other words, are all decision variable values equal to either 1 for the job assignments made, and 0 for the job assignments not made? They should be. This is a feature of this type of problem. They are generally referred to as **assignment problems**. If we add more constraints, this property is usually not guaranteed.

4.1.4 Chair Transport

After finishing your production planning job at Incredible Chairs, the company makes you responsible for all logistics operations at the company. The company has two chair production sites (P_1 and P_2) and four major storage depots (D_1, D_2, D_3 and D_4). The different retailers which sell the chairs supplied directly from the depots. Each month a new transportation plan needs to be created. From a transport perspective, the chairs are identical. This means that they can be produced at both production sites and can be stacked efficiently. Table 4.5 specify the plant production capacity and Table 4.6 specify the deliveries necessary for each depot. You can read about how to define data in Julia in Chapter 7.

P_1	P_2
7500	8500

Table 4.5: Plant production capacity

D_1	D_2	D_3	D_4
3250	3500	3500	3000

Table 4.6: Depot chair demand

The objective is to minimize the monthly transport costs. The average cost for transporting one chair one kilometer is €0.0375. The distances between each of the production plants and each of the depots is given in Table 4.16 below.

	D_1	D_2	D_3	D_4
P_1	137	92	48	173
P_2	54	109	111	85

Table 4.7: Distances between the production sites and the depots (km)

Assignment 1.5

Find the cheapest transport plan that gets the necessary chairs from the production sites to the depots.

Optimal objective value for chair transport problem: €34518.75

4.2 Production with extra constraints

The problems included in this section are production problems like, e.g., Incredible Chairs, but with extensions.

4.2.1 Jewellery Production

Your aunt runs a small jewellery production company that produces relatively cheap jewellery, necklaces (five different types), primarily for tourists. The profit for each necklace, price with material costs subtracted, is given in Table 4.8.

	Necklace type				
	1	2	3	4	5
Profit (€)	50	35	85	60	55

Table 4.8: Profit per necklace type (€)

Three machines are used to produce different jewellery components, which are then assembled by hand by two employees. Your aunt handles the machines herself; they are automatic but require some training to use. To produce the components for each necklace different amounts of time are required on each of the three machines. The machine time needed (in minutes) is given in Table 4.9

	Necklace type				
Machine	1	2	3	4	5
1	7	0	0	9	0
2	5	7	11	0	5
3	0	3	8	15	3

Table 4.9: The machine time needed for each necklace (minutes)

Additionally, each necklace must be assembled by one worker, and this takes some time (also measured in minutes). These times are given in Table 4.10. A full work day, both for the assembly and the machines, is 7.5 hours.

	Necklace type				
	1	2	3	4	5
Assembly time	12	3	11	9	6

Table 4.10: Necklace assembly time (in minutes)

Assignment 2.1

Formulate an LP that can be used to maximize the daily profit. You must determine the number of each type of necklace to produce, while taking into consideration the available machine and assembly hours.

Optimal objective value for Product Mix: €5896.15.

This is a simple example of a so-called *product mix* problem, a common application of LP.

Your aunt realize that she cannot sell all of the necklaces she makes, but estimates an average demand for each type. These values are given in Table 4.11

	Necklace type				
	1	2	3	4	5
Average Demand	25	10	12	15	60

Table 4.11: Average demands

Assignment 2.2

Find a revised production plan that maximizes the daily profit. The production of the necklace types should not exceed the average demand values. Before you implement this constraint, try to reason if the additional constraints are going to increase or decrease the profit?

Optimal objective value for jewellery production: €5643.18.

Your aunt considers whether to hire another assembly worker. Is this a good idea ?

4.2.2 Micro brewery

You and a friend want to set up a microbrewery, both, to satisfy your own demand, but also to supply some nearby cafes with beer. The deal is that the cafes will get the beer relatively cheaply, but they will have to buy a certain amount (in litres) each month. The demand is given in Table 6.2:

	Months											
	jan	feb	mar	apr	may	jun	jul	aug	sep	oct	nov	dec
Demand (L)	15	30	25	55	75	115	190	210	105	65	20	20

Table 4.12: Monthly demand for beer each month (in liters)

You set up the brewing system in your own room (you live at home with your parents) but can only brew at most 120 liters per month. This is a problem, given the demand, so you convince your mother that it is alright to store crates of beer in the basement. Her demand (to avoid excessive storage) is a storage cost of 1 € per liter of beer stored each month. Furthermore, you can store at most 200 liters of beer in the basement.

Assignment 2.3

Formulate an LP that can determine the optimal beer production and storage strategy, minimizing the storage costs.

Hint: You need a beer production variable for each month and a storage variable for each month. The storage variable should reflect what is in storage *at the end of the month*. Then, you can make a *balance constraint*, relating what is in storage at the end of a month, with what was in storage at the end of the previous month. The previous month is the current month minus 1. *However*, if it is the *first* month (jan), then there is no storage. If your storage variable is y[m] (where m is the month index), you can add the previous month in Julia with the following code `(m>1 ? y[m-1] : 0)` In general, this code `(A ? B : C)` is an if statement. It evaluates whether `A` is true or false. If `A` is true then it uses `B` and if `A` is false then it uses `C`.

Assignment 2.4

Implement your LP model in Julia/JuMP.

Optimal objective value for micro brewery broblem: €560.0

4.2.3 Vaccine Production

Vaccines are used to immunize people against serious diseases/illnesses and are currently a very topical subject. You are employed at a large pharmaceutical company and must schedule the next four weeks of production of a particular vaccine. In particular, you must schedule the production in such a way that the number of vaccines obtained are maximized. Since vaccines are extremely valuable, it is important to produce as many vaccines as possible and not unnecessarily waste the raw material from which they are made.

The vaccine doses are synthesized from bags of so-called drug substance and are stored in vials. Associated with each bag is a potency, which indicates the strength and quality of the drug substance. Until it is processed, the potency of a bag gradually decreases over time. To be effective, all vaccine doses (number of vials) that are produced must have a certain strength. As the potency drops, the number of vaccine doses that can be produced from a bag decreases. There are currently 16 bags of drug substance available of varying quality, some have a high potency and others have a low potency. Weaker bags can be mixed with stronger bags to ensure the required strength of the vaccine doses. During production of the vaccine doses, other ingredients, like stabilizer and buffer solution, must be added. Stabilizer is added to preserve the strength of the vaccine over time, while buffer helps control the pH of the dose.

Table 4.13 provides the necessary information on the bags of drug substance in stock. For each of the coming four weeks, the number of vaccine doses (or vials) that can be obtained from each bag is given, along with the required buffer volumes. The buffer volume already incorporates information on the required stabilizer volume (there is a relationship between the two). If a bag has a negative buffer value, then the bag cannot on its own provide any vaccine doses of the required strength and must be mixed with bags that could (i.e bags with a positive buffer value). Your task is to decide in which weeks to process each of the bags (production of a bag can be split across weeks). The production of a *batch* of bags in a given week must satisfy a number of practical requirements These are given below. Note that exactly one batch must be produced each week, and that the buffer volume required is directly proportional to the fraction of a bag used.

1. Processing a bag is optional.
2. A single batch cannot contain fewer than 60,000 vials and not more than 120,000.
3. The total buffer volume added must be at least zero.

The vaccine mixing process is visualized below in Figure 4.1. In each week, a batch of vaccine is produced by mixing liquids from one or more of the bags of drug substance.

The contents of a bag can be used over all 4 weeks, but naturally, the liquid in each bag can only be used one time. It is optional to use a bag. The number of vials (vaccine doses) produced in each week should be between 60000 and 120000. This is the minimum and maximum amounts which can be used in the vaccine roll-out. Finally, the PH value needs to be at a certain level, therefore the requirement of the buffer value to be positive.

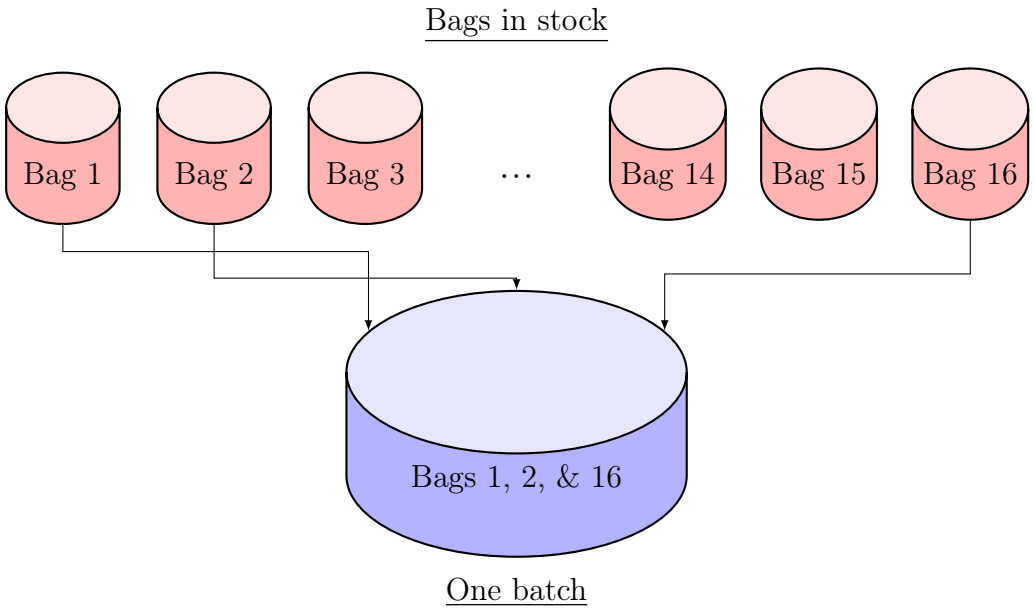


Figure 4.1: Batch mixing process

Assignment 2.5

Assuming that a bag can be fractionally assigned to multiple weeks (and that the vials and buffer are proportional to the amount of the bag produced), formulate an LP that can be used to identify a production plan that maximizes the number of vials produced. Implement your model in Julia/JuMP

Optimal objective value for vaccine vroduction: 319,425.62

	Vial count				Buffer volume (L)			
Bag	w1	w2	w3	w4	w1	w2	w3	w4
1	29317	29223	29129	29035	6.26	6.21	6.17	6.12
2	18551	18491	18432	18373	0.23	0.20	0.17	0.14
3	29441	29346	29252	29158	6.29	6.24	6.19	6.15
4	16971	16916	16862	16808	0.21	0.19	0.16	0.13
5	37084	36965	36846	36727	7.92	7.86	7.80	7.74
6	19288	19226	19164	19102	0.24	0.21	0.18	0.15
7	16315	16262	16210	16158	-2.14	-2.17	-2.19	-2.22
8	10949	10914	10879	10844	-3.59	-3.61	-3.63	-3.65
9	14071	14025	13980	13935	-1.5	-1.52	-1.55	-1.57
10	18002	17945	17887	17829	-2.37	-2.39	-2.42	-2.45
11	10766	10731	10697	10662	-3.53	-3.55	-3.57	-3.58
12	18190	18131	18073	18015	-1.94	-1.97	-2.00	-2.03
13	18296	18237	18178	18120	-2.40	-2.43	-2.46	-2.49
14	24426	24347	24269	24191	3.57	3.54	3.50	3.46
15	19768	19704	19641	19577	-2.11	-2.14	-2.17	-2.20
16	19214	19153	19091	19030	-0.43	-0.46	-0.49	-0.52

Table 4.13: Weekly data for each bag. You can find these data on the book web-site `https://www.man.dtu.dk/MathProgrammingWithJulia`

4.3 Transport and project scheduling

In this section the assignments consider LP models for transport and networks.

