THE UNIVERSITY OF MELBOURNE
SCHOOL OF MATHEMATICS AND STATISTICS

MAST 90014 - OPTIMISATION FOR INDUSTRY

A/Prof. Alysson M. Costa
School of Mathematics and Statistics
Office 149 - Peter Hall Building

February 20, 2024

Welcome to MAST 90014 - Optimisation for Industry

In MAST 90014, we present the core ideas of Mixed-Integer Programming (MIP), a powerful optimisation framework to solve optimisation problems in Industry and Public Decision Making.

We will place a strong focus on modelling. The first and foremost goal of this subject is to provide students with the ability to understand an optimisation problem and represent it as a mathemtical model using linear constraints on continuous and discrete variables.

The use of linear constraints allow us to solve the modelled problems using the MIP framework. In this subject, students will learn how to use state-of-the-art mathematical programming solvers to obtain solutions for the developed models. They will also understand the algorithms used by these solvers and learn how to extend their capabilities using decomposition methods.

Pre-requisites and requirements

Operations Research courses are taught in Engineering, Computer Science, Business Administration, Mathematics and Economics Schools. Indeed, the area of Operations Research is multidisciplinary by nature, and it has been like that since its origins.

Our course is reasonably self-contained. The only hard requirements are a basic knowledge of linear algebra and intermediate coding skills (we will use Python).

The MAST 90014 cohort historically comprises students in different majors at Unimelb (mainly pure and applied mathematics, engineering and computer science). According to the student's background, they will find different parts of the course more challenging. This is to be expected and students are responsible for coming to office hours to discuss how to catch up with any eventual missing knowledge.

Learning outcomes:

- L1: Have learned how basic techniques in operations research are applied in industry;
- L2: Understand how to turn an industrial problem into a mathematical formulation;
- L3: Know how to solve important mathematical optimisation problems arising in industrial framework;
- L4: Be able to pursue further studies in this and related areas.

Generic skills:

- GS1: Problem-solving skills: the ability to engage with unfamiliar problems and identify relevant solution strategies;
- GS2: Analytical skills: the ability to construct and express logical arguments and to work in abstract or general terms to increase the clarity and efficiency of analysis;
- GS3: Collaborative skills: the ability to work in a team; and
- GS4: Time-management skills: the ability to meet regular deadlines while balancing competing commitments

Teaching Staff

- Lecturer: A/Prof Alysson M. Costa.
- Tutors:

  Alysson M. Costa.

  Nicolau (Nico) Andres Thio.

Content:

1. Modelling in mixed integer programming
2. Strong models in mixed integer programming
3. The simplex algorithm and duality in linear programming
4. Methods for solving mixed integer programming problems
   - Branch and bound
   - Cutting planes
   - Branch and cut
5. Decomposition methods for mixed integer programming
   - Column generation and Branch and price
   - Benders decomposition
   - Lagrangian relaxation (if time allows)
6. An introduction to Stochastic Programming

Contact hours and communication

- Lectures: Weeks 1 to 12, 2h per week.
- Tutorials: Weeks 2 to 12, 1h per week.

Lectures and tutorial attendance are strongly advised. There are no recordings of tutorials and lectures are optimised for in-class participation. Watching the recordings provides a subpar experience and should not be considered a proper substitute for attendance.

Additional time for students to discuss the content is provided by means of weekly office hours. We also have an online forum (Ed Discussion). There is no consultation via e-mail.

Assessment

Individual assignment (20%)

- Students will be provided with the description of one or more optimisation problems and associated instance(s). The goal is to formulate the problem(s) as mixed-integer programming model(s), implement the model(s) in Python with the Gurobipy API and conduct computational experiments and analysis. Evaluation will consider the correctness of the model, the quality of its presentation (including proper definition of all parameters used), and the explanations presented. Proper following of instructions is also a rubric item.
- This assignment aims at developing and evaluating skills L1, L2, L3, GS1, GS2 and GS4.
- Submission:

  (Canvas) PDF document containing models, answers and analysis as requested.

  (Canvas) Jupyter notebook containing running models. The presentation of objective functions and constraints in the Jupyter notebook should follow the order in which they are presented in the PDF document.

  (Canvas) (Optional) PDF document with the history of interactions with large language models (including all prompts used) - See note below for details.

  (Gradescope) Summary sheet following the provided template provided containing numerical answers to the exercises.
- Note on plagiarism: this assignment is individual. Students are **not** allowed to share any piece of code or formulations.
- Note on use of artificial intelligence models: Use of large language models (LLMs) is allowed as long as clearly indicated in the report. If LLMs are used, an additional pdf document with a history of the prompts used must be submitted.
- Deadline (strict) - 5pm of Friday on Week 7.

Group project (40%)

- In this group project, students will work in groups of 6 and come up with a practical optimisation problem. The goal is to find and describe an optimisation problem and associated instance(s). The problem must then be formulated as one or more mixed-integer programming models. The model(s) must be implemented in Python with the Gurobipy API. Computational experiments and analysis, including managerial insights are expected. Evaluation will consider the originality/applicability of the problem selected, quality of the data colection, correctness of the model and the explanations presented, design of experiments and analysis. The quality of the presentation is also an important evaluation criteria.

- The project aims at developing and evaluating skills L1, L2, L3, GS1, GS2, GS3 and GS4.

- Submission:

  (Canvas) Proposal: a PDF document containing up to 2000 words. The document must contain the group members and a brief description of the problem to be investigated. Students might like to add their initial thoughts on data and modelling ideas.

  (Canvas) Slides: a PDF document with the slides to be presented in the last lecture of the semester.

  (Canvas) Report: a PDF document containing a report of up to 10,000 words (20 pages).

  (Canvas) Code: a Jupyter notebook. The presentation of objective functions and constraints in the Jupyter notebook should follow, as much as possible, the order in which they are presented in the report. All results presented in the report must be reproducible using the Jupyter notebook. (Additional instance files might be included as a zipped file as necessary)

- Note on use of artificial intelligence models: Use of large language models (LLMs) is allowed to help with coding tasks as long as clearly indicated in the report. Use of LLMs is **NOT** allowed for generating any part of the text in the report (usage of LLMS in this case will be considered plagiarism).

- Deadlines (strict):

  Project proposal: 5 pm of Friday on week 6.

  Slides: 5pm of the day before the day of the presentations on week 12.

  Report: 5pm of Friday on week 12.

Exam (40%)

- In the exam, students will be evaluated on the theoretical aspects of the subject (i.e., no coding is required). A comprehensive set of past exams is available on Canvas. Exams should be used as practice exercises during the semester, complementing the more applied/coding approach that will be used in the tutorials.

- The exam aims at evaluating skills L1, L2, L4, GS1, GS2 and GS4.
- Exam to be taken in-venue, during the Unimelb examination period.

Main Bibliography

- Laurence A. Wolsey. Integer Programming. John Wiley & Sons, September 2020. [Wolsey, 2020]
- Gerard Sierksma and Yori Zwols. Linear and Integer Optimization: Theory and Practice, Third Edition. Chapman and Hall/CRC, 3rd edition, 2015. [Sierksma and Zwols, 2015]

Chapter 1

Introduction

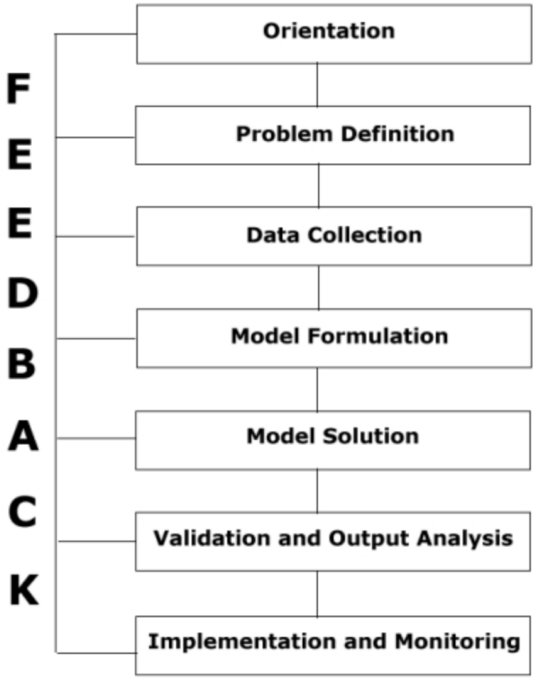
Adjefou Cave: Flying Gallop Chariots, Warriors , Herds

Decision making

- Behavioural & Cognitive Neuroscience (PSYC40004)
- Thinking, Judgement and Decision Making (PSYC90104)
- Managerial Economics (MGMT90043)
- Behavioural Economics (ECON30019)
- Competition and Strategy (ECON20005)
- Game Theory (ECON40010)
- Mathematical Game Theory (MAST90137)

Operations Research

"In a nutshell, operations research (O.R.) is the discipline of applying advanced analytical methods to help make better decisions." INFORMS | What O.R. Is



The Operations Research approach [Zandin and Maynard, 2001]

Applications of Operations Research

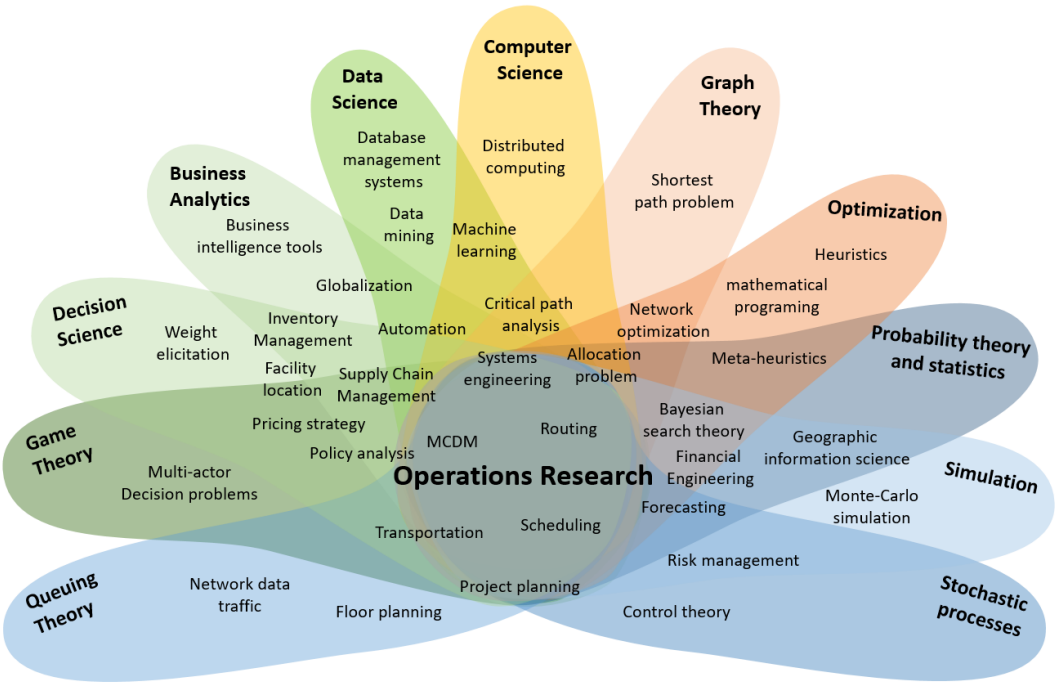
Many applications can be found in areas such as:

Jobs in Operations Research

Given the multi-disciplinary nature of Operations Research, jobs in the area cover a number of different profiles. Overall, a very employable student will have good coding, communication and mathematical skills.

Branches of Operations Research

- Mathematical Programming,
- Dynamic Programming,
- Simulation,
- Heuristics and metaheuristics
- Stochastic Processes / Queuing Theory,
- Inventory Theory,
- Graph Theory,
- etc.



Holistic illustration of the disciplines and problems related to operations research. Image by Alex Elkjaer Vasegaard

Mammoth hunting

For capturing a mammoth, you estimate you need at least 3 hunters that are strong, four smart hunters and 3 fast hunters. What is the minimum number of hunters you can assign to the mission?

Quantity available	*Fast*	*Strong*	*Smart*
8	1	0	1
6	0	1	0
5	0	0	1
3	1	1	0
1	1	1	1

Data:

- R_T: range of hunter types,
- R_C: range of hunter characteristics,
- T_t: number of hunters of type $t \in R_T$,
- B_c: needed number of hunters with characteristic $c \in R_C$ for a successful hunt.
- A_{tc}: 1 if hunter of type $t \in R_T$ has characteristic $c \in R_C$, 0 otherwise.

Model:

Implementation

```
# Tribe
import gurobipy as gp
from gurobipy import GRB

#DATA
T = [8, 6, 5, 3, 1]
B = [3, 3, 4]
A = [[1, 0, 1],
     [0, 1, 0],
     [0, 0, 1],
     [1, 1, 0],
     [1, 1, 1],]

R_T = range(len(T))
R_C = range(len(B))


#MODEL
m = gp.Model("tribe")

#Variables
x = m.addVars(R_T)

#Constraints
m.addConstrs( gp.quicksum( A[t][c]*x[t] for t in R_T) >= B[c] for c in R_C )
m.addConstrs( x[t] <= T[t] for t in R_T )

#Objective
m.setObjective( gp.quicksum(x[t] for t in R_T), GRB.MINIMIZE )


#Solve and print solution
m.optimize()

for t in R_T:
    print("x[",t,"]", "= ", x[t].x, )
```

Mammoth hunting

For capturing a mammoth, you estimate you need at least 3 hunters that are strong, four smart hunters and 3 fast hunters. What is the minimum number of hunters you can assign to the mission?

Quantity available	*Fast*	*Strong*	*Smart*
8	1	0	1
6	0	1	0
5	0	0	1
3	1	1	0
1	1	1	1

Data:

- R_T: range of hunter types,
- T_t: number of hunters of type $t \in R_T$
- S_{Fast}: set of hunter types who are fast.
- S_{Strong}: set of hunter types who are strong.
- S_{Smart}: set of hunter types who are smart.
- B_{Fast}: needed number of fast hunters for a successful hunt.
- B_{Strong}: needed number of fast hunters for a successful hunt.
- B_{Smart}: needed number of fast hunters for a successful hunt.

Model:

Implementation

```
# Tribe
from mip import Model, INTEGER, minimize, xsum

import gurobipy as gp
from gurobipy import GRB

#DATA
T = [8, 6, 5, 3, 1]
R_T = range(len(T))
S_fast = {0,3,4}
S_strong = {1,3,4}
S_smart = {0,2,4}
B_fast = 3
B_strong = 3
B_smart = 4

#Model
m = gp.Model("set_covering")
x = m.addVars(R_T,  name="x")
m.addConstr( gp.quicksum(x[t] for t in S_fast) >= B_fast )
m.addConstr( gp.quicksum(x[t] for t in S_strong) >= B_strong )
m.addConstr( gp.quicksum(x[t] for t in S_smart) >= B_smart)
m.addConstrs( x[t] <= T[t] for t in R_T)
m.setObjective(  gp.quicksum(x[t] for t in R_T), GRB.MINIMIZE)


m.optimize()

# Solution:
for t in R_T:
  print(f"x[{t}] = {x[t].x}" )
```

Chapter 2

Modelling Frameworks

Rene Magritte - The treachery of images

Ceci n’est pas une pipe

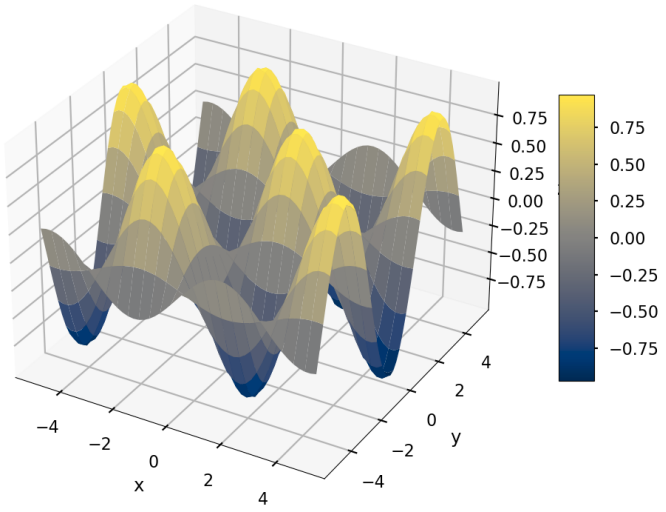
We model nature and society to understand how they work. Modelling is the art of interpreting and representing reality.

Rene Magritte - The treachery of images

Nonlinear Programming

Both constraints and objective function can be nonlinear expressions on the decision variables.

minimise $f(x)$
s.t.
$g(x) \leq 0$,
$h(x) = 0$
$x \in \mathcal{R}^n$.

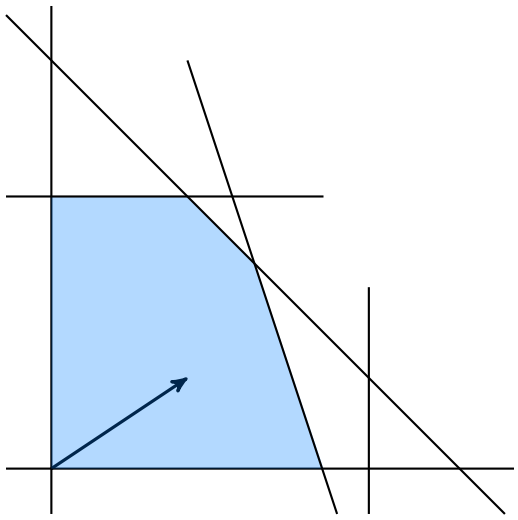


$f(x, y) = sin(x) \cdot \cos(x)$

Linear Programming

The constraints and objective function are linear expressions on continuous decision variables.

$$\begin{aligned} &\text{minimise } c^t x \\ &\text{s.t.} \\ &Ax \geq b \\ &x \geq 0 \end{aligned}$$



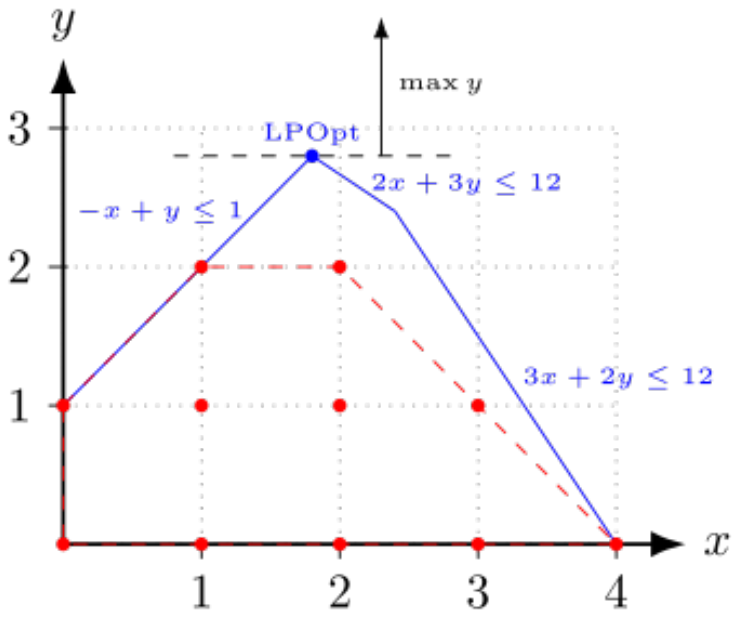
A 2-variable linear programming problem

Mixed-Integer Programming

The constraints and objective function are linear expressions on continuous or discrete decision variables.

minimise $c^t x + d^t y$
s.t.
$Ax + By \geq b$,
$x \geq 0, x \in \mathbb{R}^n$,
$y \geq 0, y \in \mathbb{Z}^n$.