4.3.1 Chair Distribution

Your successful role as head of logistics at Incredible Chairs has helped the company spot a new business opportunity; instead of letting the retailers pick up the chairs themselves, Incredible Chairs offers to transport the chairs to the six largest retailers R_1, R_2, R_3, R_4, R_6. Each retailer has a monthly demand and does not mind from where the chairs are delivered, i.e. which depot, or even if the chairs are delivered directly to their store from the production sites. Your job is to plan the (possible) transport between the production sites, depots, and retailers. Each plant still has the same monthly capacity that cannot be exceeded, while the depots too have the same capacities. This information is given (again) in Tables 4.14 and 4.15, respectively.
The average cost of transport per chair per km is €0.0375. The distances (in km) between each of the production sites and each of the depots is unchanged and is given in

P_1	P_2
7500	8500

Table 4.14: Production site capacities

D_1	D_2	D_3	D_4
3250	3500	3500	3000

Table 4.15: Depot capacities

Table 4.16.

	D_1	D_2	D_3	D_4
P_1	137	92	48	173
P_2	54	109	111	85

Table 4.16: Distances in kilometers

Now, however, there is also the option of sending chairs between depots and retailers and between plants and retailers. The respective distances are given in Tables 4.17 and 4.18. Each retailer has a monthly chair requirement that must be met. These demands are given in Table 4.19.

	R_1	R_2	R_3	R_4	R_5	R_6
P_1	307	260	215	196	148	268
P_2	234	173	194	264	204	218

Table 4.17: Distances between production sites and retailers (km)

	R_1	R_2	R_3	R_4	R_5	R_6
D_1	109	58	65	187	128	88
D_2	214	163	54	89	26	114
D_3	223	173	97	71	29	162
D_4	81	51	133	239	170	155

Table 4.18: Distances between depots and retailers (km)

R_1	R_2	R_3	R_4	R_5	R_6
1500	2500	2000	3000	2000	3000

Table 4.19: Retailer demand

Assignment 3.1

Formulate an LP to find the minimum-cost transportation plan and solve model using Julia/JuMP.

Optimal objective value for the distribution problem: €71296.87

Hint: The model becomes simplest if you use three types of variables for the transport (Production sites → Depots, Production sites → Retailers, and Depots → Retailers).

Hint: If you get €36778.12 then you should start to think about where the goods in the depots come from.

4.3.2 Project Scheduling

Your father is unhappy with the tool shed in the back garden of his house, see the picture. He wants a new one and has found a do-it-yourself kit tool shed from a hardware store. There is, however, a significant amount of work which has to be done. He splits the total project into tasks, some of which can be done by you, others must be done by specialists (electricians, plumbers etc.). He asks all contractors to commit to a day usage for the task. He offers you the first and last task and would like you to be in charge of the project. Given the list below, you need to make a *project plan*, i.e. when should each task be performed. To keep things simple, we simply work with full working days. Notice that some tasks need to be finished before other tasks can be performed. As an example, the roof can only be installed after the walls have been erected.

Figure 4.2: The tool shed to be replaced

The list of all tasks to be completed is as follows:

1. Remove Stuff: Move the tools and other stuff out of the current shed and temporarily store them (duration 1 day)
2. Shed Demolition: Knock down the old shed (after Remove Stuff, duration 3 days)
3. Pipe Work: Install plumbing, there will be running water in the new shed (after Shed Demolition, duration 5 days)
4. Electric Work: There is electricity for tools and lights in the new shed (after Shed Demolition, duration 3 days)

5. Flagstones: The area around the shed should be covered in flagstones (after Pipe Work and Electric Work, duration 7 days)
6. Shed Foundations: Set the foundations for the new shed (after Pipe Work and Electric Work, duration 4 days)
7. Wood Frames: Erect the wooden walls of the shed (after Shed Foundations, duration 3 days)
8. Painting: Paint the walls (after Wood Frames, duration 1 day)
9. Roofing: Build the roof (after Wood Frames, duration 1 day)
10. Roofing Felt: Put roofing felt on the roof (after Roofing, duration 1 day)
11. Interior Installations: Install racks, lights etc. (after Roofing, duration 3 days)
12. Insert Stuff: Get the stuff out of the temporary storage and store them in the new shed (after Roofing, duration 1 day)
13. Hand over: Present the new shed to your father (After flagstones, painting, roofing felt, interior installations and insert stuff, duration 0 days)

Assignment 3.2

Implement the Tool Shed Project Scheduling in Julia/JuMP and minimize the number of days needed before hand over

Optimal objective value for the Shed Project Scheduling : 20.0 days

Hint : Notice that each task cannot start before the previous task(s) are finished. Task Remove Stuff does not have a predecessor and task Hand Over does not have a successor.

Early finish
Your father is not happy about the project plan, he needs the tool shed to be finished earlier. Looking at the plan, you contact all the contractors, who use more than one day for the job, and ask how much they can speed up the work, and how much overtime pay they need per speed-up day. This gives the information below.

1. Shed Demolition: Can be reduced 1 day at an overtime pay of 500 per day
2. Pipe Work: Can be reduced 3 days at an overtime pay of 2000 per day
3. Electric Work: Can be reduced 1 day at an overtime pay of 4000 per day

4. Flag Stones: Can be reduced 4 days at an overtime pay of 1000 per day
5. Shed Foundations: Can be reduced 2 days at an overtime pay of 1000 per day
6. Wood Frames: Can be reduced 2 days at an overtime pay of 1500 per day
7. Interior Installations: Can be reduced 2 days at an overtime pay of 1500 per day

You realize that there is a problem. The completion time depends on the budget available for overtime, and the question your father has to answer is: How much is he prepared to pay for an earlier finish. Before you return to your father you hence first calculate how quickly the tool shed can be completed if there is an unlimited budget. For comparison, you also calculate the earliest completion times if you have a budget of 5000 and if you have a a budget of 10000.

Assignment 3.3

Implement the Shed Project Scheduling in Julia/JuMP and minimize the number of days before hand over, taking into account an unlimited budget, a 5000 budget and a 10000 budget.

Optimal solution for the Shed Project Scheduling, unlimited budget: 10.0 days
Optimal solution for the Shed Project Scheduling, budget of 5000 : 15.33 days
Optimal solution for the Shed Project Scheduling, budget of 10000 : 12.80 days

5 Mixed Integer Programming

LP models are very useful, but there are many situations where the planning problem in question cannot be modelled sufficiently precisely. In many problems fractionality of the variables is problematic. Consider the Incredible Chairs problem again, now Incredible Chairs III: Since it is chairs that are produced, they need to be produced in **whole** numbers. Updating the model to achieve this in Julial/JuMP is simple (last line of model):

$$
\begin{array}{lrcrcl}
\texttt{Maximize.} & 4 \cdot x_A & + & 6 \cdot x_B & & \\
\texttt{Subject to :} & & & & & \\
 & 2 \cdot x_A & & & \leq & 14 \\
 & & & 3 \cdot x_B & \leq & 15 \\
 & 4 \cdot x_A & + & 3 \cdot x_B & \leq & 36 \\
 & x_A, x_B \in Z^+ & & & &
\end{array}
$$

This is a Mixed Integer Programming (MIP) model where the variables are allowed to be restricted to only allowing **integer** values, as described above in the **domain constraints**. The below program will deliver the optimal **integer** solution. Only two things have been changed:

- In the definition of the variables, the variable is now declared to be integer, line 5 and line 6:

```
@variable(IC,xA>=0,Int)

@variable(IC,xB>=0,Int)
```

- When retrieving the value of the variable found by the solver, using the value() function, we **round** the resulting variable to become an integer type, line 16 and line 17:

```
println("xA: ",round(Int64,value(xA)))

println("xB: ",round(Int64,value(xB)))
```

Compact IC model version:

```
using JuMP
using HiGHS
IC = Model(HiGHS.Optimizer)

@variable(IC,xA>=0,Int)
@variable(IC,xB>=0,Int)
@objective(IC, Max, 4*xA+6*xB)
@constraint(IC, 2*xA <= 14)
@constraint(IC,3*xB <= 15)
@constraint(IC,4*xA+3*xB <= 36)
print(IC)
optimize!(IC)
println("Termination status: $(termination_status(IC))")
if termination_status(IC) == MOI.OPTIMAL
    println("Optimal objective value: $(objective_value(IC))")
    println("xA: ",round(Int64,value(xA)))
    println("xB: ",round(Int64,value(xB)))
else
    println("No optimal solution available")
end
```

From a modelling point of view, these changes are small and almost trivial. However, three important points need to be made, which are elaborated upon in the below sections:

1. The amount of important problems which can be better modelled using MIP models is **vast**. In the next section, we will briefly present a number of integer modelling tricks.

2. The required solution time of the solver may become critical: Finding the optimal solution for a MIP model becomes a much more challenging computational problem than for an LP model.

3. The type of algorithm necessary to find the optimal solution changes. To better understand the increased optimization complexity, we give a very brief introduction to the classic Branch & Bound algorithm.

5.1 Modelling power

At first glance, the addition of integer requirements for the variables does not seem like such a big change. However, many problems are **inherently** integer:

- *Chair production, or any type of production of integer amounts* - It is hard to sell fractional chairs.
- *Vehicle routing, e.g. trucks, ships, trains* - a fractional ship cannot sail.
- *Manpower planning* - planning with fractional workers is not a good option.
- *Timetabling* - the classes could be divided up, but what about the teacher?

These are just a few examples. It turns out that there are many cases where one or more variables needs to be integer ($x \in Z$) or binary ($x \in \{0, 1\}$).