If all variables are integer, we have **Integer Programming**. If all variables are binary, we have **Binary Programming**. In this subject, we use Mixed-Integer Programming (MIP) as a generic term to refer to problems with integer variables (binary or otherwise) and with or without continuous variables.



IP polytope with LP relaxation. Reprinted from Wikimedia Commons, 2010

The right level of simplification

Finding the right level of simplification is key in modelling.

———————

On Exactitude in Science - Jorge Luis Borges, Collected Fictions, translated by Andrew Hurley [Borges, 1946].

...In that Empire, the Art of Cartography attained such Perfection that the map of a single Province occupied the entirety of a City, and the map of the Empire, the entirety of a Province. In time, those Unconscionable Maps no longer satisfied, and the Cartographers Guilds struck a Map of the Empire whose size was that of the Empire, and which coincided point for point with it. The following Generations, who were not so fond of the Study of Cartography as their Forebears had been, saw that that vast Map was Useless, and not without some Pitilessness was it, that they delivered it up to the Inclemencies of Sun and Winters. In the Deserts of the West, still today, there are Tattered Ruins of that Map, inhabited by Animals and Beggars; in all the Land there is no other Relic of the Disciplines of Geography.

Suarez Miranda,Viajes devarones prudentes, Libro IV,Cap. XLV, Lerida, 1658.

—

The right model

Decision making models are mostly useful when they can be solved. Using a modelling framework allows for solving the models with generic algorithms for that framework.

The choice of the modelling framework is often guided by the capacity of solution algorithms to solve realistic instances of a problem.

Algorithms shed light on models

Chapter 3

MIP modelling

The diet problem

The goal of the diet problem is to select a set of foods that will satisfy a set of daily nutritional requirement at minimum cost.

Assume there are three food types available: corn, milk, and bread. Nominal nutrient intake must be between 2000 and 2250 for calories and between 5000 and 50,000 for vitamin A. The table lists, for each food, the cost per serving, the amount of Vitamin A per serving, and the number of calories per serving. Suppose that the maximum number of servings per food is 10.

Food	Cost per serving	Vitamin A	Calories
Corn	\$0.18	107	72
2% Milk	\$0.23	500	121
Wheat Bread	\$0.05	0	65

Data:

- F: set of foods

  c_i: cost per serving of food i,

  F^i_{min}: minimum number of required servings of food i,

  F^i_{max}: maximum allowable number of servings of food i,

- N: set of nutrients
- N^j_{min} = minimum required level of nutrient j,
- N^j_{max} = maximum allowable level of nutrient j.
- a_{ij}: amount of nutrient j in food i

Model:

Example: The diet problem - integer servings

Consider again the diet problem. Now you can only integral number of servings of each food.

Model:

$$
\begin{aligned}
&\text{minimise} \sum_{i \in F} c_i x_i \\
&\text{s.t.} \\
&\sum_{i \in I} a_{ij} x_i \geq N^j_{\min}, \quad j \in J \\
&\sum_{i \in I} a_{ij} x_i \leq N^j_{\max}, \quad j \in J \\
&x_i \geq F^i_{\min}, \quad i \in F, \\
&x_i \leq F^i_{\max}, \quad i \in F, \\
&x_i \geq 0, x_i \in \mathbb{Z} \quad i \in I, j \in J.
\end{aligned}
$$

The diet problem is a classical Linear Programming problem. If we say that servings must be integer it becomes an integer programing problem.

Implementation

```
x = m.addVars(Food, name = 'x', ub=10, vtype=GRB.INTEGER)
```

Building blocks problems

The bin packing problem

Pack all items in bins, such that the number of bins used is minimised. Each item has a weight and each bin a capacity.

Data:

- I: set of items to be packed,

  w_i: weight of item i,
- K: capacity of each bin.

Variables:

- y_j: binary variable equal to 1 if bin number j is selected.
- x_{ij}: binary variable equal to 1 if item i is placed in bin number j.

Model:

$$\text{minimise} \quad \sum_{j \in J} y_j$$

s.t.

$$\sum_{j \in J} x_{ij} = 1, \quad \forall i \in I,$$

$$\sum_{i \in I} w_i x_{ij} \leq K y_j, \quad \forall j \in J,$$

$$y_j \in \{0, 1\}, \quad \forall j \in J,$$

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

The knapsack problem

Let us consider an optimisation problem where you need to select among possible items. This definition is broad enough to metaphorically represent many real problems: select among items to put in a knapsack for a hiking trip (water, food, ...); select among possible investments in the stock market, etc.

Data:

- I : set of available items
- w_i : weight of item $i \in I$,
- p_i : value/utility of item $i \in I$.
- c : the capacity of the knapsack.

Variables:

- x_i : the selected number of items of type $i \in I$.

Model:

$$\text{maximise} \sum_{i \in I} p_i x_i$$

s.t.

$$\sum_{i \in I} w_i x_i \leq c,$$
$$x_i \in \mathbb{Z}_+, \quad i \in I.$$

The assignment problem

The assignment problem deals with the question of how to assign items to other items. In its most classical version, the goal is to assign n agents to n tasks. Each task must be executed by a single agent and each agent can only execute one task. There is a cost associated with each assignment. Minimise the total assignment cost.

Data:

- n : number of agents, tasks
- c_{ij} : preference for assigning a task j to a worker i

Variables:

- x_{ij} equal to one if worker i is assigned to task j.

Model:

$$\begin{aligned}
&\text{minimise} \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \\
&\text{s.t.} \\
&\sum_{j=1}^{n} x_{ij} = 1, \quad i = 1,\ldots,n, \\
&\sum_{i=1}^{n} x_{ij} = 1, \quad j = 1,\ldots,n, \\
&x_{ij} \in \{0,1\}, \quad i,j = 1,\ldots,n.
\end{aligned}$$

The set covering problem

Given a certain number of regions, the problem is to decide where to install a set of emergency service centres. For each possible centre, the cost of installing a service centre and which regions it can service are known. For instance, if the centres are fire stations, a station can service those regions for which a fire engine is guaranteed to arrive on the scene of a fire within eight minutes. The goal is to choose a minimum cost set of service centres so that each region is covered. [Wolsey, 2020].

Data:

- I: set of regions to be serviced,
- J: set of candidates for service centres,
- c_j: cost of installing the service centre j.
- a_{ij}: parameter equal to 1 if centre j can serve region i and to 0 otherwise.

Variables:

- x_j: binary variable equal to 1 if centre j is selected and to 0 otherwise.

Model:

$$\text{minimise} \sum_{j \in J} c_j x_j$$

s.t.

$$\sum_{j \in J} a_{ij} x_j \geq 1, \quad i \in I,$$

$$x_j \in \{0, 1\}, \quad j \in J$$

The set partitioning

Each region must be served by a single emergency centre.

$$\text{minimise} \sum_{j \in J} c_j x_j$$

s.t.

$$\sum_{j \in J} a_{ij} x_j = 1, \quad i \in I,$$

$$x_j \in \{0, 1\}, \quad j \in J$$

The set packing

Each region must be served by at most a single emergency centre.

$$\text{minimise} \sum_{j \in J} c_j x_j$$

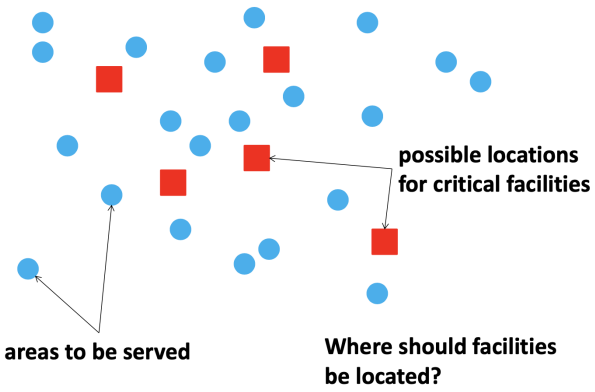
s.t.

$$\sum_{j \in J} a_{ij} x_j \leq 1, \quad i \in I,$$

$$x_j \in \{0, 1\}, \quad j \in J$$

The uncapacitated facility location problem

Given a set of potential depots $N = \{1, ..., n\}$ and a set $M = \{1, ..., m\}$ of clients, suppose there is a fixed cost f_j associated with the use of depot j, and a transportation cost c_{ij} if all of client i's order is delivered from depot j. The problem is to decide which depots to open and which depot serves each client so as to minimize the sum of the fixed and transportation costs. Note that this problem is similar to the covering problem, except for the addition of the variable transportation costs [Wolsey, 2020].



Uncapacitated facility location graphical representation

Variables:

- x_j equal to 1 if depot j is used, 0 otherwise.
- y_{ij}: fraction of client i demand served by depot j.

Model:

$$\text{minimise} \quad \sum_{i \in M} \sum_{j \in N} c_{ij} y_{ij} + \sum_{j \in N} f_j x_j$$

s.t.

$$\sum_{j \in N} y_{ij} = 1, \quad \forall i \in M,$$

$$\sum_{i \in M} y_{ij} \leq m x_j, \quad \forall j \in N$$

$$y_{ij} \geq 0 \quad \forall i \in M, j \in N,$$

$$x_j \in \{0, 1\} \quad \forall j \in N.$$

Capacitated version

What if facilities have capacities?

Single supply

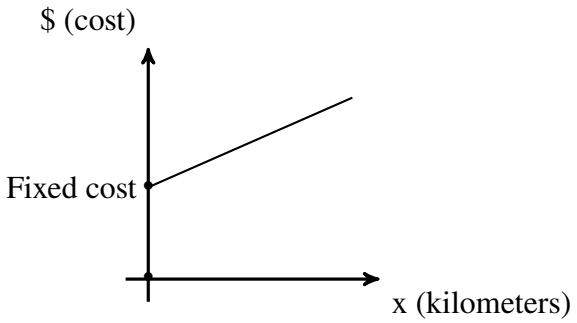
What if a client must receive all his demand from a single supplier? (Is this a practical requirement?)

Integer variables as a modelling tool

Beyond forcing the values of variables to be integer (because they should be in the practical application), integrality constraints allow us to model other characteristics of a problem.

Example: fixed costs

In many situations you have to pay a fixed cost to start one activity. This fixed-cost can be associated with the setup of machine, the building of some infrastructure, etc. Consider the modelling of the price of a taxi ride. It depends linearly on the number of kilometers ($ c per kilometer) plus a fixed starting cost, K. Assume a cost minimisation goal.



$$\begin{aligned} \text{cost} &= Ky + cx \\ x &\leq My \\ y &\in \{0,1\} \end{aligned}$$

Example: Disjunctive constraints

In many situations, some constraints maybe deactivated if other constraints are respected. Assume that you must respect one of the two constraints: $x_1 + x_2 \geq 100$ or $x_1 + x_2 \leq 20$.

Example: If-then implications

Some decisions might imply other decisions. You need to tell your model that this is the case. Let x_1 be the amount to be produced of item 1 and x_2 be the amount to be produced of item 2. How to model the fact that a minimum quantity of item 2 **must** be produced if item 1 is produced ?

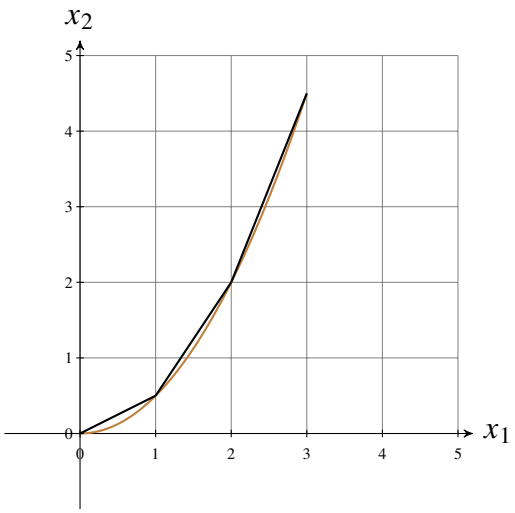
Example: Logical constraints

In general, binary variables will allow us to inpose all sort of logical relations between our decisions. Suppose you can choose among five projects (binary variables $x_1, \ldots x_5$). There are some constraints on the projects that you can choose:

1. At most three projects can be selected,
2. One of projects 1 or 2 must be selected,
3. Only one of projects 3 or 4 might be selected,
4. If project 1 is selected, then project five must be selected,
5. If either of projects 2,3 or 4 is selected, then project five must be selected.

Example: Piecewise linear modelling

Approximation of non-linear relations.



Approximate a non-linear function with linear constraints. In this example, consider that the objective function worsens as x_2 increases.

General case

Let $I = \{1, \ldots, n\}$ be associated with the segments in the linear by parts approximation and $I_0 = I \cup \{0\}$. Also, let $(a_i, b_i), i \in \{0\} \cup I_0$ define the extremities of those segments.

(In the example: $(a_0, b_0) = (0, 0)$, $(a_1, b_1) = (1, 0.5)$,$(a_2, b_2) = (2, 2)$,$(a_3, b_3) = (3, 4.5)$).

We define auxiliary variables $y_i, i \in I$ equal to one if $x_1 \in [a_{i-1}, a_i]$ and auxiliary variables $\lambda_i \in I_0$ indicating the percentage of extremity (a_i, b_i) corresponding to x_1. The model reads:

Linear by parts approximation:

$$\sum_{i \in I} y_i = 1, \tag{3.1}$$

$$\lambda_0 \leq y_1, \tag{3.2}$$

$$\lambda_i \leq y_i + y_{i+1}, \quad \forall i \in 1, \ldots, n-1, \tag{3.3}$$

$$\sum_{i \in I_0} \lambda_i = 1, \tag{3.4}$$

$$x_1 = \sum_{i \in I_0} a_i \lambda_i, \tag{3.5}$$

$$x_2 = \sum_{i \in I_0} b_i \lambda_i, \tag{3.6}$$

$$\lambda_i \geq 0, \quad \forall i \in I_0$$

$$y_i \in \{0, 1\}, \quad \forall i \in I$$

Constraints (3.1) force a single segment to be chosen. Constraints (3.2) – (3.5) define x_1 to be a convex combination between the two x-coordinate extremities of the selected segment, while (3.6) uses the parameters of the convex combination of x_1 to find the value of x_2.

Note: if the function is being minimized and is convex, or is being maximized and is concave, then binary variables are unnecessary. Why?

The lot-sizing problem

Decide what and when to produce in order to supply a demand.

We are given a finite time horizon, an overhead cost of production, a per unit cost of producing a product, a per unit cost of holding the product in storage (inventory) and the demand for the product in each period of the time horizon.

The goal is to develop a production plan for the product over the given time horizon, i.e., how many units of the product must be produced and stored in each time period, such that the demand is met in the period it occurs and the total costs are minimized.

Data:

T - number of time periods in the given time horizon

f_t - overhead cost of production in period $t \in 1, \ldots, T$

p_t - per unit cost of production in period $t \in 1, \ldots, T$

h_t - per unit cost of holding product in inventory through period $t \in 1, \ldots, T$

d_t - demand for the product in period $t \in 1, \ldots, T$

Variables:

x_t number of units produced in period t

s_t number of units held in storage at end of period t

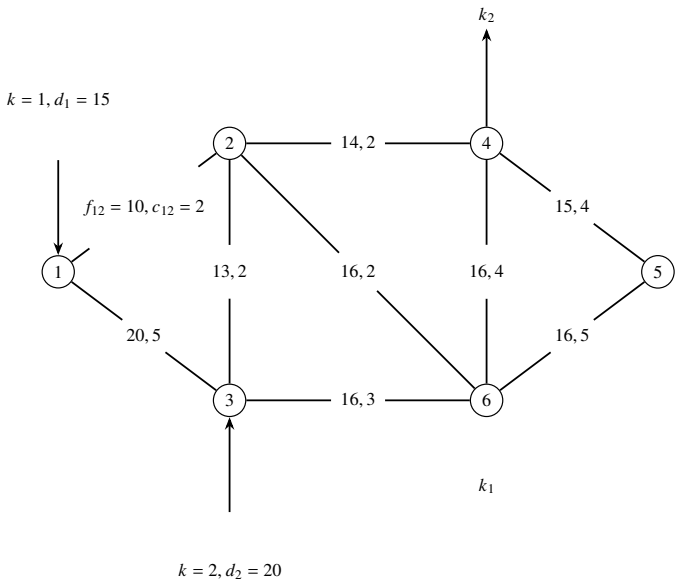
y_t is one if the product is produced in period t

Model:

$$\begin{aligned}
\text{minimise} \quad & z = \sum_{t=1}^{T} \Big(f_t y_t + p_t x_t + h_t s_t \Big) \\
\text{subject to} \quad & s_t = s_{t-1} + x_t - d_t && t = 1, \ldots, T, \\
& x_t \leqslant M_t y_t && t = 1, \ldots, T, \\
& x_t, s_t \geqslant 0 && t = 1, \ldots, T, \\
& y_t \in \{0, 1\} && t = 1, \ldots, T.
\end{aligned}$$

The fixed-cost network design problem

Network design problems appear frequently in areas such as transportation, telecommunications and power systems when we need to establish links to enable the flow of commodities such as vehicles, data packets, electricity. The goal, is to build infrastructure to allow for the flow of commodities. There are usually fixed costs associated with building the infrastructure and flow costs that depend on the amount of commodity being transported



Data:

- V: set of nodes (locations)
- A: set of possible links to be built,

  f_{ij}: cost of building link e,

  c_{ij}: cost of allowing one unit of flow through link e,
- K: set of commodities,

  d_k: demand of commodity k,

  $O(k)$: origin node of commodity k,

  $D(k)$: destination node of commodity k,

Model:

$$\text{minimize} \sum_{(i,j)\in A} \left(\sum_{k\in K} c_{ij}^k x_{ij}^k + f_{ij} y_{ij} \right)$$

s.t.

$$\sum_{j|(i,j)\in A} x_{ij}^k - \sum_{j|(j,i)\in A} x_{ji}^k = \begin{cases} d_k, & i = O(k), \quad \forall k \in K. \\ 0, & i \notin \{O(k), D(k)\}, \quad \forall k \in K. \\ -d_k, & i = D(k), \quad \forall k \in K. \end{cases}$$

$$x_{ij}^k \leq d_k y_{ij}, \qquad \forall (i,j) \in A, \quad \forall k \in K.$$

$$x_{ij}^k \geq 0, \qquad \forall (i,j) \in A, \quad \forall k \in K.$$

$$y_{ij} \in \{0,1\}, \qquad \forall (i,j) \in A.$$

Machine Scheduling (single job)

A job needs to be scheduled (the time it is done decided) within the planning horizon $[0, T]$. However the machine used to perform the job is unavailable for the period $[a, b]$, so the job must be scheduled either before a, or after b.

Variables:

- t: start time of the job.
- x: binary variable equal to 1 if $t \geq b$.

Model:

$$t \geq bx,$$
$$t \leq a(1 - x) + Mx$$

What is the smallest value of M?

(This is an example of disjunctive constraints).

Machine Scheduling (multiple jobs)

n jobs must be scheduled for processing on a machine. Each job i requires a specified processing time p_i, and has a due date d_i, by which it should be completed, $i = 1, \ldots, n$.

Variables:

- t_i: start time of the job i.
- x_{ij}: if job i is executed before job j.

Model:

Define artificial job 0, $n+1$, $t_0 = 0$, $p_0 = 0$, $p_{n-1} = 0$

$$\sum_{j=1}^{n+1} x_{ij} = 1, \quad \forall i = 0, \ldots, n$$

$$\sum_{i=0}^{n} x_{ij} = 1, \quad \forall j = 1, \ldots, n+1.$$

$$t_j \geq t_i + p_i - M(1 - x_{ij}), \quad \forall i = 1, \ldots, n, j = 0, \ldots, n, j \neq i,$$

$$t_i \leq d_i, \quad \forall i = 1, \ldots, n.$$

Scheduling on Parallel Machines

The simplest problem on machine scheduling on parallel machines consists in assigning tasks to machines such that the completion time of the last job to be completed (makespan) is minimised.

Data:

- I: set of jobs,

  p_i: processing time of job $i \in I$,

- J: set of machines,

Variables:

- x_{ij}: binary variable equal to 1 if job $i \in I$ is scheduled on machine $j \in J$.
- c: Makespan value.

Model:

minimise c

s.t.

$$\sum_{i \in I} p_i x_{ij} \leq c, \quad \forall j \in J,$$

$x_{ij} \in \{0, 1\}, c \geq 0$

Other scheduling problems

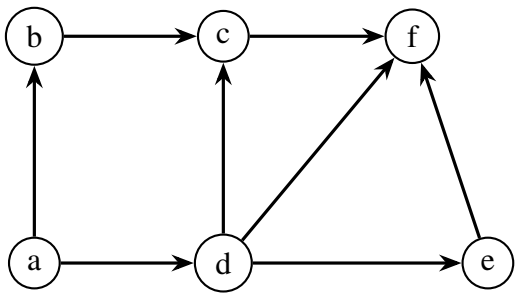
Scheduling problems appear in a variety of important contexts. We have a full subject on the topic of scheduling problems.

See Scheduling and Optimisation (MAST90050).

The assembly line balancing problem

An assembly line is a manufacturing production system which models the workforce as an homogeneous commodity. In an assembly line, the product is moved through a set of stations during the manufacturing process. At each of $S = \{1, \ldots, \overline{s}\}$ stations, some of the $I = \{1, \ldots, \overline{n}\}$ tasks needed for manufacturing are executed. The simple assembly line balancing problem consists in distributing these tasks among the stations, while respecting precedence constraints on the order of the execution of the tasks.

The order of execution of tasks might be limited by precedence relations. In a precedence graph, each node represents a task and each arc a precedence relation.



Precedence graph example

Task b can only be executed after task a is executed. Task c can only be executed after task d is executed, and so on. Let P be the set of precedence pairs:

$$P = \{(i, j) | i, j \in I, i \text{ must precede task } j\}.$$

for the example:

$$P = \{(a, b), (a, d), (b, c), (d, c), (d, e), (d, f), (e, f)\}$$

Variables:

- x_{is} equal to 1 if task i is assigned to station s, 0 otherwise.
- C: cycle time.

Model (SALBP-2):

z = minimise C

s.t.

$$\sum_{s \in S} x_{is} = 1, \quad \forall i \in I,$$

$$\sum_{s \in S} s x_{is} \leq \sum_{s \in S} s x_{js}, \quad \forall (i,j) \in P.$$

$$\sum_{i \in I} t_i x_{is} \leq C \quad \forall s \in S.$$

$$C \geq 0, x_{is} \in \{0,1\} \quad \forall s \in S, i \in I.$$

SALBP-1

Assembly line problems have many variants. For example, you might be interested in ensuring a production level while minimising the needed number of stations. How would you model this problem?

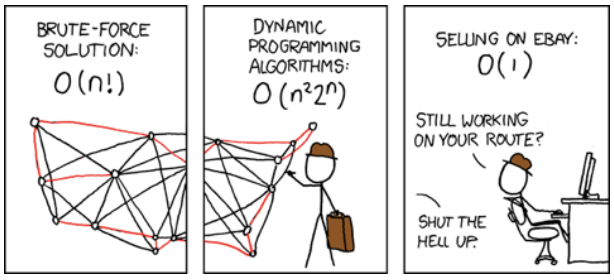
The traveling salesperson problem

The travelling salesperson problem (TSP) is the archetypal problem in operations research. Visit a number of cities and return to the origin while minimising the distance travelled. It is easy to explain and hard to solve mostly due to its combinatorial characteristic.

The travelling salesperson problem (TSP) is the archetypal problem in operations research. Visit a number of cities and return to the origin while minimising the distance travelled. It is easy to explain and hard to solve mostly due to its combinatorial characteristic.

Touristic cities in Belgium (from MIP Python manual)

Its idea is extremely simple: assume you are a travelling salesperson who wants to leave your home, visit a number of cities to offer your products and return to your home. You want to do this travelling as little as possible, so you need to decide on the order you visit the cities. In graph theory jargon, we want to find a least cost Hamiltonian circuit on a graph.



XKCD comic

Data:

- I: set of cities,
- E: set of roads between cities,

  c_{ij}: cost/time of traversing edge (i, j)

Variables:

- x_{ij} equal to 1 if the salesperson travels from city i to city j, 0 otherwise.

Model:

$$\text{minimise} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

s.t.

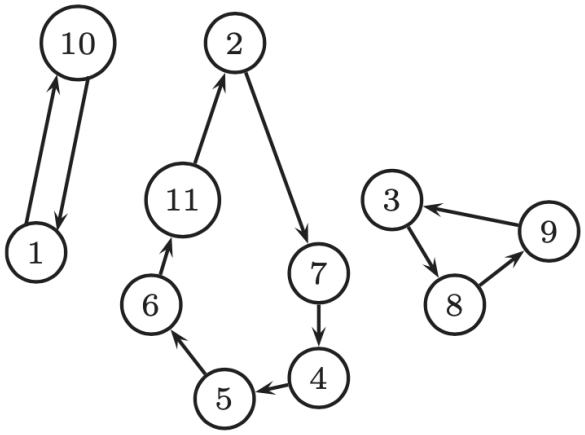
$$\sum_{j=1}^{n} x_{ij} = 1, \quad i \in I,$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad j \in I,$$

Subtour elimination constraints

$$x_{ij} \in \{0, 1\}, \quad i, j = 1, \ldots, n.$$

Dantzig Fulkerson Johnson SECs



Examples of subtours [Wolsey, 2020]

The first version of subtour elimination constraints was introduced as 'loop conditions' in the classical 1954 paper [Dantzig et al., 1954].

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \qquad S \subset I, 2 \leq |S| \leq n - 1.$$

Alternatively, you can ensure connectivity with the so-called cut-set constraints:

$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 1, \qquad S \subset I, S \neq \emptyset.$$

Miller Tucker Zemlin SECs

A polynomial-sized formulation was proposed by [Miller et al., 1960]. These constraints define y_i variables for $i \in I \backslash \{1\}$ indicating the order the cities are visited.

Variables:

- x_{ij} equal to 1 if arc (i, j) is traversed in the solution.
- $y_i \geq 0$ potential associated with node i.

Model:

$$\text{minimise} \sum_{i \in I, j \in I} c_{i,j} \cdot x_{i,j}$$

Subject to:

$$\sum_{j \in V \backslash \{i\}} x_{ij} = 1, \quad \forall i \in V$$

$$\sum_{i \in V \backslash \{j\}} x_{ij} = 1, \quad \forall j \in V$$

$$y_i - (n+1)x_{ij} \geq y_j - n, \quad \forall i \in V \backslash \{0\}, j \in V \backslash \{0, i\}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i \in V, j \in V$$

$$y_i \geq 0, \quad \forall i \in V$$

Other models

There are many ways to model subtour elimination constraints. These constraints appear in many different problems where cycles are not allowed in the solutions.

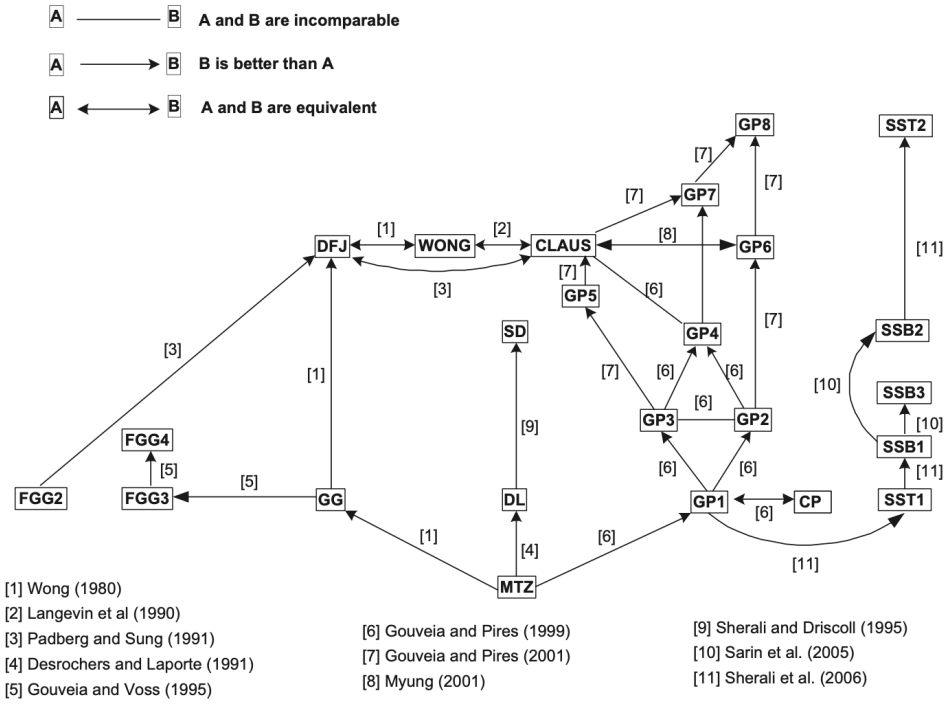


Fig. 1. Known relationships between 24 ATSP formulations.

A comparative analysis of several asymmetric traveling salesman problem formulations [Öncan et al., 2009]

The (capacitated) vehicle routing problem

In the Capacitated VRP (CVRP), all the customers correspond to deliveries and the demands are deterministic, known in advance, and may not be split. The vehicles are identical and based at a single central depot, and only the capacity restrictions for the vehicles are imposed. The objective is to minimize the total cost (i.e., a weighted function of the number of routes and their length or travel time) to serve all the customers [Toth and Vigo, 2002].

Data:

- V: set of clients + depot (node 0)

  d_i: demand of client $i \in V \backslash \{0\}$

- E: set of roads between nodes,

  c_{ij}: cost/time of traversing edge $(i, j) \in E$,

- K: number of available vehicles,

- S: a subset of cities,

  $r(S)$: minimum number of vehicles to serve the demand of nodes in set S.

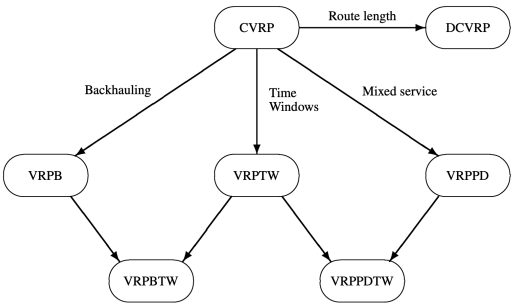
Variables:

- x_{ij} equal to 1 if a vehicle travels from node $i \in V$ to node $j \in V$, 0 otherwise.

Model:

$$
\begin{aligned}
&\text{minimise} \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \\
&\sum_{i \in V} x_{ij} = 1 \quad \forall j \in V \backslash \{0\} \\
&\sum_{j \in V} x_{ij} = 1 \quad \forall i \in V \backslash \{0\} \\
&\sum_{i \in V} x_{i0} = K, \\
&\sum_{j \in V} x_{0j} = K, \\
&\sum_{i \notin S} \sum_{j \in S} x_{ij} \geq r(S) \quad \forall S \subseteq V \backslash \{0\}, S \neq \emptyset, \\
&x_{ij} \in \{0, 1\} \quad \forall i, j \in V.
\end{aligned}
$$

Variants of the CVRP



The basic Vehicle Routing Problems [Toth and Vigo, 2002]

Examples of larger models

.

Network scheduling problem with cross-docking and loading constraints

Computers & Operations Research 132 (2021) 105271

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

Network scheduling problem with cross-docking and loading constraints

Pedro B. Castellucci [a,*], Alysson M. Costa [b], Franklina Toledo [c]

[a] Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, Brazil
[b] School of Mathematics and Statistics, The University of Melbourne, Australia
[c] Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, Brazil

ARTICLE INFO

Keywords:
Vehicle routing
Three-dimensional packing
Benders decomposition
City logistics

ABSTRACT

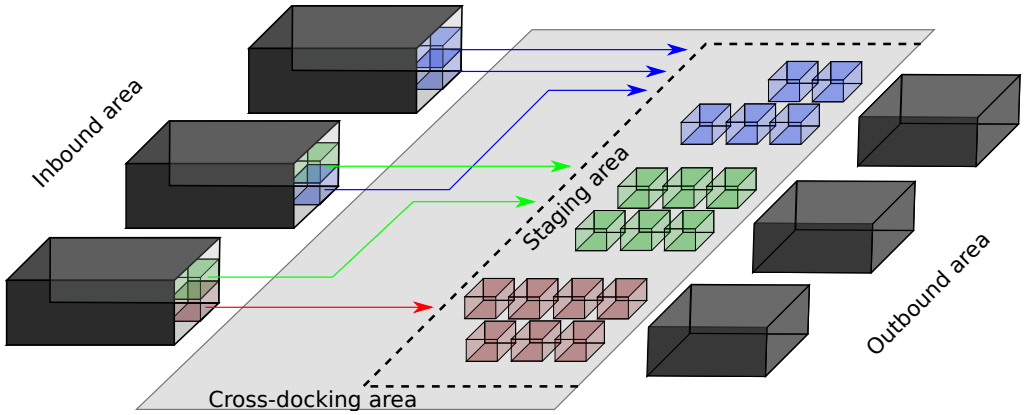
Cross-docking is a logistic strategy which can increase rates of consolidation, and reduce distribution and storage costs and delivery times. The optimization literature on cross-docking has mostly focused on the modeling and solution of problems considering a single cross-docking facility. Networks with multiple cross-docks remain rather unexplored and the few papers that deal with the problem do this by simplifying the geometry of the goods. We intend to shorten this gap by proposing a mixed-integer linear programming model for optimizing distribution and delay costs for the transportation of goods in open networks with multiple cross-docks considering the three-dimensional aspects of the cargo. Also, we propose a logic-based Benders decomposition strategy which allow for the solution of larger instances when compared with those that can be handled by a branch-and-cut MIP solver. Experiments showed that the decomposition can handle instances with two times more nodes and five times more boxes than a direct use of the solver. Also, the framework is flexible enough to accommodate other features of practical cases.

In a logistic distribution system, two of the most expensive tasks are storage and order picking. Cross-docking tries to eliminate both by synchronizing the input and the output flow of goods. In a cross-dock, goods are unloaded at an input dock, sorted according to their destination and loaded into trucks at an output dock, reducing storage requirements. This strategy aims at obtaining higher rates of consolidation, shorter delivery and lead times, and has been implemented successfully in many situations. Figure 3.7 shows a schematic representation of a typical cross-dock.

Problem definition

We focus on distribution networks in which a set of suppliers (origins), O, satisfies demands from a set of consumers (destinations), $\mathcal{D}$, by shipping their goods through intermediate (cross-docking) facilities, $\mathcal{F}$.