Generating a good integer solution from the Incredible chairs model is not complicated. The optimal value for Incredible chairs is 51 with the values $x_A = 5.25$ and $x_B = 5$. Because all the coefficients of x_A are positive in the constraints and all the constraints are less-equal $\leq$, x_A can simply be rounded down to $x_A = 5$, achieving an integer solution. This reduce the objective value to 50. But is this solution optimal? In the Incredible Chairs 3 assignment this is the topic.

In general, just rounding fractional variables is **not** guaranteed to reach an optimal solution, perhaps not even a feasible solution. In the Incredible Chairs case, the integer solution objective went from 51 to 50. In other cases, the changes in the objective can be far greater: Suppose a container ship has to be routed to pick up containers. If 3 containers have to be picked up from a port by a container ship with a capacity of 1000, an LP model will assign a $\frac{3}{1000}$ ship to do the job. If what is minimized is cost, the objective can increase substantially by increasing the value from $\frac{3}{1000}$ to 1.

An important special case of integer variables is **binary** variables i.e. $x \in \{0, 1\}$. This can model yes/no decisions, usually interpreting 1 as yes and 0 as no. A set containing C binary variable can easily be defined in Julia/JuMP as:

```
@variable(IC,x[c=1:C],Bin)
```

where IC is the name of the mathematical model. Notice that since a binary variable can **only** take the values 0 or 1, adding lower and upper bounds is not necessary. For integer variables however, upper bounds can be added in exactly the same way as for continuous variables.

```
@variable(IC,x[c=1:C],Int)
@variable(IC,x[c=1:C] >= 0,Int)
@variable(IC,0 <= x[c=1:C] <= 10,Int)
```

5.1.1 Integer modelling tricks

There are a number of standard modelling tricks which can be utilized when modelling with integer or binary decision variables. Here we briefly mention the most important constructions, and point to the assignments where they are needed.

Fixed charge link: Often a resource like a ship, depot, truck or plant, is represented by a binary variable (is it needed or not) or how many do we need. The cost of using the resource is then 0 if it is not used, but even the most limited use incurs a fixed cost. Mathematically speaking, this can be stated as:

$$h(x) = \begin{cases} 0, & x = 0, \\ f + px, & x > 0. \end{cases}$$

If we assume that there is an upperbound on $x \leq B$, then this can be modelled using binary decision variables in the following way:

$$\begin{aligned} h(x) &= fy + px, \\ x &\leq By, \\ y &\in \{0, 1\}. \end{aligned}$$

In Julia this would look like this:

```
@variable(IC,x>=0)
@variable(IC,y,Bin)
@variable(IC,hx>=0)

@constraint(IC, x <= B * y)
@constraint(IC, hx == f * y + p * x) # the resulting function
```

Such a fixed charge link could, e.g., correspond to the cost of transporting x containers on a container ship: There is an additional cost, due to extra fuel needed for the extra weight of the container, but the main cost is probably due to the need for a full ship, i.e. the y cost. This type of modelling is needed in Chair Logistics 2.

If the resource can be increased in fixed amounts, then the only change needed is to change the definition of the resource variable from binary to integer. This could correspond to, e.g., a number of container ships.

Logic on binary variables
Often binary variables can be used to formulate logical relations between different binary variables. Usually, a binary variable $x = 1$ corresponds to true and $x = 0$ corresponds to false. With these definitions, we can easily define the various logic operators:

OR: Given two binary variables x and y, define the v binary variable as $x\ OR\ y$:

$$\begin{aligned} v &\geq x \\ v &\geq y \\ v &\leq x + y \end{aligned}$$

AND: Given two binary variables x and y, define the v binary variable as $x\ AND\ y$:

$$\begin{aligned} v &\geq x + y - 1 \\ v &\leq x \\ v &\leq y \end{aligned}$$

NOT: Given one binary variable x, define the v binary variable as $NOT\ x$:

$$v = 1 - x$$

Force selection: Given a set of i items, select at least 3 of these:

$$\sum_i x_i \geq 3$$

Limit selection: Given a set of i items, select at most 4 of these:

$$\sum_i x_i \leq 4$$

A classical mistake in modelling is to try to model logic by using JuMP variables as Julia variables, as discussed in Section 3.8.4. Since this is not possible, when logic in the model needs to modelled, it can be modeled as above. This is a requirement in the Startup Fund assignment 6.4.

Binary multiplication: All MIP model problems in this book can be modeled using **linear** models, i.e. where the variables are not multiplied. Given two binary variables x and y the multiplication of these can be formulated in a linear fashion, where the binary variable z is the result: (exactly as the x AND y operator)

$$
\begin{aligned}
z &\leq x \\
z &\leq y \\
z &\geq x + y - 1
\end{aligned}
$$

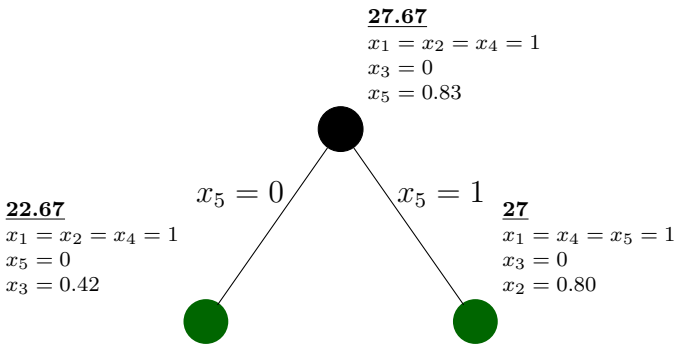
5.2 Solution algorithm: Branch & Bound

The focus in this book is on **modelling** LP and MIP models, not on the optimization algorithms needed to find the optimal solution to the models. In Chapter 8, we give references for further study, if you want to dive into this. It is, however, beneficial to have some intuition about the classical Branch & Bound algorithm for solving MIP models in order to better model different problems. In this section, we give a very simple and intuitive description of the basic approach.

The description of the algorithm is going to consider a very simple, small MIP model, the knapsack problem, see below:

$$
\begin{aligned}
&\texttt{Maximize.} && 7 \cdot x_1 + 10 \cdot x_2 + 4 \cdot x_3 + 4 \cdot x_4 + 8 \cdot x_5 && \\
&\texttt{Subject to :} && && \\
& && 3 \cdot x_1 + 5 \cdot x_2 + 12 \cdot x_3 + 2 \cdot x_4 + 6 \cdot x_5 && \leq \quad 15 \\
& && x_1, x_2, x_3, x_4, x_5 \in \{0, 1\} &&
\end{aligned}
$$

The knapsack problem contains 5 binary variables. In the Branch & Bound algorithm (B&B), first the LP version of the above problem, where the variables are **relaxed**, i.e. $x_1, x_2, x_3, x_4, x_5 \in [0, 1]$, is solved. This problem can easily be solved using an LP solver, reaching the optimal value 27.67, with the values: $x_1 = x_2 = x_4 = 1$ and $x_3 = 0$ and $x_5 = 0.83$. This solution is almost an integer solution, and the value 27.67 is a valid upper bound, i.e. we now know that **no** integer solutions with higher objective values exist. The B&B algorithm then selects the branching variable, i.e., selects one of the variables that is fractional and forces it to be either 0 or 1 by considering two new problems, where each of the new problems has an additional constraint. In one of the problems, $x_5 = 0$, and in the other $x_5 = 1$. This leads to two new solutions, shown as branches in a binary tree.

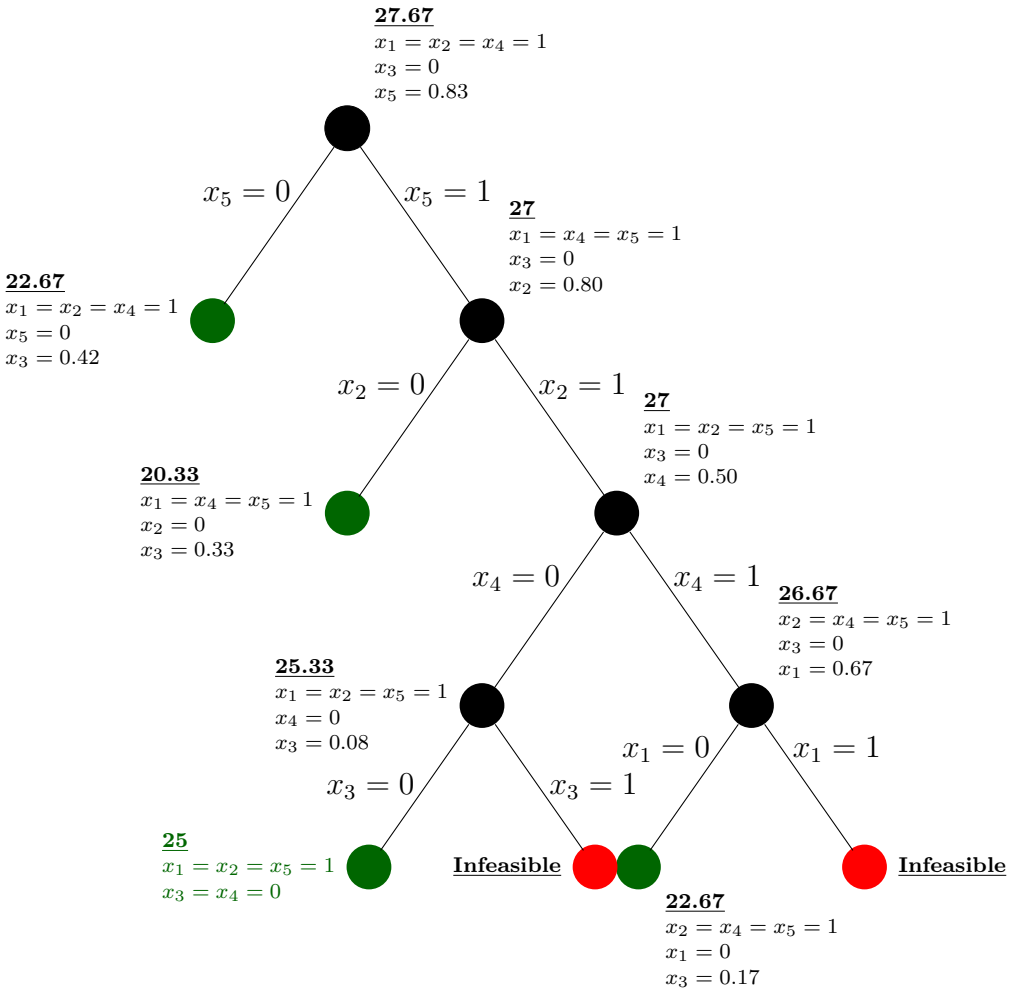


The black node is the so-called root solution. This is the LP relaxation we solved above and has an objective value of 27.67. The two new, green nodes reflect the impact of branching on variable x_5. The two problems are then solved giving solutions with objective values 22.67 ($x_5 = 0$) and 27 ($x_5 = 1$), respectively. The additional constraints are **restrictions** as mentioned in Section 3.6 and therefore the objective in both branches cannot have a higher value than 27.67. There are three possibilities for each of the new problems. These are:

- **The problem is infeasible:** It is not possible to find a (fractional or integer) solution, which is feasible. This means that any solution with more constraints will also be infeasible. Thus, we can **fathom** this node, i.e. not consider this node solution and any solution derived from it by branching. Fathoming is possible since it is guaranteed that no solution in the sub-tree of this node can improve the current best solution because all the solutions in the sub-tree will be infeasible.
- **The solution is integer:** This may be the optimal solution. If it is either the first solution found or better than the current best, then the solution is saved and termed the **incumbent**. It is the best integer solution found so far. All solutions in the sub-tree of this node can be ignored since they will be at best be as good as this solution.
- **The solution is fractional:** If the objective value is worse than the current incumbent, i.e., lower than the incumbent value for a maximization problem, then the node can be fathomed. Otherwise, a new fractional variable is selected and the process is repeated.