Leaving each origin $o \in O$, there is a set of shipments $k \in \mathcal{K}_O$ and each shipment contains a set of boxes, or goods, $\mathcal{B}_k$. Let ℓ_{ia} be the length of each box $i \in \mathcal{B}$, $\mathcal{B} = \bigcup_{k \in \mathcal{K}} \mathcal{B}_k$, along axis $a \in \mathcal{A}$, $\mathcal{A} = \{\text{x-axis, y-axis, z-axis}\}$. Also, let UT_k and ST_k be the time required to unload and sort shipment $k \in \mathcal{K} = \bigcup_{o \in O} \mathcal{K}_o$, respectively. Each cross-dock $f \in \mathcal{F}$ has a set of available vehicles,



Schematic representation of a cross-dock. Trucks are unloaded at the inbound area, the boxes are sorted according to their destination and loaded into trucks at the outbound area.

$\mathcal{W}_f$, each with length L_a along axis, $a \in \mathcal{A}$, and loading time LT_w, $w \in \mathcal{W} = \bigcup_{f \in \mathcal{F}} \mathcal{W}_f$. Each destination $d \in \mathcal{D}$ demands for the set of boxes $\mathcal{B}_d \subset \mathcal{B}$. All boxes demanded are supplied (i.e. $\bigcup_{k \in \mathcal{K}} \mathcal{B}_k = \bigcup_{d \in \mathcal{D}} \mathcal{B}_d$). Furthermore, let the cost and time of transporting goods from supplier $i \in O$ to cross-dock $j \in \mathcal{F}$ or from cross-dock $i \in \mathcal{F}$ to consumer $j \in \mathcal{D}$ be c_{ij} and t_{ij}, respectively. Finally, there is a deadline τ_d associated with each consumer $d \in \mathcal{D}$. All the trucks must reach their respective destinations before the deadline, otherwise there is a penalty of R per unit of delay. Also, we assume that each truck visits at most a single demand point (consumer), but a consumer might be visited by multiple trucks. Finally, let $\overline{u}_{if}$ be the time from which box $i \in \mathcal{B}$ is available for loading in a truck in facility $f \in \mathcal{F}$ (we show how these parameters can be computed later in this section). All parameters are summarized in Table 3.1.

The goal is to find an assignment of the shipments leaving suppliers to cross-docks and of boxes to trucks which minimizes total distribution and delay costs. For this, let Δ_{kf} be binary variables indicating whether shipment $k \in \mathcal{K}$ is assigned to cross-dock $f \in \mathcal{F}$ and p_{iw} be binary variables indicating whether box $i \in \mathcal{B}$ is assigned to truck $w \in \mathcal{W}$. Also, binary variables v_{wd} define whether truck $w \in \mathcal{W}$ is assigned to destination $d \in \mathcal{D}$. Moreover, there are continuous non-negative variables m_w defining the time a truck $w \in \mathcal{W}$ is ready for departure and e_{wd} defining the lateness of delivery truck $w \in \mathcal{W}$ for consumer $d \in \mathcal{D}$. To ensure the feasibility of truck loading, we also need to position each box inside the respective truck. For this, let x_{ia} be non negative variables indicating the position of box $i \in \mathcal{B}$ along axis $a \in \mathcal{A}$ and $r^{af}_{ija}(r^{bf}_{ija})$ be binary variables indicating whether box $i \in \mathcal{B}$ is after (before) box $j \in \mathcal{B}, j > i$, in axis $a \in \mathcal{A}$. In summary, we have the variables in Table 3.2.

Model

We formulate the resulting network scheduling problem with cross-dock (NSCD) (3.7)–(3.17).

Summary of parameters for the network scheduling problem.

O:	Set of suppliers.
$\mathcal{D}$:	Set of destinations.
$\mathcal{F}$:	Set of cross-docks.
$\mathcal{K}_O$:	Set of shipments leaving supplier $o \in O$.
$\mathcal{K}$:	Set of all shipments $\left(\bigcup_{o \in O} \mathcal{K}_o \right)$.
$\mathcal{B}_k$:	Set of boxes in shipment $k \in \mathcal{K}$.
$\mathcal{B}$:	Set of all boxes $\left(\bigcup_{k \in \mathcal{K}} \mathcal{B}_k \right)$.
ℓ_{ia}:	Length of box $i \in \mathcal{B}$ along axis $a \in \mathcal{A}$.
UT_k:	Unloading time of shipment $k \in \mathcal{K}$.
ST_k:	Sorting time of shipment $k \in \mathcal{K}$.
$\mathcal{W}_f$:	Set of trucks available at cross-dock $f \in \mathcal{F}$.
$\mathcal{W}$:	Set of all available trucks $\mathcal{W} = \bigcup_{f \in \mathcal{F}} \mathcal{W}_f$.
L_a:	Length of each truck along axis $a \in \mathcal{A}$.
LT_w:	Loading time of truck $w \in \mathcal{W}$.
c_{ij}:	Cost of moving from a supplier (cross-dock) i to a cross-dock (destination) j.
$\overline{u}_{if}$:	The time from which box $i \in \mathcal{B}$ is available for loading in a truck in facility $f \in \mathcal{F}$.
$\mathcal{B}_d$:	Set of boxes demanded by consumer $d \in \mathcal{D}$.
t_{ij}:	Travel time between points $i \in O \cup \mathcal{F}$ and $j \in \mathcal{F} \cup \mathcal{D}$.
τ_d:	Deadline associated with client $d \in \mathcal{D}$.
R:	Penalty per unit of time delay.

Summary of the variables for the network scheduling problem.

Δ_{kf}:	Binary variables indicating whether shipment $k \in \mathcal{K}$ is sent to facility $f \in \mathcal{F}$.
p_{iw}:	Binary variables indicating whether box $i \in \mathcal{B}$ is loaded into truck $w \in \mathcal{W}$.
v_{wd}:	Binary variables indicating whether vehicle $w \in \mathcal{W}$ serves client $d \in \mathcal{D}$.
e_{wd}:	Lateness associated with truck $w \in \mathcal{W}$ and destination $d \in \mathcal{D}$.
x_{ia}:	Non-negative variables indicating the position of box $i \in \mathcal{B}$ in axis $a\mathcal{A}$.
$r^a_{ija}(r^b_{ija})$:	Binary variable indicating if box $i \in \mathcal{B}$ is after (before) $j \in \mathcal{B}$ in axis $a \in \mathcal{A}, i < j$.

$$\text{(NSCD)Min} \sum_{o\in O}\sum_{k\in\mathcal{K}_o}\sum_{f\in\mathcal{F}} c_{of}\Delta_{kf} + \sum_{f\in\mathcal{F}}\sum_{w\in\mathcal{W}_f}\sum_{d\in\mathcal{D}} c_{fd}v_{wd} + R\sum_{w\in\mathcal{W}}\sum_{d\in\mathcal{D}} e_{wd} \tag{3.7}$$

subject to:

$$\sum_{f\in\mathcal{F}} \Delta_{kf} = 1, \qquad k \in \mathcal{K}, \tag{3.8}$$

$$\Delta_{kf} = \sum_{w\in\mathcal{W}_f} p_{iw}, \qquad f \in \mathcal{F}, k \in \mathcal{K}, i \in \mathcal{B}_k, \tag{3.9}$$

$$v_{wd} \geq p_{iw}, \qquad d \in \mathcal{D}, i \in \mathcal{B}_d, w \in \mathcal{W}, \tag{3.10}$$

$$\sum_{d\in\mathcal{D}} v_{wd} \leq 1, \qquad w \in \mathcal{W}, \tag{3.11}$$

$$m_w \geq (LT_w + \overline{u}_{if})p_{iw}, \qquad f \in \mathcal{F}, w \in \mathcal{W}_f, i \in \mathcal{B}, \tag{3.12}$$

$$m_w + t_{fd} - \tau_d - (1 - v_{wd})T_{dfw} \leq e_{wd}, \qquad d \in \mathcal{D}, f \in \mathcal{F}, w \in \mathcal{W}_f, \tag{3.13}$$

$$x_{ia} + \ell_{ia} \leq x_{ja} + (1 - r^{bf}_{ija})L_a, \qquad i, j \in \mathcal{B}, i < j, a \in \mathcal{A}, \tag{3.14}$$

$$x_{ja} + \ell_{ja} \leq x_{ia} + (1 - r^{af}_{ija})L_a, \qquad i, j \in \mathcal{B}, i < j, a \in \mathcal{A}, \tag{3.15}$$

$$\sum_{a\in\mathcal{A}} (r^{b}_{ija} + r^{a}_{ija}) \geq p_{iw} + p_{jw} - 1, \qquad i, j \in \mathcal{B}, i < j, \tag{3.16}$$

$$x_{ia} \leq L_a - \ell_{ia} \qquad i \in \mathcal{B}, a \in \mathcal{A}. \tag{3.17}$$

Domain of the variables.

The objective function (3.7) minimizes the cost of first and second layer distributions and the cost of delays. The first layer distribution cost is defined as the total cost of moving goods from origins $o \in O$ to cross-docks $f \in \mathcal{F}$ and the second layer distribution costs is defined as the total costs of moving goods from cross-docks $f \in \mathcal{F}$ to respective destinations $d \in \mathcal{D}$. The third term is the cost of delays. Constraints (3.8) ensure that every shipment goes through exactly one of the cross-docks. In each cross-dock $f \in \mathcal{F}$, the inbound boxes are unloaded and loaded into one of the available trucks $\mathcal{W}_f$, according to constraints (3.9). The outbound trucks deliver goods to no more than one consumer (constraints (3.10) and (3.11)). The time each truck is ready for departure is defined by constraints (3.12), whilst delivery delays for each destination are computed by constraints (3.13) in which T_{dfw}, $d \in \mathcal{D}$, $f \in \mathcal{F}$, $w \in \mathcal{W}_f$ is a sufficiently big number. Each truck has a limited capacity, defined by its three dimensional geometry. Constraints (3.14) and (3.15) ensure there is no overlapping between any pair of boxes. Note that constraints (3.14) and (3.15) are only active if two boxes are in the same truck due to constraints (3.16). Constraints (3.17) define an upper bound for the position of the boxes.

Pattern-based models and a cooperative parallel metaheuristic for high school timetabling problems

European Journal of Operational Research 280 (2020) 1064–1081

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Innovative Applications of O.R.

Pattern-based models and a cooperative parallel metaheuristic for high school timetabling problems

Landir Saviniec[a,*], Maristela O. Santos[b], Alysson M. Costa[c], Lana M. R. dos Santos[d]

[a] Universidade Federal do Paraná, Campus Avançado de Jandaia do Sul, Rua Doutor João Maximiano 426, Jandaia do Sul, Paraná 86900-000, Brazil
[b] Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, Avenida Trabalhador São-carlense 400, São Carlos, São Paulo 13566-590, Brazil
[c] School of Mathematics and Statistics, The University of Melbourne, Parkville, VIC 3010, Australia
[d] Departamento de Matemática, Universidade Federal de Viçosa, Avenida Peter Henry Rolfs – Campus Universitário, Viçosa, Minas Gerais 36570-000, Brazil

ARTICLE INFO

Article history:
Received 31 October 2018
Accepted 2 August 2019
Available online 8 August 2019

Keywords:
Timetabling
Parallel metaheuristics
Column generation
Iterated local search

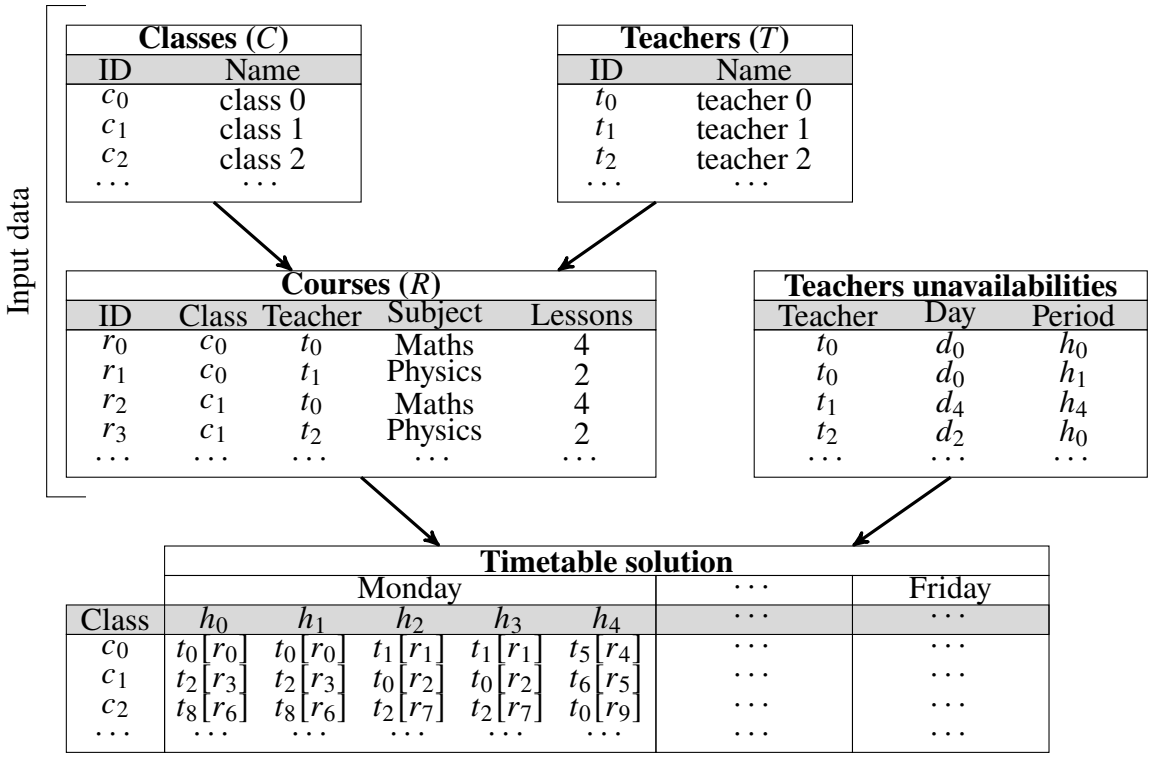
ABSTRACT

High school timetabling problems consist in building periodic timetables for class-teacher meetings considering compulsory and non-compulsory requirements. This family of problems has been widely studied since the 1950s, mostly via mixed-integer programming and metaheuristic techniques. However, the efficient search of optimal or near-optimal solutions is still a challenge for many problems of practical size. In this paper, we investigate mixed-integer programming formulations and a parallel metaheuristic based algorithm for solving high school timetabling problems with compactness and balancing requirements. We propose two pattern-based formulations and a solution algorithm that simultaneously exploits column generation and a team of metaheuristics to build and improve solutions. Extensive computational experiments conducted with real-world instances demonstrate that our formulations are competitive with the best existing high school timetabling formulations, while our parallel algorithm presents superior performance to alternative methods available in the literature.

Educational timetabling problems consist in assigning meetings between teachers (or exams) and students, considering a list of different hard (mandatory) or soft (non-mandatory) requirements. A timetable that satisfies all hard requirements is said to be feasible, and a timetable that is feasible and also has a minimum number of violated soft requirements is said to be optimal.

Problem definition

We consider the high school timetabling problem originated in the context of Brazilian public high schools. The problem structure is represented in Figure 3.8. The school has a set of classes C and a set of teachers T. Classes are disjoint groups of students enrolled in the same list of subjects. Classes should have no free periods during the week. Each class' subject has a number of weekly lessons that are taught by a preassigned teacher. The set of all classes' subjects is called "Courses" (R) and each class' subject is called "a course". Teachers may be unavailable in some periods/days. Given these input data, the goal is to obtain a weekly timetable specifying the schedule for all courses in a particular shift (morning, afternoon, or evening). A timetable is constructed for each independent shift, and it usually spreads over the set of weekdays D (Monday to Friday) with (usually) five periods each. We define H as the set of periods per day and refer to periods as a set of timeslots $(d, h) \in D \times H$.



Example of high school timetabling data and solution.

The timetabling process must fulfill the following hard requirements:

1. **Meeting of weekly lessons**: the lessons of each course $r \in R$ must be assigned.
2. **No clashes in classes' schedules**: each class $c \in C$ must attend exactly one lesson per period.
3. **No clashes in teachers' schedules**: each teacher $t \in T$ must teach at most one lesson per period.
4. **No assignment of teachers in their unavailable periods**: teachers must not be assigned to periods in which they are unavailable.

The following soft requirements are also considered:

5. **No daily workload violation for courses**: for each course $r \in R$, there is a soft limit $\widetilde{\delta}_r$ on the amount of lessons per day to each given class.
6. **No gaps in courses' schedules**: gaps in courses' schedules should be avoided. A *gap* is a period splitting the daily meetings of a course to a given class into non-consecutive lessons.
7. **Meeting of double lessons for courses**: for each course $r \in R$, there is a specified minimum number of consecutive double lessons $\widetilde{\pi}_r$ that should be met.
8. **No idle periods in teachers' schedules**: teachers should not have free periods between busy periods within the same day. For the purpose of this requirement, unavailable periods are considered as busy periods. Therefore, they are not computed as idle periods.
9. **Minimizing the number of working days for teachers**: teachers should be scheduled to come to school the minimum possible number of days.
10. **Balancing on teachers' extra working days**: the extra working days should be balanced among teachers. The number of extra working days for a teacher $t \in T$ is the difference between the number of worked days and an estimated lower bound $\widetilde{d}_t$, defined later in Table 3.3.

Model

This section presents our two models for the problem. Both models are extensive formulations as they rely on teachers/classes meeting patterns. We propose a type of pattern with limited information: teachers patterns only indicate if a teacher is teaching or not in a given period (and relegate the decision on which class they are teaching to the master problem) while courses patterns only indicate if a given course is being taught or not in a given period (and relegate the decision on the class to which this course is being taught to the master problem).

We call the limited-information patterns 'layouts'. Our first extensive formulation, EF1, exploits the fact that the cardinality of the sets of layouts is much smaller than the cardinality of traditional (full-information) patterns. The second formulation, EF2, considers a more traditional approach with weekly layouts indicating the full schedule of a teacher. Both models and solution strategies are discussed in the remainder of this section, using the notation in Table 3.3.

The model uses the idea of assignment layouts for the daily schedules of teachers and course. A *daily assignment layout* is defined as a binary vector of size $|H|$ in which the h^{th} position is associated with the h^{th} period of the day. For a teacher, a bit value of "*1*" indicates that the teacher is assigned at that period, while for a course it indicates that there is one lesson associated with that course at that period.

Input parameters of a high school timetabling instance.