In this way, the Branch & Bound algorithm searches the binary search tree. If fathoming is not used and there are N binary variables, $2^{N+1} - 1$ nodes need to be searched, i.e. LP optimized. The fully searched tree is shown below. Notice that due to fathoming, only 11 nodes are investigated. The full tree would contain 63 nodes. One of these leads to an integer solution with an objective value of 25, the optimal solution. The red nodes

are infeasible, and the remaining non-integer green nodes have optimal values less than 25 and therfore do not need to be explored.



Modern MIP solvers have evolved tremendously over the last 40 years. The description here is only meant to provide intuition behind the procedure.

5.2.1 Making solvable models

How hard a MIP model is, depends on both the size and formulation of the MIP model. More specifically, there are two aspects which are very important for the solvability of the model:

The number of integer/binary variables: The height of the Branch & Bound tree depends directly on the number of integer/binary variables. Two things can be done:

- Some integer variables will take integer values, even if not forced to. Then it should be formulated as a continuous variable.
- Integer variables for which it is possible to deduce their value should be fixed to this value, e.g., $x_{1,5} = 1$. The easiest way to do this in Julia/JuMP is:

```
fix(x[1,5],1; force = true)
```

MIP Gap: The difference between the optimal value of the MIP model compared to the value of the optimal **relaxed** MIP model, where all the integer/binary variables have been converted to continuous variables. The size of this difference is critical. To find this gap, solve the first MIP model and compare to the optimal relaxed model which can relatively easily be found, using the following model:

```
relax_integrality(model)
optimize!(model)
```

If several different models are available for a problem, which often happens, we can compare the models and choose the best model, wich is often the model with the fewest integer/binary variables and/or the lowest gap.

5.3 Solution hardness

The Branch & Bound algorithm for solving MIP models was invented by Ailsa H. Land and Alison G. Doig in 1960 Land and Doig (1960). Today MIP models are one of the most important tools applied in Operations Research. Importantly however, solving them is much harder than solving LP models. LP models can be solved with millions of variables; however, even MIP models with fewer than 100 variables may not be solvable to proven optimality. There are deep theoretical reasons for this difference, which we will briefly discuss in Chapter 8. However, since the invention of the Branch & Bound algorithm, researchers have continued to improve the approach and in particular in the last 30 years, the size of the MIP models which can be solved to optimality have improved tremendously. This has gradually enabled the use of MIP models to solve large real-world problems. A behaviour which is observed for all MIP models is that the solution time will grow **exponentially** with the size of MIP model, i.e. with the number of integer variables in the MIP model. In Figure 5.1 below, a MIP model for different

sized Travelling Salesman Problems (TSPs) is solved to optimality. The model of TSP tested is the model from the Grocery problem 6.13. Five different solvers are considered: HiGHS (light blue), CPLEX (grey), Gurobi (yellow), Cbc (dark blue) and GLPK (red) and they are all tested on 20 different random TSP problems for each size. All solvers show the same behaviour: Smaller models can be solved fast but when the size increase, at a certain **bending point**, the solution time suddenly increases dramatically. This kind of bending point exists for all MIP models and for all solvers, but as can be seen in Figure 5.1, it happens first for the GLPK solver, then the HiGHS solver, then the Cbc solver, then the CPLEX solver and finally the Gurobi solver. However, given a MIP model, if the model is of sufficiently small size, the solvers can handle the problem.

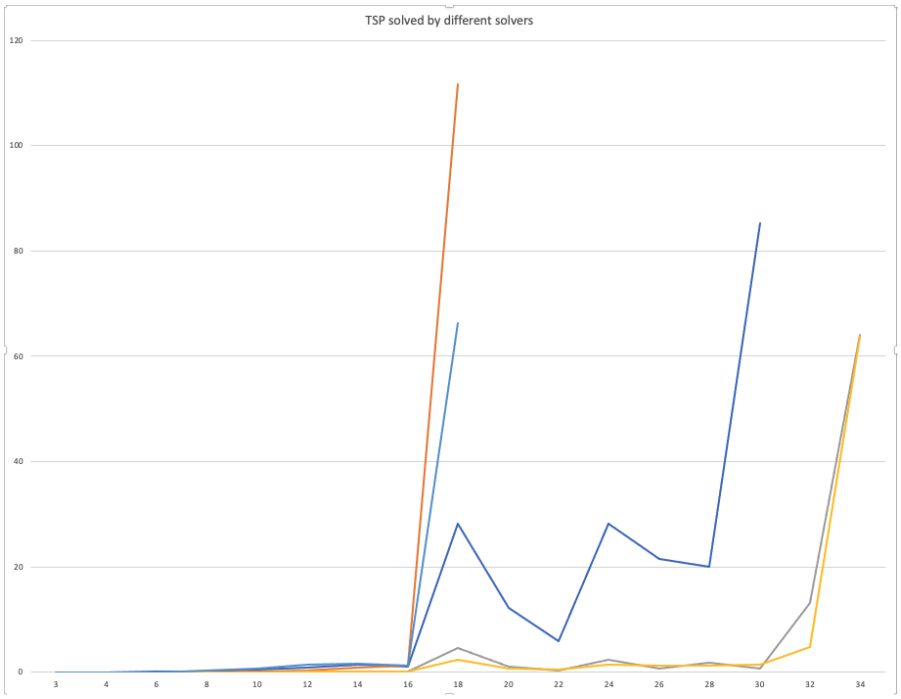


Figure 5.1: Average solution time of TSP with different solvers

Looking at the above graph, you may be wondering why we choose to prefer the HiGHS solver in this book. The answer is that in the future, the HiGHS solver is going to be further developed whereas the development of the Cbc solver seems stalled. Furthermore, the printout from the HiGHS solver is much better.

6 Mixed Integer Programming problems

This chapter contains MIP exercises. All of the assignments require the use of discrete decision variables and thus the formulation of MIP models. The problems considered appear in increasing order of difficulty.

6.1 Incredible Chairs 3

We begin with a simple problem.

Assignment 1.1

Reformulate the LP model from the Incredible Chairs assignment 4.1.1 so that it becomes a MIP model in which only discrete quantities of chairs are considered.

Optimal objective value for Incredible Chairs 3 problem, truck mix: 50

6.2 Chair Logistics

The Chair Distribution job has been a success, **but** the distribution plan only decides how many chairs to transport between plants, depots and retailers. However, this plan does not consider the fact that the chairs are transported by trucks, and trucks have a limited capacity.

Incredible Chairs can lease a number of trucks with a capacity of 40 chairs. We assume that the trucks can transport from plants to depots, from plants retailers and from depots to retailers. The company has calculated that the cost per kilometer for the trucks is €1.5 per kilometer per truck.

Assignment 2.1

Formulate an MIP to find the cheapest trucking cost.

Optimal objective value for Chair Logistics Problem, truck mix: €71494.5

What is the cost of the fact that we do not have trucks which can transport just one chair?

Hint: This assignment is relatively easy to solve, extending the previous Chair Distribution assignment, but now adding integer-truck variables, and optimizing based on these.

6.3 Class Jobs 2

The teacher in charge of the community outreach realizes that you only have limited time to do the jobs, i.e. at most 3 hours. She decides to estimate how long time it takes for each child to do each job, and create the following matrix $TimeRequired_{j,c} \in Z+$:

	c_1	c_2	c_3	c_4	c_5
j_1	1	2	1	4	4
j_2	6	2	4	2	2
j_3	3	3	2	4	4
j_4	1	1	4	4	2
j_5	7	2	2	3	1

Update the previous Class Jobs 2 model, to include the limitation that all jobs must be finished in at most 3 hours.

Assignment 3.1

Solve the updated Class Jobs 2 model, **as an LP model** and answer: What is the value of the optimal solution, **and** is it a feasible solution?

Optimal LP solution for Class Jobs 2: 19.92

Change the variables to binary variables, and switch to the Cbc solver. Solve the problem again. What is the optimal value, and is the solution now feasible?

Assignment 3.2

Solve the updated Class Jobs 2 model, **as an MIP model** and answer: What is the value of the optimal solution, **and** is it a feasible solution?

Optimal MIP solution for Class Jobs 2: 18

6.4 Startup Fund

A startup fund manager decides to use MIP models in order to optimize their profit. A new investment fund is just starting up, and €100 million is available for investment. The startup fund staff have identified nine possible investment opportunities in startup companies, the core information being the required investment and the expected value increase in 3 years.

	Investment								
	1	2	3	4	5	6	7	8	9
Estimated profit factor	1.7	1.4	1.3	2.1	1.9	1.8	1.5	2.2	1.6
Capital required	17	25	19	25	28	23	29	31	18

Table 6.1: Investment details (millions of dollars)

Given an available budget of €100 million, the objective for the fund is to select the best possible set of investments. Since these investment opportunities are in startup companies, the full amount of capital needed for an investment must be invested (and it is not possible to invest more). Furthermore, money which is not invested is considered lost.

Assignment 4.1

Formulate a binary integer program that can be used to solve this problem and implement it in Julia/JuMP.

Optimal objective value, profit factor, for Investment Optimization Problem: 7.8

Assignment 4.2

After creating the first investment plan, the manager decides that investments in 1 and 4 are competing companies, so it is not possible to invest in both companies and the same applies for investments in 6 and 8. Furthermore, the investments in 6 and 9 can only be performed if either investment 2 or investment 3 is done, since the developed technology in these is the basis of 6 and 9.

Optimal objective values for Investment Optimization Problem: 7.3

6.5 Stamps

Your uncle Archibald, whom you have not seen for 20 years, dies, and in his will, to your big amazement, let you inherit his stamp collection. Your only problem is that you have absolutely no interest nor knowledge about stamps. Some of the countries from where the stamps originate, you have not even heard of!

Initially you consider whether to simply scrap the entire stamp collection of 100 stamps. Before you do so, you do, however, make a quick internet picture search for the stamps, and suddenly realize that at least some of the stamps are rather valuable! Therefore you decide to make some money, but now the problem is that you have zero knowledge about stamps. You consider to contact the local stamp collecting club but realize that your very limited knowledge makes it easy to trick you into a bad deal.

A friend comes up with a brilliant suggestion: Make an online auction! Since at least some of these stamps are rather valuable, it makes sense to do a little work. You and your friend make a simple website, where you put pictures of all of the 100 stamps, numbered from 1 to 100. You then send out emails to a number of stamp-collector clubs and ask for email-bids before a certain date.

A new problem then arises: Stamp collectors typically collect complete sets of stamps, e.g. all Danish stamps from 1977. Thus, if a stamp collector lacks two stamps to complete a collection, then he/she wants to buy *both* stamps or none of them. You receive 213 bids (in €) for different subsets of stamps from your set of 100 stamps. Each bidder wants *all* stamps in their bid at the price they offer. In other words, if they cannot get all of stamps in the bid, then they will not have any.

The data for the problem can be downloaded from the book web-site `https://www.man.dtu.dk/MathProgrammingWithJulia`. The data, in Julia format, are:

- $BidPrice_b$: The price offered in bid b in € for a set of stamps.
- $BidSets_{b,s}$: Incidence matrix, where $BidSets_{b,s} = 1$, if bid b includes stamp s and 0 otherwise.

You can then either simply copy-paste the data into your program or you can use the `include(<file name>)` command.

Assignment 5.1

Formulate the problem as an integer program and solve it using Julia/JuMP.

Optimal objective value for the stamp problem: €11084

You are very happy about the earnings but, to your surprise, not all stamps are sold! You discuss this with a friend, and he considers this a stupid loss, and blames the outcome on poor optimization. Hence, he suggests to *require* that all stamps are sold.

Assignment 5.2

Make the new model, what happens? Why? And is the suggestion good?

6.6 Scrap Removal

After cleaning up in the shed (Project scheduling), your father has identified a number of scrap items, which he needs to get rid of. There are 20 items of different weight. Your father contracts a haulage contractor, to come and pick up big-bags of scrap. The contractor is paid €50 for each big-bag he has to transport away. Each big-bag can carry at most 100 kg of scrap and your father has ordered 10 big-bags. He makes the following deal with you: If you pack the items in big-bags, not breaking the 100 kg limit, and are able to use fewer big-bags, then **you** will get the €50 for each big-bag saved. How much money can you make?

	Items									
	1	2	3	4	5	6	7	8	9	10
weight (kg)	35	10	45	53	37	22	26	38	63	17

	Items									
	11	12	13	14	15	16	17	18	19	20
weight (kg)	44	54	62	42	39	51	24	52	46	29

Table 6.2: Weight for each of the 20 items

Assignment 6.1

Implement a MIP model in Julia/JuMP which minimizes payment for the contractor.

Optimal objective for Scrap Removal: €400

6.7 Food Festival

A close friend of your mother is in charge of a food festival in your town, i.e. a number of stands in tents around in the city, where food can be presented and tasted by the festival participants. There is however a planning problem at the festival: During the late evening, night and early morning, security guards are needed to watch the different facilities around the city. There is a list of 25 work shifts which needs to be assigned a security guard. However, some of the shifts cannot be covered by the same security guard because of overlap in time. Below in Table 6.3 is a list of the 25 shifts and for each shift the list of shifts which cannot be done by the same person. If a person is working shift 1 that person cannot work shift 2, shift 4, shift 6, shift 11, shift 17, shift 21, shift 22, and shift 25. Notice, that it goes both ways so a person working shift 25 cannot also work shift 1.

The concern is how many persons it is necessary to hire for the shifts. It is easy to hire 25 persons and then let every person work exactly one shift. This will then not lead to any conflicts. Even though they will all receive the same hourly wage, there are a number of expenses associated with hiring the persons, therefore she wants to hire as few persons for the shifts as possible.

You can find these data on the book web-site
`https://www.man.dtu.dk/MathProgrammingWithJulia`.The data contains:

- $s \in Shifts = \{1, 2, 3, ..., 23, 24, 25\}$
- $Conflict_{s1,s2}$: An incidence matrix such that if $Conflict_{s1,s2} = 1$ then shift $s1$ and shift $s2$ conflicts and cannot be worked by the same security worker.

Assignment 7.1

Formulate a model which minimizes the number of persons it is needed to hire.

Optimal objective value for the Food Festival problem: 5

Shift	Conflicting shifts									
1	2	4	6	11	17	21	22	25		
2	3	7	9	14	16	17	23			
3	4	9	10	11	18	19	22	23		
4	8	9	10	19	20	21	24	25		
5	6	10	11	13	15	16	21	22		
6	8	11	12	13	14	15	16	17	18	19
7	9	10	14	15	21	22	25			
8	9	10	11	15	16	24				
9	12	13	15	19	21	24				
10	11	14	15	16	19	22				
11	12	13	15	19	20	21	25			
12	14	15	16	17	18	23				
13	14	16	19	23	25					
14	15	17	18	22						
15	16	17	23	24	25					
16	18	19	20	21	24					
17	19	22	24							
18	20	21								
19	23	24	25							
20	22	25								
21	22	23	24							
22	24									
23	24	25								
24	25									

Table 6.3: List of shifts and conflicting shifts

6.8 Micro brewery 2

Encouraged by your good results as a beer producer, you realize that if you offer a broader selection of beers, you may be able to sell more beers. So after some trial production, you decide to start the production of three types of beer: TSP-Stout, Knapsack-Dark, and Set-Partitioning-Light. After some tastings, the cafes order different amounts of beer. The details are given in Table 6.4:

	TSP-Stout	Knapsack-Dark	Set-Partitioning-Light
jan	35	15	5
feb	20	10	20
mar	15	20	20
apr	45	15	35
may	25	15	35
jun	65	55	80
jul	40	90	60
aug	50	80	30
sep	35	25	35
oct	85	45	20
nov	50	5	20
dec	55	30	40

Table 6.4: Beer demand (litres)

You have the same brewing system, with a capacity of 120 litres, but you can only brew **one** type of beer per month! Since your mother wants to encourage your business, she allows you to store 300 litres, for the beer types combined (i.e., the sum of stored beer for all tree types can at most be 300 per month). As in the previous assignment, she still charges €1 per 10 litres per month, for all beer types. Furthermore, you have produced an initial storage, 25 litres of TSP-Stout, 65 litres of Knapsack-Dark, and 75 litres of Set-Partitioning-Light.

Assignment 8.1

Make a Mixed Integer Linear Programming model, which plans the production and storage optimally, minimizing the storage costs, and take into account that only one type of beer can be produced per month. Implement the Microbrewery model in Julia/JuMP.

Optimal objective value for Micro brewery 2 Problem: 192.5

6.9 Chair Logistics 2

Incredible Chairs now considers if it is possible reduce the transport cost by using **other** depots. Changing the factories is very expensive, but a depot is relatively inexpensive to create and may reduce the transportation costs. Two new **possible** depots D_5, capacity 3750, and D_6, capacity 3250, can be established at a cost of €5000 per site, and the four existing depots can be closed down, each saving €5000. The distances (in km) from the plants to the two new depots are given in Table 6.5. Similarly, the distances (also in km) from the two new depots to the six retailers are given in Table 6.6. The capacity of depot D_5 is 3750 and the capacity of depot D_6 is 3250.

	D_5	D_6
P_1	88	109
P_2	128	105

Table 6.5: Distances between the plants and the new depots (km)

	R_1	R_2	R_3	R_4	R_5	R_6
D_5	223	172	62	80	13	124
D_6	185	135	31	113	54	96

Table 6.6: Distances between the new depots and the retailers (km)

Assignment 9.1

Formulate an MIP to find the cheapest distribution costs when using trucks and allowing the possibility of opening and closing depots.

Optimal objective value for Chair Logistics 2 problem, truck mix: €70780.5

6.10 Student Teacher Meetings

In Danish schools, it is customary to have meetings during the school year, often twice a year, between the student and parents and some of the teachers of the student. The meetings take place in the evening. Before the student-teacher meeting evening, the student (or parents!) make a list of the teachers they would like to meet with. When all students have submitted their teacher meeting request, a schedule needs to be made.

The teachers each use their own classroom for their meetings. The students and their parents walk around the school and meet with the teachers they wish to talk to. A number of time slots are given, e.g., a 10 minute interval, and your job is simply to plan when, i.e. in which time slot, the meetings will occur.

You can find data for the assignment on the book web-site `https://www.man.dtu.dk/MathProgrammingWithJulia`. Six different problem instances are available, where StudentTeacherMeetingData_15_6.jl have 15 students and 6 teachers:

```
StudentTeacherMeetingData_15_6.jl
StudentTeacherMeetingData_20_6.jl
StudentTeacherMeetingData_25_10.jl
StudentTeacherMeetingData_25_8.jl
StudentTeacherMeetingData_30_10.jl
StudentTeacherMeetingData_40_10.jl
```

In each file, the following information is given:

- $StudentTeacherMeetings_{s,t}$: An incidence matrix, with one row for each student s and one column for each teacher t. If student s should meet teacher t, then $StudentTeacherMeetings_{s,t} = 1$, otherwise $StudentTeacherMeetings_{s,t} = 0$
- S: Number of students
- T: Number of teachers
- TS: Number of time slots

The meetings are private, so a teacher can only meet one family at a time. The families would like to spend as little time as necessary on the meetings. Therefore, the overall objective is to minimize the sum of the latest time slot indices assigned to the families.

The problem is surprisingly difficult, so the data sets are all rather small and open-source solvers all struggle on even the smallest instances. This problem thus shows the value

of the expensive commercial solvers CPLEX and Gurobi.

As described Section 3.7, it is possible to set a maximal solve time using

```
set_time_limit_sec(model, 1800) # timelimit of 1800 sec.
```

It is, however, very likely that the solver will not find the optimal solution and will terminate with a solution that is not optimal, or even without a solution.

Assignment 10.1

Formulate the student-teacher meeting planning problem as a MIP, minimizing the sum of late time slots. Implement your model in your Julia/JuMP.