Notation	Definition
Sets	
C	the set of classes.
T	the set of teachers.
D	the set of weekdays.
H	the set of periods per day.
$H_{td} \subseteq H$	the subset of periods in day $d \in D$ for which teacher $t \in T$ is available.
R	the set of courses.
$R_c \subseteq R$	the subset of courses associated to a class $c \in C$.
$R_t \subseteq R$	the subset of courses associated to a teacher $t \in T$.
Parameters	
$\widetilde{\theta}_r \in \mathbb{N}$	the number of weekly lessons for a course $r \in R$.
$\widetilde{\delta}_r \in \mathbb{N}$	the daily limit for the number of lessons of a course $r \in R$.
$\widetilde{\pi}_r \in \mathbb{N}$	the desired minimum number of double lessons for a course $r \in R$.
$\widetilde{d}_t = \left\lceil \frac{\sum_{r \in R_t} \widetilde{\theta}_r}{\lvert H \rvert} \right\rceil$	a lower bound on the minimum number of days for which a teacher $t \in T$ can be assigned and still meet his/her teaching requirements.

The number of all possible daily layouts for a course or teacher (disregarding unavailable periods) is given by $2^{|H|}$. For $|H| = 5$, a teaching layout '00000' for a given day indicates that the associated teacher has that day off while a layout '00111' indicates that the teacher has lessons assigned to him/her in the last three teaching periods of the day. This definition of "layouts" is different from the one used in most of the literature which usually explicitly codify which class the teacher should meet at each timeslot. Here, for teacher layouts, there is only the definition of the presence or absence of the considered teacher in a timeslot. Analogously, for course layouts, a layout '11000' indicates that a double lesson of that subject is taking place in the first two periods of the day while a layout '10100' represents an undesired gap between two lessons of the same course.

Table 3.4 presents the data of the HSTP in an appropriate format and also lists the two sets of binary decision variables, x_{rdl} and y_{tdl}. These binary variables indicate whether a course $r \in R$ (x_{rdl}) or teacher $t \in T$ (y_{tdl}) follows an assignment layout $l \in \{0, 1, \cdots, 2^{|H|} - 1\}$ on day $d \in D$. Model EF1 can thus be written as:

$$\textbf{Minimize} \quad \sum_{r \in R} \sum_{d \in D} \sum_{l \in L} (\alpha_5 \widetilde{e}_{rl} + \alpha_6 \widetilde{h}_l) x_{rdl} + \alpha_7 \sum_{r \in R} \widehat{\pi}_r + \sum_{t \in T} \sum_{d \in D} \sum_{l \in L} (\alpha_8 \widetilde{i}_{tdl} + \alpha_9 \widetilde{\phi}_l) y_{tdl} + \alpha_{10} \widehat{\beta} \tag{3.18}$$

Subject to:

Additional notation and decision variables used in formulation EF1.

Notation	Definition
Sets	
L	the set of assignment layouts that a course r (or teacher t) can follow in a day.
Parameters	
$\widetilde{\mu}_{lh} \in \{0,1\}$	indicates whether a layout $l \in L$ has an assignment at period $h \in H$.
$\widetilde{a}_l \in \mathbb{N}$	the number of assignments in a layout l.
$\widetilde{e}_{rl} \in \mathbb{N}$	the number of lessons that exceed the daily limit $\widetilde{\delta}_r$ in a layout l for a course r.
$\widetilde{h}_l \in \mathbb{N}$	the number of gaps in a layout l for course.
$\widetilde{\omega}_l \in \mathbb{N}$	the number of consecutive double lessons in a layout l for courses.
$\widetilde{\phi}_l \in \{0,1\}$	indicates whether the number of assignments in a layout l is greater than zero.
$\widetilde{i}_{tdl} \in \mathbb{N}$	the number of idle periods in a layout l for a teacher t in a day d.
Decision variables	
$x_{rdl} \in \{0,1\}$	indicates whether a course r follows a layout l in a day d.
$y_{tdl} \in \{0,1\}$	indicates whether a teacher t follows a layout l in a day d.
Auxiliary variables	
$\widehat{\pi}_r \in \mathbb{N}$	the number of unmet weekly double lessons for a course r.
$\widehat{\beta} \in \mathbb{N}$	the largest value among all teachers' extra working days.

$$\sum_{d \in D} \sum_{l \in L} \widetilde{a}_l x_{rdl} = \widetilde{\theta}_r \qquad \forall\, r \in R \tag{3.19}$$

$$\sum_{r \in R_c} \sum_{l \in L} \widetilde{\mu}_{lh} x_{rdl} = 1 \qquad \forall\, c \in C;\ d \in D;\ h \in H \tag{3.20}$$

$$\sum_{l \in L} x_{rdl} = 1 \qquad \forall\, r \in R;\ d \in D \tag{3.21}$$

$$\sum_{l \in L} y_{tdl} = 1 \qquad \forall\, t \in T;\ d \in D \tag{3.22}$$

$$\widehat{\pi}_r \geq \widetilde{\pi}_r - \sum_{d \in D} \sum_{l \in L} \widetilde{\omega}_l x_{rdl} \qquad \forall\, r \in R \tag{3.23}$$

$$\widehat{\beta} \geq \sum_{d \in D} \sum_{l \in L} \widetilde{\phi}_l y_{tdl} - \widetilde{d}_t \qquad \forall\, t \in T \tag{3.24}$$

$$\sum_{l \in L} \widetilde{\mu}_{lh} y_{tdl} = \sum_{r \in R_t} \sum_{l \in L} \widetilde{\mu}_{lh} x_{rdl} \qquad \forall\, t \in T;\ d \in D;\ h \in H \tag{3.25}$$

$$\widehat{\pi}_r \geq 0 \qquad \forall\, r \in R \tag{3.26}$$

$$\widehat{\beta} \geq 0 \tag{3.27}$$

$$x_{rdl} \in \{0,1\} \qquad \forall\, r \in R;\ d \in D;\ l \in L \tag{3.28}$$

$$y_{tdl} \in \{0,1\} \qquad \forall\, t \in T;\ d \in D;\ l \in L \tag{3.29}$$

Objective function (3.18) minimizes violations of soft requirements **5** to **10**, in which α_i is a penalty parameter associated with the relative importance of the i^{th} requirement. Note that for requirement **10**, we use a min-max approach where the objective is to minimize the 'extra busy days' of the teacher with maximum value of this parameter. This choice is in accord with what is done in previous papers but has the issue of being *blind* to the load of teachers other than the most loaded one (in terms of busy days). Alternative modeling could penalize the total sum (over all teachers) of extra days, either linearly or quadratically. In the experiments we used the objective function as shown in (3.18) and (3.24). Constraints (3.19) ensure that the weekly lessons of all courses are scheduled. Constraints (3.20) avoid clashes in classes' schedules. Constraints (3.21) and (3.22) ensure that a single layout is assigned to each course and each teacher on a given day, respectively. Constraints (3.23) and (3.24) link the auxiliary variables to the decision variables in order to quantify violations of soft requirements **7** and **10**: constraints (3.23) compute the number of unmet double lessons for each course while constraints (3.24) capture the largest value among all teachers' extra working days. Finally, the linking constraints (3.25) ensure that the daily schedules of teachers match their course' schedules.

To avoid assigning teachers to their unavailable periods, we preprocess the associated variables and forbid layouts which contain infeasible assignments. This task can be accomplished by setting the following variables to zero:

$$y_{tdl} = 0 \qquad \forall\, t \in T;\ d \in D;\ l \in L;\ h \in H \setminus H_{td};\ \widetilde{\mu}_{lh} > 0 \tag{3.30}$$

$$x_{rdl} = 0 \qquad \forall\, t \in T;\ r \in R_t;\ d \in D;\ l \in L;\ h \in H \setminus H_{td};\ \widetilde{\mu}_{lh} > 0 \tag{3.31}$$

Chapter 4

Linear Programming

Rules of Linear Programming:

- Variables can assume continuous values.
- All constraints of the problem are written in terms of the variables, in a linear fashion.
- The objective function is written in terms of the variables, in a linear fashion.

Brief timeline:

- Fourier 1826: systems of linear inequalities
- Dantzig 1947: simplex method designed for air force logistics
- Nobel Prize in Economics 1975: Kantorovich and Koopmans
- Khachian 1979: Fast solution methods in theory - NY Times
- Karmarkar 1984: Practically fast solution methods, headlines in the New York Times, academic uproar
- Cplex Optimization Inc. founded 1988, fast, reliable software for solving linear programs, now IBM Cplex.
- Frontline Systems Solver for linear programs distributed with Microsoft Excel since 1990, 80 million distributed worldwide
- Today hundreds of companies develop and sell software solving optimization problems.

Suggested reading: Bixby, Robert E. "A Brief History of Linear and Mixed-Integer Programming Computation." Documenta Mathematica, 2012, 16. [Bixby, 2012].

Example: The Dovetail problem [Sierksma and Zwols, 2015]

The company Dovetail produces two kinds of matches: long and short ones. The company makes a profit of 3 (x$1,000) for every 100,000 boxes of long matches, and 2 (x$1,000) for every 100,000 boxes of short matches. The company has one machine that can produce both long and short matches, with a total of at most 9 (x100,000) boxes per year. For the production of matches the company needs wood and boxes: three cubic meters of wood are needed for 100,000 boxes of long matches, and one cubic meter of wood is needed for 100,000 boxes of short matches. The company has 18 cubic meters of wood available for the next year. Moreover, Dovetail has 7 (x100,000) boxes for long matches, and 6 (x100,000) for short matches available at its production site. The company wants to maximize its profit in the next year. It is assumed that Dovetail can sell any amount it produces.

Variables:

$x_1 \geq 0$: number of 100,000 boxes of long matches produced,

$x_2 \geq 0$: number of 100,000 boxes of short matches produced.

Model:

Let z be the maximum profit the company can obtain given such technological and resource constraints.

maximise $3x_1 + 2x_2$
s.t.
$x_1 + x_2 \leq 9$,
$3x_1 + x_2 \leq 18$,
$x_1 \leq 7$,
$x_2 \leq 6$,
$x_1, x_2 \geq 0$.

Dovetail model with the use of slack variables:

maximise $3x_1 + 2x_2$

s.t.

$x_1 + x_2 + s_1 = 9,$

$3x_1 + x_2 + s_2 = 18,$

$x_1 + s_3 = 7,$

$x_2 + s_4 = 6,$

$x_1, x_2, s_1, s_2, s_3, s_4 \geq 0.$

Writing linear programs with equality constraints.

What if we had ≥ constraints? What if we need variables to assume negative values? What if our objective function is a minimisation one?

Canonical representations of a LP

maximise $c^t x$
s.t.
$Ax \leq b$,
$x \geq 0$.

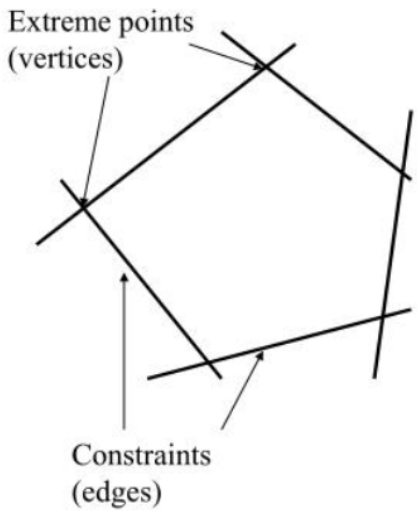
$$A = \begin{bmatrix} 1 & 1 \\ 3 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, b = \begin{bmatrix} 9 \\ 18 \\ 7 \\ 6 \end{bmatrix}, c = \begin{bmatrix} 3 \\ 2 \end{bmatrix}.$$

maximise $c^t x$
s.t.
$Ax = b, \quad \forall b \geq 0$
$x \geq 0$.

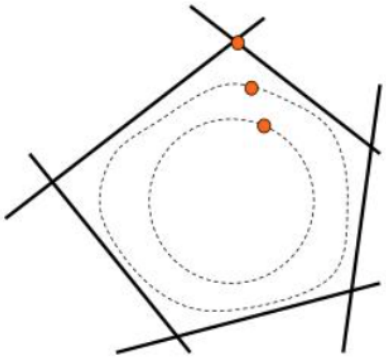
$$A = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}, b = \begin{bmatrix} 9 \\ 18 \\ 7 \\ 6 \end{bmatrix}, c = \begin{bmatrix} 3 \\ 2 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Simplex and Interior Point Methods

We will see that the optimal solutions of linear programs occur on the vertices of the feasible region. To find the optimal vertex, there are two approaches:



Simplex: search from vertex to vertex along the edges



Interior-point methods: go through the inside of the feasible space

Definitions

Convexity

A subset of a Euclidean space over the reals is convex if, given any two points, it contains the whole line segment that joins them.

Let $a \in \mathbb{R}$ and $b \in \mathbb{R}$ be two real numbers. If C is a convex set and $a, b \in C$, then: $c = \alpha a + (1-\alpha)b \in C$, for all $\alpha \in [0, 1]$

Hyperplanes

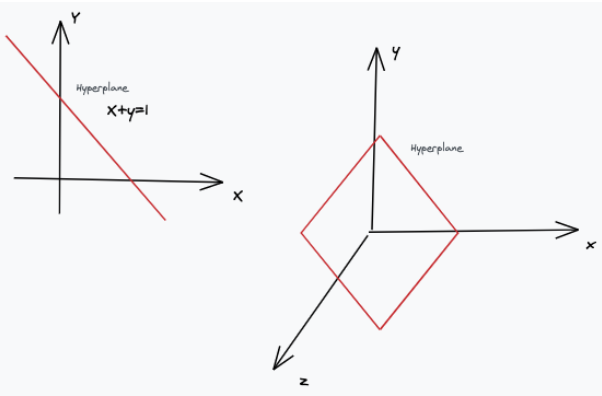
A hyperplane in $\mathbb{R}^n$:

$$H = \left\{ \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n \mid a_1 x_1 + \ldots + a_n x_n = b \right\} = \left\{ \mathbf{x} \in \mathbb{R}^n \mid \mathbf{a}^\top \mathbf{x} = b \right\},$$

Hyperplanes divide the space they lie into two halfspaces.

Halfspaces

Each of the spaces defined by a hyperplane.



Examples of hyperplanes

Polyhedrons

A polyhedron P is an intersection of affine half-spaces.

If $P = \emptyset$, then we say that the LP is infeasible, and write $z = -\infty$ (max problem); otherwise it is feasible.

If for all $v \in \mathbb{R}$ there exists $x \in P$ such that $c^t x \geq v$, then we say the LP is unbounded, and write $z = +\infty$ (max problems) .

Theorem 1 *The space defined by linear constraints $Ax = b$ is convex.*

Proof:

Assume x_1 and x_2 are such that they respect the system of linear equations.

Consider a point x_3 in the convex combination of x_1 and x_2:

$$x_3 = \alpha x_1 + (1 - \alpha)x_2, \text{with } 0 \leq \alpha \leq 1.$$

Therefore, $Ax_3 = A(\alpha x_1 + (1 - \alpha)x_2)$,

$Ax_3 = \alpha Ax_1 + (1 - \alpha)Ax_2$,

$Ax_3 = \alpha b + (1 - \alpha)b$,

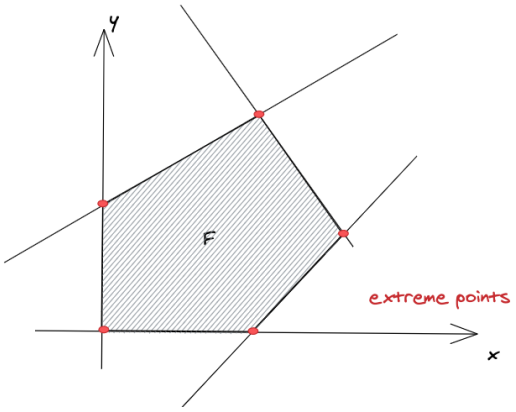
$Ax_3 = b$,

and therefore it also respects the equalities, showing that a set of equality constraints defines a convex set.

Extreme points

A vertex in the polyhedron defined by the constraints of our problem.

The figure below shows extreme points in the feasible region F.



Feasible extreme points for feasible region F

Three ways to define an extreme point [Sierksma and Zwols, 2015]:

- The vector $\mathbf{x}^0 \in F$ is called a vertex of F if and only if there are n independent hyperplanes in the collection $\{H_1, \ldots, H_{m+n}\}$ that intersect at $\mathbf{x}^0$.
- The vector $\mathbf{x}^0 \in F$ is called a vertex of F if and only if there is a hyperplane H (not necessarily one of $H_1, \ldots, H_{m+n}$) with corresponding halfspace H^+ such that $F \subseteq H^+$ and $F \cap H = \{\mathbf{x}^0\}$.
- The vector $\mathbf{x}^0 \in F$ is called a vertex of F if and only if there are no two distinct $\mathbf{x}', \mathbf{x}'' \in F$ such that $\mathbf{x}^0 = \lambda\mathbf{x}' + (1-\lambda)\mathbf{x}''$ for some $\lambda \in (0, 1)$

Extreme points

What are the coordinates of the extreme points of the Dovetail problem? Which are feasible? Which variables have zero value at each extreme point?

maximise $3x_1 + 2x_2$

s.t.

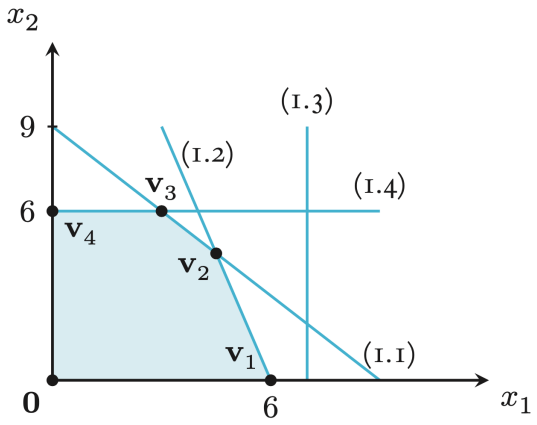
$x_1 + x_2 + s_1 = 9,$

$3x_1 + x_2 + s_2 = 18,$

$x_1 + s_3 = 7,$

$x_2 + s_4 = 6,$

$x_1, x_2, s_1, s_2, s_3, s_4 \geq 0.$



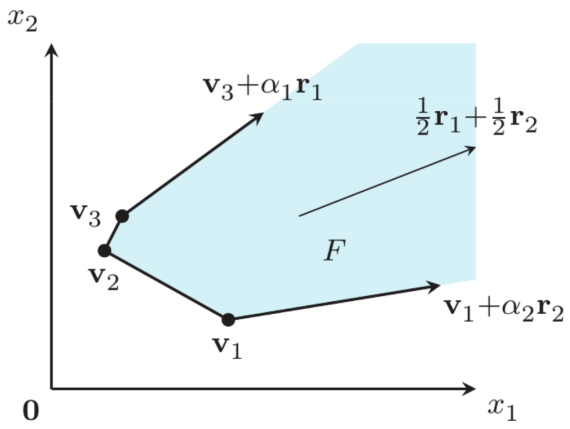
Feasible region[Sierksma and Zwols, 2015]

Extreme rays

We call $r \in \mathcal{R}^n$ an extreme ray of F if and only if for every $x \in F$, $x + \alpha r$ is also in F for any $\alpha \geq 0$.