Only the optimal objective values for the first five problems are reported (together with the Gurobi solution time). These results were obtained using a Mac Book Pro, 2018.

Optimal objective values for the student-teacher meeting problems

- StudentTeacherMeetingData_15_6.jl : 66 (sec. 52.9)
- StudentTeacherMeetingData_20_6.jl : 110 (sec. 632.6)
- StudentTeacherMeetingData_25_8.jl : 131 (sec. 1393.3)
- StudentTeacherMeetingData_25_10.jl : 113 (sec. 1345.4)
- StudentTeacherMeetingData_30_10.jl : 152 (sec. 603.7)
- StudentTeacherMeetingData_40_10.jl : 248 ($\geq$ 244, 1.61% gap) (sec. 929.7)

6.11 Workplan for Teaching Assistants

A Mathematical Programming crash course, over 9 days, is taught by 4 teaching assistants (TAs) and 2 professors. The professors give the lectures and the TAs are available for help with the exercises. Suppose that four TAs (AL, FR, JE, and MI) are the TAs in the course. In this assignment, you have to plan when they should work during the course. Each TA must work exactly 52 hours (a total of 208 hours are available for the four TAs). The professors have attempted to come up with a plan of how many TAs to assign for exercise help in each of the 8 time slots from 9 to 17. This TA demand is given below in Table 6.7.

	Day								
	1	2	3	4	5	6	7	8	9
9-10	4	4	4	0	3	0	3	0	2
10-11	4	4	4	2	4	2	4	2	2
11-12	4	4	4	4	4	4	4	4	2
12-13	4	3	3	3	3	4	3	3	1
13-14	4	3	3	3	3	4	3	3	1
14-15	4	3	3	3	3	4	3	3	1
15-16	3	3	2	3	3	4	3	3	1
16-17	3	3	2	2	2	3	2	2	1

Table 6.7: TA demand for the first 9 days

To make the work easier for the TAs, we wish to take into account their personal preferences. Each TA has specified inconvenience scores for each day d and time period t in the matrix of inconvenience scores $Inconvenience^{ta}_{t,d}$. You can find these data on the book web-site `https://www.man.dtu.dk/MathProgrammingWithJulia`. You can then either simply copy-paste the data into your program or you can use the `include(<file name>)` command. Notice that the data also contains a simple normalization of the Inconvenience matrixes.

Assignment 11.1

Formulate a mathematical model that can be used to find the optimal work plan for the TAs. Implement your model using Julia/JuMP to find the optimal solution.

Optimal solution TA work plan (minimal inconvenience): 94.97

When you inspect the generated plan, you observe that the TAs work several times on each day. This is rather inconvenient. You decide to modify your model and include the restrictions that each TA can only work consecutive hours and that if a TA works on a particular day, then the TA has to work at least two hours.

Assignment 11.2

Revise the mathematical model with additional restrictions. Implement this model using Julia/JuMP to obtain the optimal plan.

Optimal solution TA workplan (minimal inconvenience): 121.97

Notice that while these extra requirements are very reasonable, it leads to a worse solution, regarding the TA in-convenience. On the other hand, the revised solution is an improvement for the TAs in a different way.

6.12 Tennis

If you play tennis on a clay tennis court, then you must prepare it for the next couple of players. This is done by first brushing the clay and then sweeping the lines, using a special broom, see Figure 6.1.

Figure 6.1: Sweeping a tennis line

This assignment deals with the problem of finding the easiest (shortest) way of sweeping the lines (of one side) on the tennis court. In Figure 6.2, the outline of a tennis court is given, with all measurements in feet.

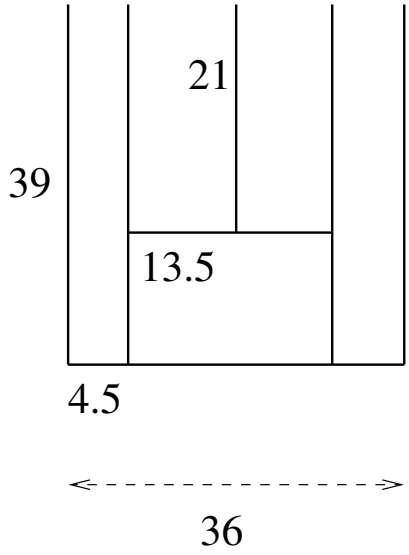


Figure 6.2: Court measurements (feet)

The objective is to minimize the walking distance when sweeping the lines. When solving this problem, different formulations are possible.

Hint: Define a coordinate system and let all of the places where lines start or connect to other lines be points in the coordinate system. There are 12 such points. Then, calculate a distance matrix, which can easily be calculated in a couple of for loops.

Assignment 12.1

If you are *only* allowed to step on the lines, then what is the best route you can walk, if you have to return to the starting position? This can be calculated analytically.

Optimal distance: 390 feet

Assignment 12.2

Construct a mathematical model to find the best route if you are allowed to walk between the lines, but still have to start and stop at the same points. Implement your model in Julia/JuMP.

Optimal distance: 306 feet

Assignment 12.3

Relax the model such that it is allowed to start and stop at different places. What is the best route now?

Optimal distance: 280.5 feet

Assignment 12.4

Constrain the model such that the start and stop places are only allowed on the outside lines (the 3 outside lines being the leftmost line, the bottommost line and the rightmost line in Figure 6.2. What is the best route?

Optimal distance: 287.47 feet.

6.13 Groceries

The local grocer in your neighbourhood decides to offer delivery of goods to home and offers you the job of driving around with the deliveries. The job is simple: You go to the grocery shop 16.00, get the list of persons to deliver to and the bags of groceries in. You then make a round-trip in the grocery van, visiting all the customers, delivering their goods. When you return the van to the grocery shop, your work is over. You are paid a fixed amount each day, so the faster you can deliver the goods, the faster your job is over.

Since you want to work as little as possible, you want to plan the shortest possible route. Given 5 customers, with the coordinates below in Table 6.9, where the grocery is in origo (0,0). We will for simplicity assume that direct travel between the different points is possible, and simply use euclidean distance. How can you formulate a Mixed Integer Programming model to minimize the total round-trip tour length ?

Customer	X coordinate	Y coordinate
Grocery	0	0
1	104	19
2	370	305
3	651	221
4	112	121
5	134	515

Table 6.8: Grocery and customer coordinates

You quickly realize that any round trip will consist of a number of different decisions, i.e. whether to travel from one customer to another or not. Each such decision can be defined using a binary decision variable $x_{i,j} \in \{0, 1\}$, which is 1 if you travel from customer i to customer j. This makes it simple to calculate the total travelled distance: $\sum_i \sum_j Dist_{i,j} \cdot x_{i,j}$. This is obviously the objective function.

However, how can you define constraints so that only solution values of $x_{i,j}$ which represents correct round trips are feasible ? First you realize that any round-trip tour, means that you travel exactly once from one of the other customers j **to** a customer i and **from** a customer i to one of the other customers j. These two requirements can be implemented with two abstract constraints. It is also necessary to fix the variables $x_{i,i}$ to zero, i.e. going from city i to city i is not really interesting, so the value of these variables needs to be fixed to zero.

Assignment 13.1

Formulate the first model with the objective function to be minimized and the two different abstract constraints, ensuring that you travel to each of the customers and leave each of the customers.

Optimal objective value for the grocery problem: €1576.85

When you print out the values of the binary $x_{i,j}$ variables, you do, however quickly realize that this is not a feasible solution, since it consists of two separate round-trips. But the solution **is** legal according to your model. Therefore, you need one or more additional constraints, which will make separate round-trips impossible. How to do that ?

One of your friends suggest the following: Assign a counter (a decision variable) $u_i >= 0$ to each customer. This can simply be a positive continuous variable. Then create a constraint, which ensures that whenever you go from one customer to the next, the value of the counter is at least one unit larger than the customer you come from. At first this seems easy: $x_{i,j} + u_i \leq u_j \forall i, j$. This constraint does really do the job, when $x_{i,j} = 1$, but it also introduces a lot of relations between customer i and customer j when $x_{i,j} = 0$, and that is wrong. So the constraint needs to be altered such that it does not have any influence on the solution, unless $x_{i,j} = 1$. Therefore, you make a little trick. Let C be the number of customers (and the Grocer) in the model. Then re-write the constraint to: $u_i + 1 \leq u_j + C \cdot (1 - x_{i,j}) \forall i, j$. Notice, that if $x_{i,j} = 1$, we get exactly the previous formula, but if $x_{i,j} = 0$ we add a large number on the left hand side, to allow any value of u_j.

However, if you simply add this constraint, no solutions will exist, because for each tour, **one** starting point is necessary, to go from a high count to 0. But, this is exactly the point: If you only allow not to update the counter for one customer, e.g. the grocery, only one tour will exist. Hence, the constraint should hold for all pairs of customers, **except** one, e.g. the first one.

Assignment 13.2

Formulate the second model where you extend the previous model with counting variables and constraints which forces the counter variable to be one unit higher than the previous customer on the tour.

Optimal objective value for the grocery problem: €1857.51

After a one month trial period, the grocer opens up for taking more customers, and it is a great success, now there are suddenly 13 customers, coordinates given below, in combination with the size of the delivery size. The capacity of the van is 26.

Customer	X coordinate	Y coordinate	grocery size
1	0	0	0
2	104	19	3
3	370	305	9
4	651	221	7
5	112	121	11
6	134	515	11
7	797	424	6
8	347	444	7
9	756	141	7
10	304	351	2
11	236	775	4
12	687	310	2
13	452	57	8

Table 6.9: Grocery and customer, coordinates, and delivery size

This, however presents a new problem: There is simply not room in the small van for all the groceries. Actually 3 vans of this size are required. So the grocer decides to acquire 2 more vans of the same size and hire 2 more delivery workers. But now you have a problem: Your nice model only works with **one** van.

The model needs to be updated to accommodate 3 vans. Since all the vans start going out of the grocery parking lot, the sum over the $x_{1,j}$ variables, from the grocery the customers should be equal to 3, and 3 should be going into the grocery node $x_{j,1}$. Instead of the counting variable counting units, we now count how much material has been delivered at this point of the route, i.e. $u_i + q_i \leq u_j + Q \cdot (1 - x_{i,j})$, where Q is the capacity of the van, here 29. this problem may take significant time using the open-source solver HiGHS.

Assignment 13.3

Formulate the third model where you extend the previous model to handle 3 vans.