A ray (or direction) r is a direction that you can 'walk' indefinitely from any feasible point and still remain feasible.

An extreme ray is a direction that can not be written as a convex combination of other directions. In the figure below, r_1 and r_2 are extreme rays of the Feasible region.



Extreme points and rays [Sierksma and Zwols, 2015]

Basic solutions of a LP

The feasible region of a linear programming model can be seen as the intersection of the hyperspaces defined by the problem constraints (including the non-negativity ones).

In developing the simplex method, we are interested in developing a representation of extreme points, as we already know the method will use these points in its way to the optimal solution. We will start with an example and then generalise the ideas.

Theorem 2 *(Basic solutions) A solution $x \in \mathbb{R}^n \geq 0$ of a system with m constraints $Ax = b$ with rank(A) = m, with at least $n - m$ of its components at zero is an extreme point of the feasible space.*

Proof: Assume x^R is a feasible point with $n - m$ components at zero but is not an extreme point. Therefore, it can be written as a convex combination of two distinct feasible extreme points:

$$x^R = \alpha x^P + (1 - \alpha)x^Q, \text{ with } 0 \leq \alpha \leq 1.$$

The equation can be written as:

$$\begin{bmatrix} 0 \\ \vdots \\ 0 \\ x^R_{n-m} \\ \vdots \\ x^R_n \end{bmatrix} = \alpha \begin{bmatrix} x^P_1 \\ \vdots \\ x^P_{n-m} \\ x^P_{n-m+1} \\ \vdots \\ x^P_n \end{bmatrix} + (1 - \alpha) \begin{bmatrix} x^Q_1 \\ \vdots \\ x^Q_{n-m} \\ x^Q_{n-m+1} \\ \vdots \\ x^Q_n \end{bmatrix}$$

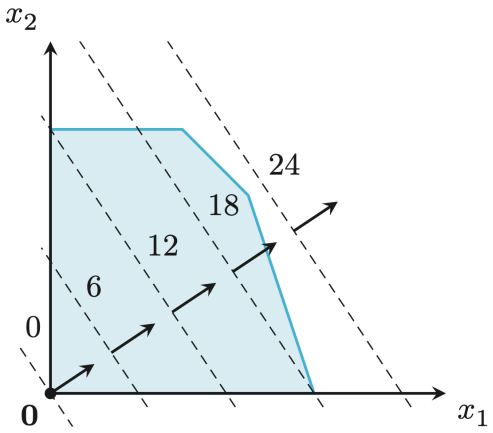
Since x^P and x^Q are both feasible, all their components $x^P_i, x^Q_i \geq 0$.

With $0 \leq \alpha \leq 1$, we can conclude that the first $n - m$ components of x^P and x^Q must also be zero.

Moreover, by hypothesis, all three points respect the system of equalities $Ax = b$ (which has $n - m$ degrees of liberty) and therefore all their remaining components must also be equal: $x^R = x^P = x^Q$ This contradicts the hypothesis that x^P and x^Q are distinct.

The graphical method for LP

We saw that the space defined by linear constraints is convex. With a linear objective function, if an optimal solution exists, there will always be an optimal extreme point. This can be found with the graphical method for problems with 2 variables.



Feasible region with objective function level curves [Sierksma and Zwols, 2015]

Example: Graphical Solution method

Sove the problem with the graphical method.

maximise $z = 3x + 2y$

s.t.

$$2x + y \leq 10$$
$$x + y \leq 8$$
$$x, y \geq 0$$

Other methods

Brute Force:

Test all intersections of constraints (possible extreme points). Check they are feasible. Calculate objective function value. Choose the best.

Questions: How long could this take? How much computation is involved? How does the computational effort depend on the number of variables and constraints?

Interior Point Methods:

Ellipsoid algorithm: guaranteed convergence in number of iterations polynomial in the number of variables and constraints. Difficult to make computationally effective in practice.

Predictor-corrector methods: follow central path through the feasible region. Call nonlinear equation solver such as Newton's method to take next steps. Can be very effective in practice.

The simplex algorithm:

Start at one extreme point. Identify "neighbouring" extreme point that improves the objective value. Move to that extreme point. Repeat until no moves possible.

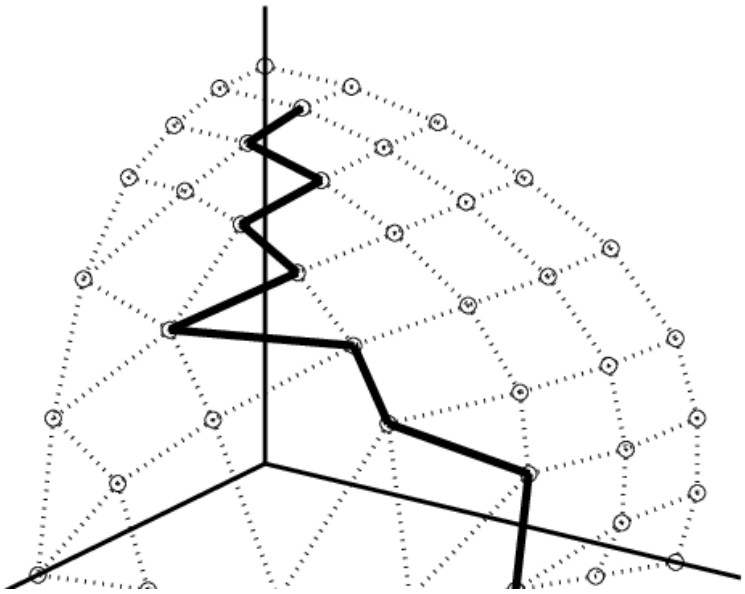
Questions: How long could this take? How much computation is involved? How does the computational effort depend on the number of variables and constraints? Worst case? Average?

The simplex algorithm

Dantzig's simplex algorithm starts at a feasible extreme point and at each step it finds a neighbour feasible extreme point with better solution. If no neighbour with better objective function can be found, the current solution is optimal.

Algorithm 1 Simplex algorithm main ideas

1: Find a feasible extreme point.
2: If the point is optimal: stop!
3: Find a better neighbour extreme point and go to 2.



Vertices and simplex path [Nielsen, 1999]

Step 2: Checking optimality

Assume that from *Step 1* we have obtained a feasible extreme point with a partition $A = [B|N]$. Writing the system of equalities in terms of this partition, we obtain:

$$B_{x_B} + Nx_N = b$$

B is invertible and, therefore, the basic variables x_B can be expressed in terms of the x_N variables:

$$x_B = B^{-1}b - B^{-1}Nx_N. \tag{4.1}$$

This is called the general solution of the system.

The cost of a solution is given by $z = c^t x = c_B^t x_B + c_N^t x_N$. Using the general solution of the system, this cost can be rewritten as:

$$z = c_B^t(B^{-1}b - B^{-1}Nx_N) + \ c_N^t x_N$$

and, rearranging the terms:

$$z = c_B^t B^{-1}b + \ (c_N^t - c_B^t B^{-1}N)x_N$$

The first term $c_B^t B^{-1}b$ is the value of the basic solution associated with this partition. The vector $\hat{c}_N$ coeficients multiplying the non-basic variables are called the reduced costs. They capture the net change in the objective function of increasing each of the non-basic variables.

For a given partition $x = (x_B, x_N)$, the vector of reduced costs associated with non-basic variables x_N is given by:

$$\hat{c}_N^t = c_N^t - c_B^t B^{-1}N$$

The reduced cost of a specific variable $x_i \in x_N$ is given by:

$$\hat{c}_i = c_i - c_B^t B^{-1}a_i,$$

Optimality Condition: a solution of a maximisation (minimisation) problem is optimal if all the reduced costs are negative (positive).

Example: The Dovetail problem

maximise $3x_1 + 2x_2$

s.t.

$x_1 + x_2 + s_1 = 9,$

$3x_1 + x_2 + s_2 = 18,$

$x_1 + s_3 = 7,$

$x_2 + s_4 = 6,$

$x_1, x_2, s_1, s_2, s_3, s_4 \geq 0.$

$$\max \begin{bmatrix} 3 & 2 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix}$$

s.t.

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} 9 \\ 18 \\ 7 \\ 6 \end{bmatrix}$$

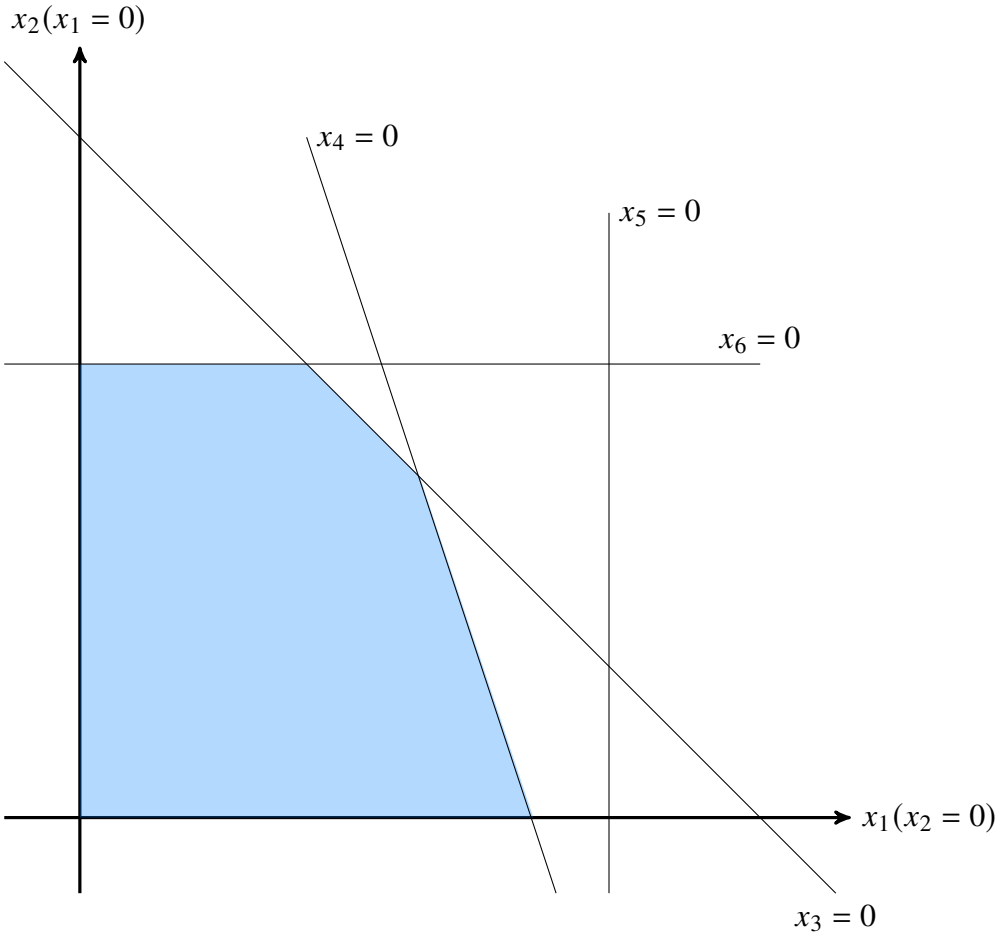
Assume we know point $(x_1, x_2) = (0, 0)$ (out of the basis) is a feasible extreme point, $x_B = [x_3, x_4, x_5, x_6]^t, x_N = [x_1, x_2]^t$, then:

$$B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, N = \begin{bmatrix} 1 & 1 \\ 3 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, c_B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, c_N = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

and

$$\hat{c}_N = c_N^t - c_B^t B^{-1} N = \begin{bmatrix} 3 & 2 \end{bmatrix}$$

and therefore the solution is not optimal since we are maximising and there are positive reduced costs.



Step 3: Finding a better solution

If the current solution of the maximisation (minimisation) program is not optimal, there exists a non-basic variable $x_e \in x_N$ with a reduced cost $c_e > 0 (c_e < 0)$ reduced cost that will enter the basis.

Increasing this variable x_e while maintaining all other non-basic variables at zero produces the following changes in the basic variables:

$$x_B = B^{-1}b - B^{-1}a_e x_e,$$

$$x_B = \hat{x}_B - yx_e,$$

where a_e is the column coefficients in A of variable x_e, $\hat{x}_B$ is the value of the basic solution at the current extreme point and the vector y is defined as $y = B^{-1}a_e$. Note that y is a vector that captures the decrease in the value of each of the basic components as x_e increases.

The maximum possible increase in x_e is given by the first basic variable (if any) to zero out. Let l be the index of this variable:

$$l = \underset{j|x_j \in x_B, y_j > 0}{\operatorname{argmin}} \left\{ \frac{\hat{x}_j}{y_j} \right\}.$$

where $\hat{x}_j$ is the value of the basic variable $x_j \in x_B$ at the extreme point associated with partition $A = [B|N]$.

In the new iteration, the entering variable x_e will occupy the position of x_l in the basis, which will be moved to the set of non-basic variables.

Example: The Dovetail problem (continued)

Finding the neighbouring point and iterating.

Let the entering variable be $x_e = x_1$,

$$x_B = \begin{bmatrix} x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = B^{-1}b = \begin{bmatrix} 9.0 \\ 18 \\ 7 \\ 6 \end{bmatrix}, y = B^{-1}a_1 = \begin{bmatrix} 1.0 \\ 3 \\ 1 \\ 0 \end{bmatrix}$$

The index of the leaving variable is

$$l = \underset{j|x_j \in x_B, y_j > 0}{\operatorname{argmin}} \left\{ \frac{\hat{x}_j}{y_j} \right\} = \underset{j \in \{3,4,5\}}{\operatorname{argmin}} \left\{ \frac{9}{1}, \frac{18}{3}, \frac{7}{1} \right\}$$

which gives x_4 as leaving variable. Notice that we did not use x_6 above as variable x_6 does not change when we increase x_1.

We obtain the new basis with partition $x_B = [x_3, x_1, x_5, x_6]^t, x_N = [x_4, x_2]^t$.

Iteration 2

We stoped at the extreme point associated with the partition $x_B = [x_3, x_1, x_5, x_6]^t, x_N = [x_4, x_2]^t$.

$$B = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, N = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}, c_B = \begin{bmatrix} 0 \\ 3 \\ 0 \\ 0 \end{bmatrix}, c_N \begin{bmatrix} 0 \\ 2 \end{bmatrix}$$

and

$$\hat{c}_N = c_N^t - c_B^t B^{-1} N = \begin{bmatrix} -1.0 & 1.0 \end{bmatrix}$$

We compute the leaving variable:

$$x_B = \begin{bmatrix} x_3 \\ x_1 \\ x_5 \\ x_6 \end{bmatrix} = B^{-1}b = \begin{bmatrix} 3.0 \\ 6.0 \\ 1.0 \\ 6.0 \end{bmatrix}, y = B^{-1}a_2 = \begin{bmatrix} 2/3 \\ 1/3 \\ -1/3 \\ 1 \end{bmatrix}$$

$$l = \underset{j|x_j \in x_B, y_j > 0}{\operatorname{argmin}} \left\{ \frac{\hat{x}_j}{y_j} \right\} = \underset{j \in \{3,1,6\}}{\operatorname{argmin}} \left\{ \frac{3}{2/3}, \frac{6}{1/3}, \frac{6}{1} \right\}$$

which gives x_3 as leaving variable. Notice that we did not use x_5 above as variable x_5 decreases when we increase x_2.

The partition for the next iteration is: $x_B = [x_2, x_1, x_5, x_6]^t, x_N = [x_4, x_3]^t$.

Iteration 3

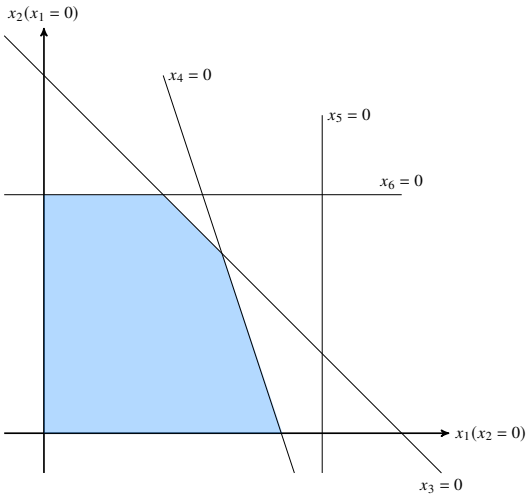
$x_B = [x_2, x_1, x_5, x_6]^t, x_N = [x_4, x_3]^t$.

$$B = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 3 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}, N = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, c_B = \begin{bmatrix} 2 \\ 3 \\ 0 \\ 0 \end{bmatrix}, c_N = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

and

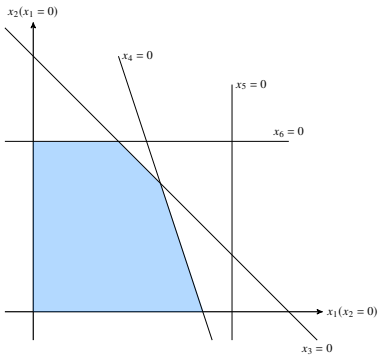
$$\hat{c}_N = \begin{bmatrix} -0.5 & -1.5 \end{bmatrix}$$

Since we are maximising and all reduced costs are negative, we have found the optimal solution. It is given by the extreme point associated with the current partition.



Step 1: Finding a feasible basic solution

The simplex algorithm starts at a feasible extreme point and at each step it finds another neighbouring feasible extreme point with better solution. If no neighbour with better objective function can be found, the current solution is optimal.