Optimal objective value for the grocery problem: €5063.35

6.14 Beer Flow Routing

One of the bars you supply beer to, assignment 6.8, is itself a micro-brewery with its own bar. This micro-brewery is significantly larger than yours and delivers beer to five other bars together with another micro-brewery, which also has its own bar. The point is that each of the two larger micro-breweries make different beers, and the 7 bars sells both types of beer, and others. The two owners of the micro-breweries have realised that their employees spend a significant amount of time every day transporting crates of beer to the five other bars, and to the other micro-brewery/bar. The owners have therefore come up with an interesting beer distribution idea: A beer pipe network! Their idea is to build a network of pipes that will enable beer flow from each of the two micro-breweries to the other bars/micro-brewery according to their demand. We will in this assignment only consider the beer flow, not production and storage of beer, and assume that both of the micro-breweries have large beer storage tanks and can always deliver the needed quantities.

Because all the bars are in the town center, the building costs are not prohibitive. Furthermore, since all seven bars are well established and have existed for decades, even significant building costs are acceptable since the system can probably be used for many years. The positions of the bars, in x and y coordinates, are given in Table 6.10. In the remainder of this assignment, we will refer to Bars and MB-Bars. MB-Bars are the locations in the network where beer is injected into the network

	Coordinates	
Bars	x	y
MB-Bar A	3	10
MB-Bar B	9	10
Bar C	8	6
Bar D	2	5
Bar E	6	2
Bar F	10	2
Bar G	7	11

Table 6.10: Location coordinates

The owners of the micro-breweries have already planned the network of pipes that they will build between the different bars. The connections are specified Table 6.11. We will refer to the bars as nodes in the network and the pipes as edges.

Pipes
A-B
A-C
A-D
B-C
C-E
C-F
F-G

Table 6.11: Pipe connections

You decide to gradually build your optimization model. First, you implement a simple LP model that models the flow of one litre of beer from MB-Bar A to Bar G through the network suggested by the microbrewery owners. You define a beer-flow variable $x_{(i,j)}$, which models the flow of beer from Bar i to Bar j. For each node you can now make a balance constraint, see Section 3.5. Basically, for each Bar i in the network the constraint $\sum_j x_{(j,i)} = \sum_j x_{(i,j)}$ defines the flow, where the left-hand-side defines the flow from all the other Bars j **into** Bar i. And the right-hand-side defines the flow from Bar i **out to** all the other Bars j. Beer is injected into the network at MB-Bar A and exits the network at Bar G, this needs to be added to the constraint. The objective, to keep it simple, is to minimize the number litres of beer flowing in each pipe.

Assignment 14.1

Formulate a simple LP model that moves one litre of beer from node A to node G

Optimal objective value for the beer flow problem: 3

The next step is to extend the simple model to include delivery of beer from MB-Bar A to all the other bars. Table 6.12 specifies that MB-Bar A injects six litres of beer into the network and that each of the other bars in the network receives one litre of beer. Let us assume that the capacity of a pipe is 10 litres. The objective is now to minimize the distance traveled by the beer.

Assignment 14.2

Formulate an LP model that minimizes the beer litre distance needed to supply beer from MB-Bar A to all the other bars.

	Beer needed from MB-Bar A
MB-Bar A	- 6
MB-Bar B	1
Bar C	1
Bar D	1
Bar E	1
Bar F	1
Bar G	1

Table 6.12: MB-Bar A beer flow

Optimal objective value for the beer flow problem: 55.22

After succeeding with the two previous models, now is the time to minimize the beer distance flow from **both** MB-Bars to all the other Bars. The demand is now given as a matrix. There is a row for each supply point in the network, the MB-Bars, and a column for each demand point. An element of the matrix indicates how much beer must be moved between each supply and demand point pair. As an example, MB-Bar A must deliver 20 litres of beer to Bar F.

	A	B	C	D	E	F	G
MB-Bar A	0	70	80	20	80	20	40
MB-Bar B	50	0	90	50	10	100	50

Table 6.13: Demand matrix (litres)

The pipe capacity of 10 litres is naturally too small for this demand. The bar owners plan to use the pipe system during the day, so even thin pipes can be used. They estimate that each pipe can flow 500 litres during the day.

Assignment 14.3

Formulate an LP model that minimizes summed pipe distance needed to meet the demand stated in Table 6.13

Optimal objective value for the beer flow problem: 5616.94

Hint: To formulate this assignment, you need a four-dimensional variable: $x_{(k,l),(i,j)}$. This states the amount of beer with origin MB-Bar k and destination Bar l moving

between Bar i and Bar j.

Proud of your result, you present it to the two micro-brewery owners, but they instantly point out two critical weaknesses:

- A pipe cannot flow beer in both directions at the same time.
- The same pipe cannot be used for different beer types. The beers get mixed and ruined.

You have to agree on these considerations, but then counter and suggest that the network they have designed may not necessarily be the best network. The micro-brewery owners then ask you what type of network you would suggest. You are to design the network such that only flow of the same type of beer in a pipe is acceptable. But you can put two pipes between a pair of bars, one for each type of beer.

Assignment 14.4

Formulate an MIP model that minimizes the total pipe distance necessary to support a beer flow which satisfies the demand.

Optimal objective value for the beer pipe network construction problem: 50.67

Hint: Here you need binary variables, i.e. if a pipe should be established for flowing beer originating from bar k between bar i and bar j.

Hint2: Due to the number of necessary binary variables, this problem may take some time to solve using the open-source solver HiGHS.

You realize that you have just designed **two** separate networks, one for each type of beer, where the cost of the necessary pipes is minimized. However, the pipes are actually not that expensive, the main expense for the micro-brewery owners is the **digging work** in the streets between the bars. If an **edge** between two bars is established, i.e. dug, putting several pipes into the ground is relatively cheap. Hence the micro-brewery owners now want you to minimize the summed dug distance between the bars.

Assignment 14.5

Formulate an MIP model that minimizes the dug distance necessary to support the pipes in which a beer flow to satisfy the demand.

Optimal objective value for the beer pipe network construction problem: 25.33

6.15 The Wedding Planner

This assignment is about seating guests at tables, during a dinner. With data from a real wedding, G guests are to be seated at T tables. We want to find the best feasible seating-plan such that the guests at the same table share interests, have a somewhat equal distribution of men and women, and that each guest at least know some of the other guests at the table. We will assume that the tables are small, 9 seats, such that everybody at the table can talk to everybody. The feasibility requirements are:

- Each guest should sit at a table
- There is a limit of 9 people at each table
- Couples that attend the wedding should sit at the same table

You can find these data on the book web-site `https://www.man.dtu.dk/MathProgrammingWithJu`
You can then either simply copy-paste the data into your program or you can use the `include(<file name>)` command. The data contains the following parameters:

- Guests: The set of guests
- G: The number of guests
- Tables: The set of tables
- T: The number of tables
- SharedInterests: A two-dimensional matrix where SharedInterest[g1,g2] specify how many shared interests guest g1 and g2 has.
- Couple: An incidence matrix where Couple[g1,g2]=1 if g1 and g2 are a couple
- Male: Male[g]=1 if guest g is male
- Female: Female[g]=1 if guest g is female
- Know: Know[g1,g2]=1 if guest g1 and guest g2 know each other
- TableCap: No of guests which can maximally be seated at the table, equal to 1.

The final model is rather complex, so we will iteratively build the model in four stages.

Assignment 15.1

Formulate a mathematical model (without an objective) that can be used to solve this problem, i.e. just the constraints, no objective function needed.

Assignment 15.2

Add the following objective function to your model:

Maximize the number of shared interests for persons sitting at the same table

Optimal objective value: 67

Hint: Define a new variable that takes the value of one if two guests are seated at the same table. If you get a higher result, then check to see if you are counting people twice or counting self-shared interests.

Assignment 15.3

Now consider a different objective function. For each table, penalize the number of males and the number of females that are in excess of two. As an example, a table with seven males and two females has an excess of five males and incurs a penalty of three. Minimize the sum of penalties for all tables.

Optimal objective value for: 0

Hint: First formulate two sets of constraints that strictly enforce the requirements that there can be at most 2 too many of each gender. You can then introduce slack variables that allows these restrictions to be broken, but is penalized in the objective.

Assignment 15.4

Now consider a different objective function. You are to minimize the number of unfamiliar connections. The parameter *Know*, given in the datafile, defines which other guests each guests knows. The idea is that each person at the wedding should ideally sit at a table where that person knows at least 3 other people and if they do not know 3 at the table, the number less than three is the penalty.

Optimal objective value: 2

Hint: Make a hard constraint that forces everybody to know at least 3 persons that they sit together with. You can then relax this constraint with a slack variable, which is minimized in the objective function.

When the bride & groom consider the seating plan, they really consider all three objectives *at the same time*. Hence, we have here a *multi-objective optimization problem*. To solve the combined problem we need to evaluate how important each of the objectives is. We consider them to be equally important.

Assignment 15.5

Solve your model again, but this time minimize the sum of the negative number of interests, the number of non-equal male-female tables and the number of unfamiliar connections.

Optimal solution for Wedding Planner Problem: -60

The Wedding Planner problem is actually very very hard. The actual wedding where these data originates had 74 guests. But the MIP models cannot solve the full problem.

6.16 Hot Air

This assignment is designed to give you an appreciation of the complexity of a very important real-life problem involving integer variables. It focuses on a simplified version of an airline fleet assignment problem.

Hot Air is an airline company that is considering an aggressive move into new territory in Texas. They have forecasts for the demand, i.e. the number of passengers that wish to travel on Hot Air, for one city to another city. Hot Air now needs to decide which routes to fly, that is, assign aircraft to. Their objective, not unexpectedly, is to maximize profit. Profit is generated by transporting fare-paying passengers, while taking into account any costs incurred when operating aircraft. All the necessary data are given below.

To make the problem manageable, Hot Air has made a number of simplifying assumptions, some more important than others. These are listed below.

Simplifying assumptions:

1. No penalty is incurred for **not** transporting passengers that wish to travel; these passengers will just find another means of travel.

2. All passengers transported must be flown non-stop from their origin to their destination. No change of planes is allowed, nor is staying on a plane that travels via another city (*1-stop routes*). This assumption is relaxed in the last question.

3. Hot Air is seeking a daily schedule. They operate (like most airlines) the same schedule every day. A day is assumed to consist of 3 time periods, where planes can fly from any city to any other city: Dawn to mid-morning, mid-morning to afternoon, and afternoon to evening.

4. Due to restrictions on hangar lease and maintenance/service personnel (who work during the night), a specific number of planes must spend the night at each city. That number, of course, is the number of planes that depart each morning and return each evening.

5. Planes do not have to fly. A plane not flying does not incur any cost.

6. Planes are fungible (interchangeable): Hot Air essentially flies only one type of aircraft. Therefore, only the *number* of planes at any position at any time is important, not the identity of the planes.