Consider the linear program:

maximise $c^t x$
s.t.
$Ax = b, \quad \forall b \geq 0$
$x \geq 0.$

We can easily create an *artificial* feasible solution for a new problem with $y \in \mathbb{R}^n$ additional variables.

maximise $c^t x$
s.t.
$Ax + y = b, \quad \forall b \geq 0$
$x \geq 0.$

A feasible basic solution for the new problem is given by $x_N = x = 0$ and $x_B = y$.

Big M and two-phase approach

Two different ways can be used to remove these undesired variables as the simplex algorithm iterates:

Big-M

The new variables are added with prohibitive costs (M) to the original objetive function:

maximise $c^t x - My$
s.t.
$Ax + y = b$,
$x \geq 0$.

Phase-1

A new problem is solved in order to eliminate the artificial variables from the current solution:

maximise $-y$
s.t.
$Ax + y = b$,
$x \geq 0$.

In both cases, if any artificial variable is present in the optimal solution basis, the original problem is infeasible.

Duality in LP

Consider again the Dovetail problem:

$$\text{maximise } 3x_1 + 2x_2 \tag{4.2}$$
$$\text{s.t.}$$
$$x_1 + x_2 \leq 9, \tag{4.3}$$
$$3x_1 + x_2 \leq 18, \tag{4.4}$$
$$x_1 \leq 7, \tag{4.5}$$
$$x_2 \leq 6, \tag{4.6}$$
$$x_1, x_2 \geq 0.$$

We will develop what is called the dual model associated with this formulation by using the concept of bounds. We will use the constraints to obtain dual bounds for the original model and then write an optimisation problem to obtain the best possible bound.

Consider constraint (4.3) multiplied by 3.

$$3x_1 + 3x_2 \leq 27 \tag{4.7}$$

the lhs of (4.7) is $3x_1 + 3x_2$ and since x_1 and x_2 are non-negative in a feasible solution:

$$z = 3x_1 + 2x_2 \leq 3x_1 + 3x_2 \leq 27$$

and we can conclude that 27 is an upper bound for z (the problem objective function). We could do the same for any other constraint or any linear combination of constraints with non-negative multipliers, as long as the coefficient we get for x_1 and x_2 in the rhs are larger than the original coefficients for these variables in the objective function. For example, consider:

$$(4.4) + (4.6)$$
$$3x_1 + 2x_2 \leq 24$$

This bound is better than the one we had before (closer to the actual optimal solution of the model). What is the linear combination of constraints that will provide the best possible bound ? Let the multipliers be variables $u \geq 0$ of this associated optimisation problem, with constraints:

$$u_1 + 3u_2 + u_3 \geq 3,$$
$$u_1 + u_2 + u_4 \geq 2,$$

$u_1, u_2, u_3, u_4 \geq 0.$

The obtained bound is given by:

$$9u_1 + 18u_2 + 7u_3 + 6u_4$$

The complete optimisation problem of obtaining the best dual bound for Dovetail is therefore:

$$\begin{aligned}
Dual \quad & \text{minimise } 9u_1 + 18u_2 + 7u_3 + 6u_4 && (4.8)\\
& \text{s.t.} \\
& u_1 + 3u_2 + u_3 \geq 3, && (4.9)\\
& u_1 + u_2 + u_4 \geq 2, && (4.10)\\
& u_1, u_2, u_3, u_4 \geq 0.
\end{aligned}$$

This is called the dual model associated with the original (primal) Dovetail model.

The dual model uses the coefficients of the original objective function ($c = [3, 2]^t$) as the resource values and the original resource values ($b = [9, 18, 7, 6]^t$) in its objective function.

Economical interpretation of the dual:

Consider again Dovetail problem and assume another company, Salmonnose, that wants to rent the machine of Dovetail for one year and buy its resources.

Salmonnose wants to decide the minimum price it needs to pay for the rental and for the wood, boxes of long matches and boxes of short matches. For this, it will develop a linear problem based:

Variables:

- u_1 : the price to rent one unit out of the 9 ($\times$ 100000) unities of capacity of Dovetail's machine.
- u_2 : price per unit ($\times$ 100000) of Dovetail's wood.
- u_3 : price per unit ($\times$ 100000) of Dovetail's boxes of long matches.
- u_4 : price per unit ($\times$ 100000) of Dovetail's boxes of short matches.

The price Salmonnose needs to pay is given by:

$$9u_1 + 18u_2 + 7u_3 + 6u_4$$

Consider now the production of long matches. Each unit of long matches require 1 unit of production capacity, 3 units of wood, and 1 unit of boxes for long matches. The price that Salmonnose has to pay cannot be less than the profit of Dovetail by selling this unit of long matches. Therefore,

$$u_1 + 3u_2 + u_3 \geq 3$$

For short matches:

$$u_1 + u_2 + u_4 \geq 2$$

And we obtain the same dual as before:

$$\begin{aligned}
&\text{minimise } 9u_1 + 18u_2 + 7u_3 + 6u_4 \\
&\text{s.t.} \\
&u_1 + 3u_2 + u_3 \geq 3, \\
&u_1 + u_2 + u_4 \geq 2, \\
&u_1, u_2, u_3, u_4 \geq 0.
\end{aligned}$$

Canonical forms of primal and dual

$$
\begin{aligned}
\textit{Primal} \quad & \text{maximise } c^t x \\
& \text{s.t.} \\
& Ax \leq b, \\
& x \geq 0.
\end{aligned}
$$

$$
\begin{aligned}
\textit{Dual} \quad & \text{minimise } b^t u \\
& \text{s.t.} \\
& A^t u \geq c, \\
& u \geq 0.
\end{aligned}
$$

Note that each constraint in the primal will give origin to a constraint in the dual and each constraint in the primal will originate a variable in the dual. In order to obtain the dual model associated with a primal model that is not in canonical form, you can represent it in canonical form and do the conversion as above or use the following "conversion table".

Primal LP	Dual LP
max	min
Constraint i	Variable i
$\leq$ form	$y_i \geq 0$
= form	$y_i \in \mathbb{R}(free)$
Variable j	Constraint j
$x_j \geq 0$	$\geq$ form
$x_j \in \mathbb{R}(free)$	= form
Costs	Resources
Resources	Costs

The labels "primal" and "dual" are relative, as the dual of the dual model is the original primal model.

Finding the dual model:

Convert the following LP into its dual. Implement and solve both models, what do you observe?

maximise $5x_1 + 4x_2$
s.t.
$3x_1 - 8x_2 \geq -6,$
$x_1 + 6x_2 = -5,$
$8x_1 + 3x_2 = 10$
$x_2 \geq 0, x_1 \in \mathbb{R}$

We write the primal model as:

maximise $5x_1 + 4x_2$
s.t.
$-3x_1 + 8x_2 \leq 6,$
$x_1 + 6x_2 = -5,$
$8x_1 + 3x_2 = 10$
$x_2 \geq 0, x_1 \in \mathbb{R}$

and then use the conversion table directly:

minimise $6u_1 - 5u_2 + 10u_3$
s.t.
$-3u_1 + u_2 + 8u_3 = 5,$
$8u_1 + 6u_2 + 3u_3 \geq 4,$
$u_1 \geq 0, u_2, u_3\ free$

The primal model of this exercise is infeasible and the dual model is unbounded.

Weak duality

The value of any feasible solution to the max problem (either you call it primal or dual) will always be a lower bound on the min problem. Conversely, the value of any feasible solution to the min problem (either you call it primal or dual) will always be an upper bound on the max problem.

This is known as weak duality and can be easily shown by remembering that any feasible solution to the primal has $x \geq 0$ and $Ax \leq b$ and any feasible solution to the dual has $u \geq 0$ and $A^t u \geq c$. Therefore:

$$c^t x \leq (A^t u)^t x = u^t A x \leq u^t b = b^t u \qquad \square$$

From which we can also conclude that a primal-dual solution pair with same objective function must be primal and dual optimal.

Possibilities of solutions for primal-dual pairs

There are only 4 possibilities for a dual pair of LPs:

1. their optimal values are finite and equal,
2. the primal is unbounded and the dual is infeasible,
3. the dual is unbounded and the primal is infeasible,
4. both are infeasible.

Example:

What can you say about the feasibility of the problem below. What about its dual ?

$$P = \max\{x + y : \quad x - y \leq -1, \quad y - x \leq -1, \text{ and } \quad x, y \geq 0\}.$$

Strong duality

The strong duality theorem states that if a model has an optimal solution, its dual also has an optimal solution and their values are the same and the dual can be obtained from the optimal primal basis. We will prove the following version of the theorem:

Theorem 3 *Strong Duality: If a primal model (P) is feasible and bounded, its dual (D) also has an optimal solution and both have the same objective function.*

Proof: consider the primal problem in canonical form, with the addition of slack variables. If the primal is feasible and bounded, there is an optimal extreme point and we can express it as:

$$x^* = \begin{bmatrix} x_B^* \\ x_N^* \end{bmatrix}, x_N^* = 0$$

Consider a tentative solution $y = (B^{-1})^t c_B$, we first show that this solution is feasible:

(i) Feasibility: we need to show that $A^t y \geq c$. This can be expressed as:

$$A^t y = \begin{bmatrix} B^t \\ N^t \end{bmatrix} y \geq \begin{bmatrix} c_B \\ c_N \end{bmatrix}$$

Since by hypothesis $y = (B^{-1})^t c_B$, we have:

$$A^t y = \begin{bmatrix} B^t (B^{-1})^t c_B \\ N^t (B^{-1})^t c_B \end{bmatrix} = \begin{bmatrix} c_B \\ N^T (B^{-1})^t c_B \end{bmatrix} \geq \begin{bmatrix} c_B \\ c_N \end{bmatrix}$$

where the first part is an equality and the second part holds since the partition is optimal and therefore the reduced costs given by $c_N - N^t (B^{-1})^t c_B$ are non-positive.

(ii) Optimality:

$$b^t y = b^t (B^{-1})^t c_B = (B^{-1} b) c_B = (x_B^*)^t c_B = c^t x^*$$

More economical interpretation:

Consider the following primal/dual pair:

$$
\begin{aligned}
Primal \quad & \text{minimise } c^t x \\
& \text{s.t.} \\
& Ax \geq b, \\
& x \geq 0.
\end{aligned}
$$

$$
\begin{aligned}
Dual \quad & \text{maximise } b^t u \\
& \text{s.t.} \\
& A^t u \leq c, \\
& u \geq 0.
\end{aligned}
$$

Given a basis B for the primal, we saw earlier that the objective function, z, of the primal can be written as:

$$z = c_B^t B^{-1} b + \ (c_N^t - c_B^t B^{-1} N) x_N$$

Assume this is an optimal basis and consider a slight change in one of the coefficients of b, b_i. The change is small enough such that the current basis remains optimal. The change in the objective function with respect to the change in b_i can be written as:

$$\frac{\partial z^*}{\partial b} = c_B^t B^{-1} b$$

Let B_i^{-1} be the i^{th} column of B^{-1}:

$$\frac{\partial z^*}{\partial b_i} = c_B^t B_i^{-1} = u_i^*$$

Therefore, the rate of change in the optimal objective function for a small change in b_i is given by u_i^*.

Dovetail/Salmonose example continued

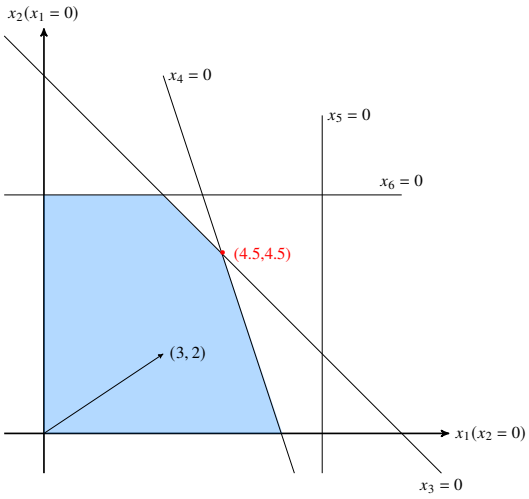
Obtain the dual variables for the problem. Find the change in the objective function for different values of the first resource.

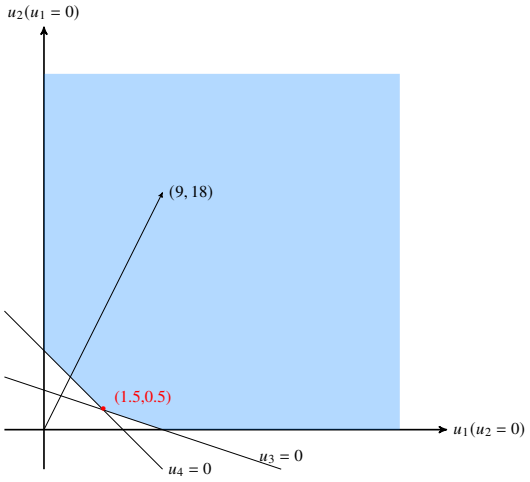
```
import gurobipy as gp
from gurobipy import GRB

#Dovetail primal
m = gp.Model("dovetail")
c = [3,2]
A = [[1, 1], [3,1], [1,0], [0,1]]
b = [9,18,7,6]
N = range(len(c))
M = range(len(b))

x = m.addVars(N, name='x')

m.setObjective( gp.quicksum(x[i]*c[i] for i in N), GRB.MAXIMIZE)
cons = m.addConstrs( gp.quicksum(A[j][i]*x[i] for i in N) <= b[j] for j in M
    )
m.optimize()

# Display primal optimal values of decision variables
prim = m.getAttr("x", m.getVars())
print("Primal solution: ", prim)

# Display primal optimal values of decision variables
duals = m.getAttr("Pi", m.getConstrs())
print("Dual solution:", duals)

# Display optimal solution value
print('Total profit: ', m.objVal)
```

Optimal solution:

```
Objective:  22.5
Primal solution
x[0] = 4.5
x[1] = 4.5
Dual solution
u[0] =  1.5
u[1] =  0.5
u[2] =  0.0
u[3] =  0.0
```

The pictures below show the feasible space of the primal on variables x_1, x_2 and the projection of the feasible space of the dual on variables u_1, u_2 for u_3, u_4 fixed.



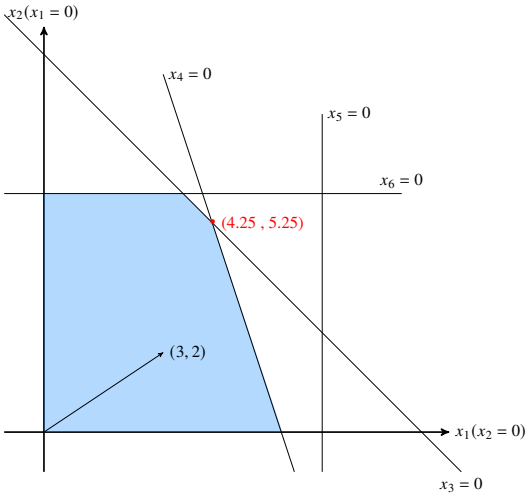


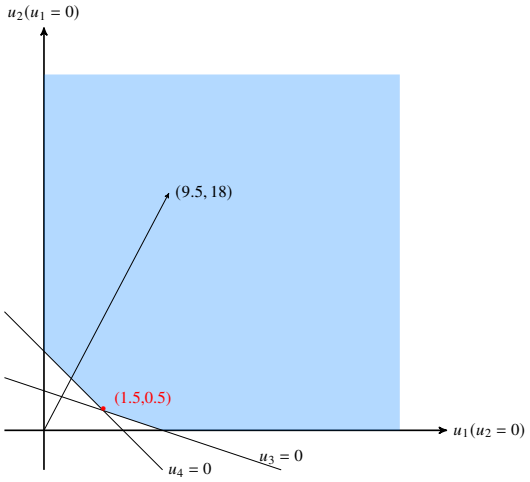
Changing the right hand side to 9.5 (i.e., increasing .5 unit of the first resource b_1):

```
#Slightly changing b[0] from 9 to 9.5:
cons[0].rhs = 9.5
m.optimize()


# Display primal optimal values of decision variables
prim = m.getAttr("x", m.getVars())
print("Primal solution: ", prim)

# Display primal optimal values of decision variables
duals = m.getAttr("Pi", m.getConstrs())
print("Dual solution:", duals)

# Display optimal solution value
print('Total profit: ', m.objVal)
```

```
Objective:  23.25
Primal solution
x[0] = 4.25
x[1] = 5.25
Dual solution
u[0] =  1.5
u[1] =  0.5
u[2] =  0.0
u[3] =  0.0
```

Note that the active constraints remains the same. Therefore, the objective function, as expected increases by $\Delta b_1 \times u_1 = 0.5 * 1.5 = 0.75$.