7. More than one plane can fly a given route at the same time.

The data considers four Texas cities: Brownsville, Dallas, Austin, and El Paso. The aircraft type used by Hot Air is the Itty-Bitty-47, which holds a maximum of 120 passengers, and Hot Air has four of these planes. There is hangar capacity for two over-nighting planes in Brownsville and one each in Dallas and Austin. Three time periods per day are considered, each with its own demand for travel among the cities, see Table 6.14, Table 6.15 and Table 6.16. Ticket prices are given in Table 6.17. A ticket from Austin to Brownsville costs \$109. The costs (per flight), which are the same across time periods, are given in Table 6.18. For instance, it costs \$4400 to fly from Brownsville to Austin. This includes landing fees, crew costs, fuel etc.

Assignment 16.1

You have been hired to help Hot Air to optimize their daily schedule. Formulate a MIP model, implement it using Julia/JuMP, and present a solution to Hot Air management. During the flight to Hot Air's headquarters in the Windy City, you already realize that a possible set of decision variables would be $y_{t,i,j}$, which is the number of aircraft that fly during time period t from city i to city j, and $x_{t,i,j}$, the number of passengers being transported per time period t and from city i to city j. Of course, the latter is bounded by the number of people who want to fly, and the capacity of the planes flying on that route at that time. The data for the problem is given below.

Optimal for Hot Air 1: 12411

Assignment 16.2

Hot Air liked your solution. Now, however, they ask if there's a better way to assign planes to cities for overnight stays. In other words, could a higher profit be realized by changing the home ports of the planes? There's a one-time cost for any possible change (which will not matter in the long run), but operational costs will stay the same.

Hint: This is actually a simplification of the first model and can be solved very elegantly.

Optimal for Hot Air 2: 22368

Assignment 16.3

A manager at Hot Air wants to know about the actual profits for each flight and asks you to calculate these. Create a table that specifies the profit you make for each of the flights.

Assignment 16.4

The manager is shocked to find out that Hot Air is actually *losing* money on certain flights and decides that for now, no flights are allowed if they generate a negative profit. Reformulate your model from Assignment 16.2 to include this requirement. What happens?

Optimal for Hot Air 4: 20950

Assignment 16.5

Hot Air is beginning to feel the pressure from low-cost carriers and is looking for ways to compete with these. The marketing department discovers a new group of possible customers: Low-cost customers (e.g., students!) who can be attracted from bus transportation by lower prices and who do not expect premium service. The marketing department estimates that approximately an extra 50% of customers could be attracted if prices were lowered to 70% of the full-fair price. Hot Air does, however, *not* want to lower the price for the regular customers. Therefore, an employee from the marketing department comes up with the following idea: Only offer discount tickets at 70% of the price to customers who do *not* fly direct flights but have one stopover. Assume that you can still sell the full fair demand tickets, but also that you could additionally fill up the planes with discount passengers. Extend the model from the Assignment 16.2. Does this approach lead to higher profits?

Optimal for Hot Air 5: 28355.6

Hint: Remove the additional constraint from the previous question (engineers know better than managers) and consider introducing a variable to count passengers flying from city i to city j *via* city k. This refers to two flights, with the first flight in period t and the second in period $t + 1$. We will ignore discount travelers who wait for a period or wait overnight. The new variable for the discount traveling passengers (probably students!) should then be added to the objective function *and* to the capacity constraints

(they have to share planes with the normal passengers). When estimating the increased demand you should consider the time period of the first flight.

	Cities			
	Brownsville	Dallas	Austin	El Paso
Brownsville		50	53	14
Dallas	84		80	21
Austin	17	58		40
El Paso	31	79	34	

Table 6.14: Demand, period 1

	Cities			
	Brownsville	Dallas	Austin	El Paso
Brownsville		15	53	52
Dallas	17		134	29
Austin	24	128		99
El Paso	23	15	30	

Table 6.15: Demand, period 2

	Cities			
	Brownsville	Dallas	Austin	El Paso
Brownsville		3	16	9
Dallas	48		104	48
Austin	62	92		68
El Paso	13	15	21	

Table 6.16: Demand, period 3

	Cities			
	Brownsville	Dallas	Austin	El Paso
Brownsville		99	89	139
Dallas	109		99	169
Austin	109	104		129
El Paso	159	149	119	

Table 6.17: Ticket prices

	Cities			
	Brownsville	Dallas	Austin	El Paso
Brownsville		5100	4400	8000
Dallas	5100		11200	6900
Austin	4400	11200		5700
El Paso	8000	6900	5700	

Table 6.18: Cost per takeoff

7 Data

In Julia there are many different data types. The data types we need in this book for our LP models and MIP models are simple: Scalar numbers and arrays of numbers of different size and dimensionality.

7.1 Direct data definition

In the assignments, we need arrays of different dimensions:

- Scalar numbers (zero-dimensional arrays):

```
f=90
Q=31
```

- Vectors (one-dimensional):

```
customer_orders=[11 7 6 12]
C=length(customer_orders)
```

  Notice: The space character separates the numbers. Here we have a vector with 4 numbers. This number can be found using the length function. To access the 3 element simply write:

```
customer_orders[3]
```

  The index always starts at 1 and ends at the length of the array, here 4. If an attempt is done to access an element outside the array, e.g. 0, Julia will give an error.

- Matrix's (two-dimensional):

```
distances=[
17  39  84 ;
109  32  98 ;
```

```
139   4  31 ;
123 129 116 ]
(I,J)=size(distances)
```

Notice: Again the space character separates the numbers, but additionally the semicolon separates the different rows. The size function can be used to find that there are 4 rows (I=4) and 3 columns (J=3). To access the 3 element of the 4 row write:

```
distance[4,3]
```

which has the value 116. For each dimension index validity is checked.

- Matrix's (three-dimensional):

```
time_distances=zeros(Int16,2,4,3)
time_distances[1,:,:]=[
17  39  84 ;
109  32  98 ;
139   4  31 ;
123 129 116 ]
time_distances[2,:,:]=[
37  49  54 ;
609  72  88 ;
939   1  21 ;
133 149 156 ]
(T,I,J)=size(time_distances)
```

Notice: This time, firstly a 3-dimensional matrix of zero values is created. Then the two two-dimensional sub-matrices are filled out. Again we can access elements:

```
time_distance[2,2,3]
```

7.2 Including Julia data

The data section in a Julia program can become rather large and it may be convenient to store the data in a separate file. This can be done very simply by:

```
include("NetworkData.jl")
```

Notice that the data file has to have a Julia format and it is simply inserted into the Julia file at the place of the include statement. Notice that you may have to write the full path to the file.

8 Future reading

This book aims at teaching basic Mathematical Programming modelling, both for LP models and MIP models. However, there are a lot of important subjects regarding Mathematical Programming and the broader field of Operations Research that have been omitted. In this section, we will try to give pointers to material for further studies.

The most used introductory textbook globally on Operations Research is "Introduction to Operations Research" Hillier and Lieberman (2020). This book introduces a lot of topics in Operations Research, including LP and MIP. This book offers a good starting place for anyone who would like a broader introduction to Operations Research.

For a good and more mathematical introduction to the mathematics behind solving MIP, we recommend "Integer Programming" Wolsey (2020).

In our opinion, a good book introducing Julia as a programming language, is still missing. The best material is available online at `julialang.org`, which is also kept up to date.

8.1 Classic OR problems

Most real-world optimization problems are both unique and often very similar. The similarity arises by including parts of a list of classical, well-known Operations Research problems. A number of the exercises in this book are actually formulations of these classic problems, see the list below:

- **The Assignment Problem**: Class Jobs 4.1.3
- **The Transportation Problem**: Chair Distribution 4.1.4
- **The Project Scheduling Problem**: Project Scheduling 4.3.2
- **The Knapsack Problem**: Startup Fund 6.4.
- **The Set Packing/Set Partitioning Problem**: Stamps 6.5

- **The Bin Packing Problem**: Scrap Removal 6.6
- **The Graph Coloring Problem**: Food Festival 6.7
- **The Lot Sizing Problem**: Micro Brewery 2 6.8
- **The Facility Location Problem**: Chair Logistics 2 6.9
- **The Chinese Postman Problem**: Tennis 6.12
- **The Travelling Salesman Problem**: Groceries 6.13
- **The Vehicle Routing Problem**: Groceries 6.13
- **The multi-commodity flow problem**: The Beer Routing Problem 6.14
- **The network design problem**: The Beer Routing Problem 6.14

8.2 Optimization hardness

As mentioned in Chapter 5, the required solution times for MIP models above a certain problem size increases dramatically. The theoretical explanation for this increased solution time is based on the computational complexity theory $P \neq NP$ Garey and Johnson (1979). While computational complexity theory deals with **decision problems** and MIP models handles optimization problems, many of the MIP models are generalizations of so-called NP-complete decision problems. Therefore, the exponential increase in solution time of MIP models is unavoidable. The good news is, however, that tremendous improvements have been a achieved in the last decades Bixby (2012), so that even though a problem may be theoretically hard to solve, MIP models of the relevant size can be solved. Unfortunately, it is impossible to predict how hard it is for a MIP model solver to solve a problem because the complexity of the solvers has grown significant and so has their behaviour on different MIP models, despite the improved average performance Lodi (2013). The easiest way to find out whether a MIP model can be solved is simply to implement it.

Author biographies

Richard Lusby is an Associate Professor at the Technical University of Denmark and teaches courses in Operations Research, Mathematical Modelling, and Decomposition Methods. His main research interests include Integer Programming, Decomposition Methods, and Matheuristics, and their application to public transportation and manpower planning problems.

Thomas Stidsen is an Associate Professor at the Technical University of Denmark and teaches courses in Mathematical Modelling, Metaheuristics, and Decomposition Methods. His main research interests include optimization approaches for educational timetabling and manpower planning, and multi-objective optimization.

Bibliography

Bixby, R. E. (2012). A brief history of linear and mixed-integer programming computation. *Documenta Mathematica*, 2012:107–121.

Danzig, G. (1947). The dantzig simplex method for linear programming.

Garey, M. R. and Johnson, D. S. (1979). *Computers and intractability*, volume 174. freeman San Francisco.

Hillier, F. S. and Lieberman, G. J. (2020). *Operations research concepts and cases*.

Land, A. and Doig, A. (1960). An automatic method of solving discrete programming problems, econometrica, vol. 28, no. 3.

Lodi, A. (2013). The heuristic (dark) side of mip solvers. In *Hybrid metaheuristics*, pages 273–284. Springer.

Wolsey, L. A. (2020). *Integer programming*. John Wiley & Sons.