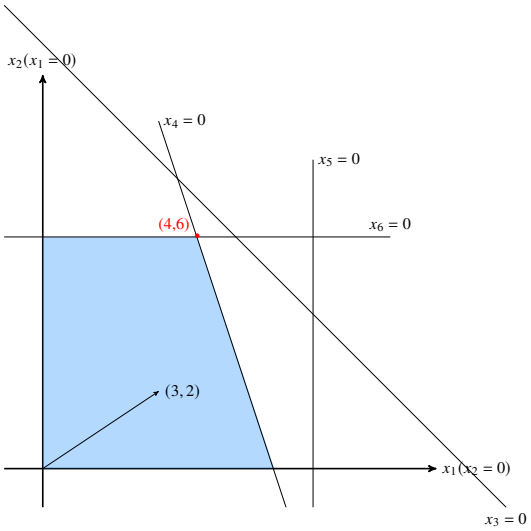
This interpretation is valid only for small changes in b (the basis needs to remain the same). If we increase the right hand side to a value such that the basis changes, we can not use the dual variables to obtain the change. For example, increasing b_1 to 12:

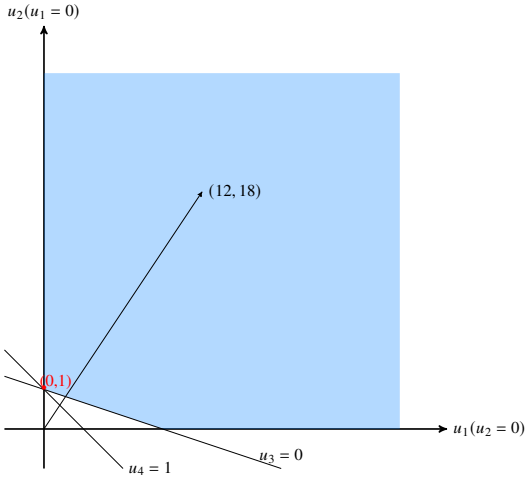
```
# changing b[0] to 12:
cons[0].rhs = 12
m.optimize()


# Display primal optimal values of decision variables
prim = m.getAttr("x", m.getVars())
print("Primal solution: ", prim)

# Display primal optimal values of decision variables
duals = m.getAttr("Pi", m.getConstrs())
print("Dual solution:", duals)

# Display optimal solution value
print('Total profit: ', m.objVal)
```

```
Objective:  24.0
Primal solution
x[1] = 4.0
x[2] = 6.0
Dual solution
u[1] =  0.0
u[2] =  1.0
u[3] =  0.0
u[4] =  1.0
```

Note that the increase in the objective function with respect to the original problem is $24 - 22.5 = 1.5 \neq \Delta b_1 \times u_1 = 3 \times 1.5 = 4.5$

Primal and dual feasible spaces:


$x_2(x_1 = 0)$
$x_4 = 0$
$x_5 = 0$
(4,6)
$x_6 = 0$
(3, 2)
$x_1(x_2 = 0)$
$x_3 = 0$


$u_2(u_1 = 0)$
(12, 18)
(0,1)
$u_1(u_2 = 0)$
$u_3 = 0$
$u_4 = 1$

Chapter 5

Branch and Bound

A generic approach for MIP

The generic mixed-integer programming solution technique:

Consider the following generic optimisation problem:

$$Z = \max \{c(x) : x \in X \subseteq \mathbb{R}^n\},$$

which can be used, for example, to represent any mixed-integer programming model. In this case, the set X is defined by linear constraints with additional integrality constraints. The function $c(x)$ must be linear and can therefore be written as $c^t x$.

The overall strategy we will use to solve this model is to generate guaranteed bounds for the value of Z. If, we can find an algorithm that ensures the obtention of a sequence of lower bounds $\underline{Z}$ and upper bounds $\overline{Z}$ to the value of Z, respecting: $\underline{Z}_1 < \underline{Z}_2 < \cdots < \underline{Z}_t \leq Z$ and $\overline{Z}_1 > \overline{Z}_2 > \cdots > \overline{Z}_s \geq Z$ we will eventually reach a point where we know the value of Z within any desired precision ϵ.

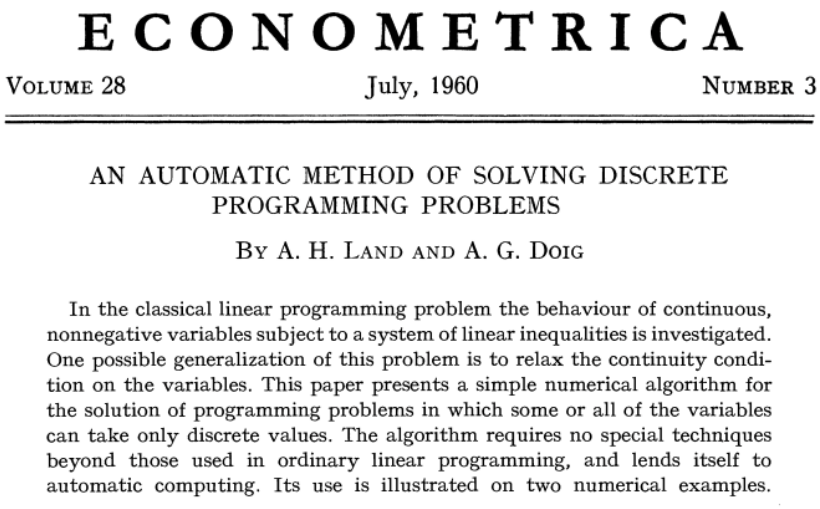
$$\overline{Z}_s - \underline{Z}_t \leq \epsilon$$

Note that if the problem was $Z = \min \{c(x) : x \in X \subseteq \mathbb{R}^n\}$, $\overline{Z}_i$ above would be a lower bound while $\underline{Z}_i$ would be an upper bound. To avoid confusion, we will talk about primal bounds (given by a feasible solution) and dual bounds (given by a relaxation).

Branch and bound

Branch and bound central idea is to deal with decision making by branching in the feasible space and solving smaller problems. The bound part consists in eliminating as much as possible from the resulting tree.

Ailsa Land and Alison Harcourt (née Doig) proposed the Branch-and-Bound method for mixed-integer linear programming in their paper 'An Automatic Method of Solving Discrete Programming Problems' [Land and Doig, 1960].



ECONOMETRICA

VOLUME 28 July, 1960 NUMBER 3

AN AUTOMATIC METHOD OF SOLVING DISCRETE PROGRAMMING PROBLEMS

BY A. H. LAND AND A. G. DOIG

In the classical linear programming problem the behaviour of continuous, nonnegative variables subject to a system of linear inequalities is investigated. One possible generalization of this problem is to relax the continuity condition on the variables. This paper presents a simple numerical algorithm for the solution of programming problems in which some or all of the variables can take only discrete values. The algorithm requires no special techniques beyond those used in ordinary linear programming, and lends itself to automatic computing. Its use is illustrated on two numerical examples.

Land and Doig seminal paper

Alison Harcourt, AO

In October 2018 Harcourt was named as 2019 Senior Victorian Australian of the Year. In early December 2018, the University of Melbourne awarded Harcourt with an honorary Doctor of Science degree.

In June 2019, Harcourt was made an Officer of the Order of Australia in recognition of her "distinguished service to mathematics and computer science through pioneering research and development of integer linear programming"

Alison is a staff member of our school. In 2019, she came to our lectures and talked about the time she developed the method with Ailsa.

A great and simple to read article in honour of Alison Harcourt is:

"A seminal contribution of Ailsa Land and Alison Doig Harcourt to the field of mathematical programming" by Gilbert Laporte [Laporte, 2024].

Divide and conquer

Enumerate possible solutions by 'branching' on decisions (fixing or restricting their values).

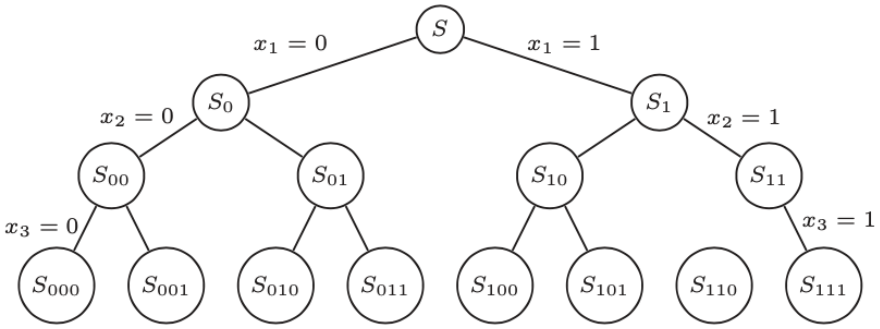


Figure 7.1 Binary enumeration tree.

Binary enumeration tree [Wolsey, 2020]

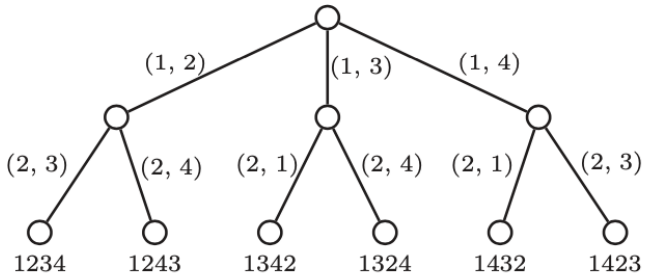


Figure 7.2 TSP enumeration tree.

Enumeration tree for the TSP [Wolsey, 2020]

Prunning the B&B tree

Reduce the number of branches that need to be explored by doing a top-down approach and using primal bounds and dual bounds available at each step.

We will mostly focus on linear programming-based branch-and-bound algorithms, which obtains the dual bounds with Linear Programming relaxations.

The key concept here is the obtention of the best bounds available at any given point and use them to prune the tree.

Proposition 7.2 [Wolsey, 2020]: Let $S = S_1 \cup \cdots \cup S_K$ be a decomposition of S into smaller sets, and let $Z^k = \max\{cx : x \in S_k\}$ for $k = 1, \ldots, K$, $\bar{Z}^k$ be an upper bound on Z^k and $\underline{Z}^k$ be a lower bound on Z^k. Then $\bar{Z} = \max_k \bar{Z}^k$ is an upper bound on Z and $\underline{Z} = \max_k \underline{Z}^k$ is a lower bound on Z.

Intuition: You are maximising, so the best lower bound is given by the best (with maximum objective function) feasible solution you have available (the maximum value among all feasible solutions you have found on your tree). You can guarantee that your optimal solution is not lower than that value (otherwise the current incumbent solution would be the optimal).

The best upper bound is the maximum promise you have available as, eventually, it could lead to a feasible solution with that value.

What happens for a minimisation problem?

In Land and Doig's branch-and-bound, linear programming is used to produce bounds on the integer problem. For a minimisation problem, this bound is a lower bound (the optimal integer solution value can not be lower than the LP relaxation value). For a maximisation problem, this bound is an upper bound (the optimal integer solution value can not be larger than the linear programming relaxation).

Another bound used is the one given by a feasible integer solution. It provides an upper bound if we are in a minimisation problem and a lower bound if we are in a maximisation problem.

We start at the root node and if an integer solution is found.

Terminology

- the act of bounding and then branching is called processing,
- a subproblem that has not yet been considered is called a candidate for processing,
- the set of candidates for processing is called the candidate list,
- going back on the path from a node to its root is called backtracking.

Pseudo-code

Below is a pseudo-code for a B&B algorithm.

A primal bound is given by a feasible solution and represents a lower bound (upper bound) for a minimisation (maximisation) problem. A dual bound is the bound given by the linear relaxation and represents an upper bound (lower bound) for a minimisation (maximisation) problem.

- S: Set of open nodes that need to be explored,
- LB = best current lower bound,
- $LP(i)$ = relaxed problem at node i
- UB_i = optimal solution value of $LP(i)$.

In the algorithm below, we consider a maximisation problem.

Algorithm 2 LP based B&B algorithm (Max)

1: $S = \{\text{root node}\}$
2: $LB = -\infty$ (no feasible solution)
3: Select one node $i \in S$, if S is empty stop $\rightarrow$ LB is the optimal solution value (if $-\infty$, the problem is infeasible)
4: Remove i from S and solve $LP(i)$ to obtain UB_i.
5: Prune node i if:
6: Case 1: $LP(i)$ is infeasible (pruning by infeasibility):
7: Case 2: $UB_i \leq LB$ (pruning by quality)
8: Case 3: The solution of $LP(i)$ is integer (pruning by integrality). In this case, if $UB_i \geq LB$ then $LB = UB_i$ (update best primal bound)
9: If no pruning:
10: Select a variable with fractional value in the optimal solution of $LP(i)$, branch on it and add the child nodes to S
11: Go back to step 3.

BB tree for a knapsack problem

Consider the following knapsack problem and solve using the branch and bound algorithm.

Integer Knapsack: $z = \max x_1 + x_2$

s.t.

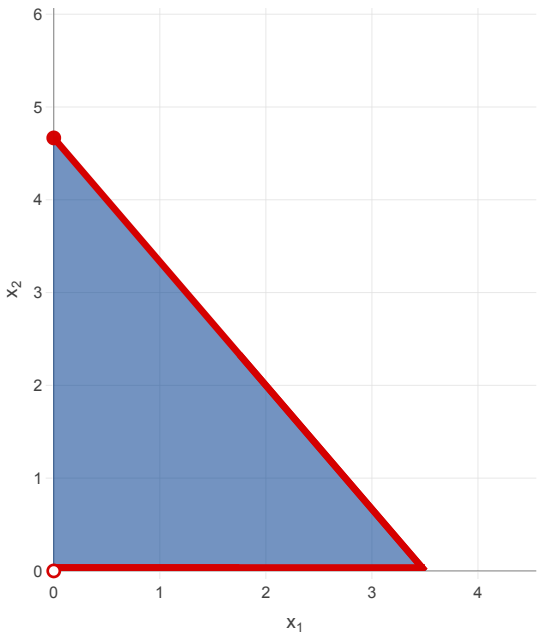
$4x_1 + 3x_2 \leq 14$

$x_1, x_2 \geq 0,$ and integer.

B&B example in GILP

In [1]:
```
#Install package
# !pip install gilp

#Imports
import gilp
from gilp.simplex import LP
from gilp.visualize import simplex_visual
import numpy as np


A = np.array([[4, 3]])
b = np.array([14])
c = np.array([1,1])

lp = LP(A,b,c)

nodes = gilp.bnb_visual(lp)
```

In [2]:
```
nodes[0]
```

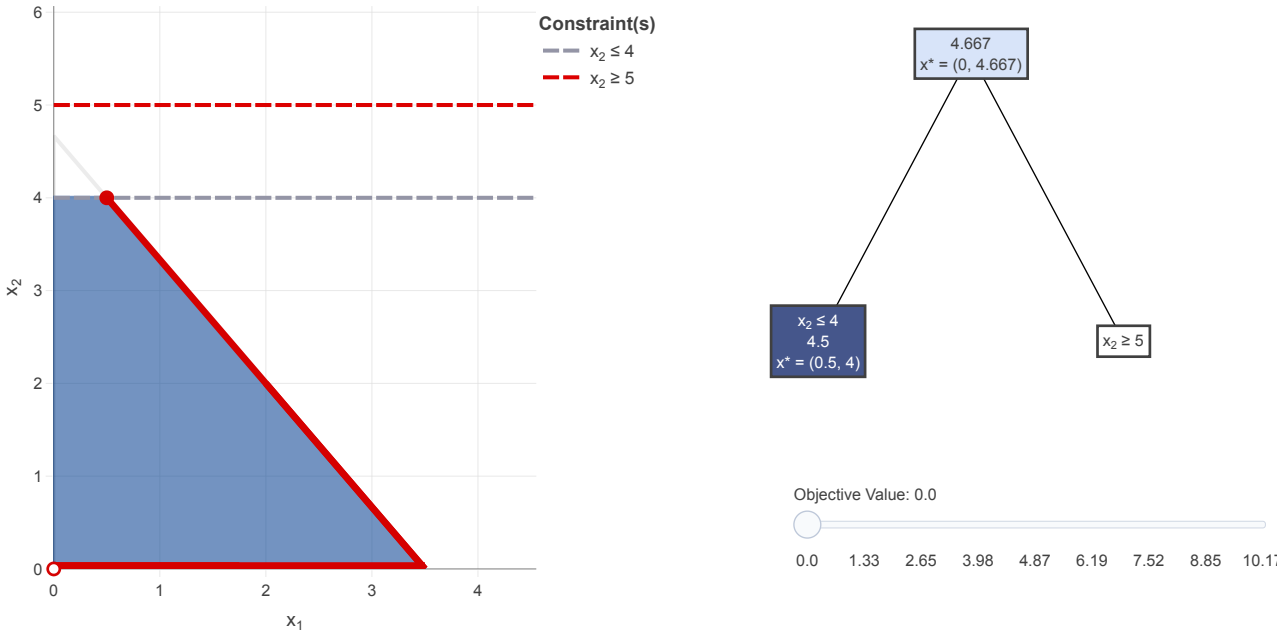


4.667
x* = (0, 4.667)

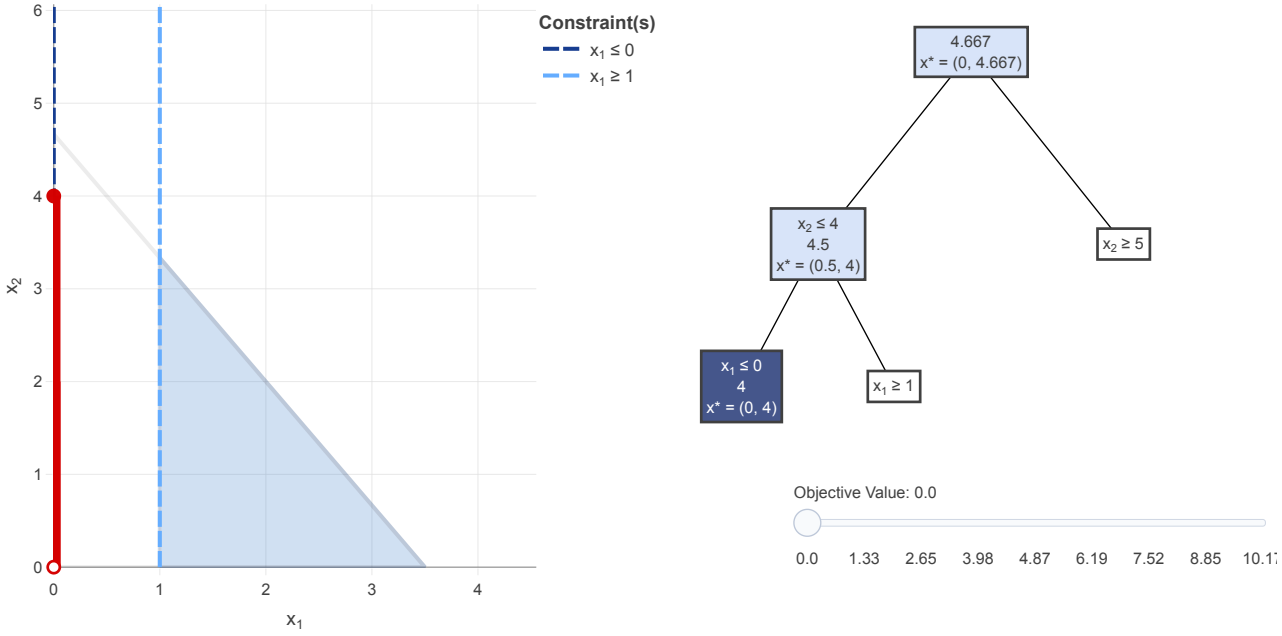
Objective Value: 0.0

0.0 1.33 2.65 3.98 4.87 6.19 7.52 8.85 10.17

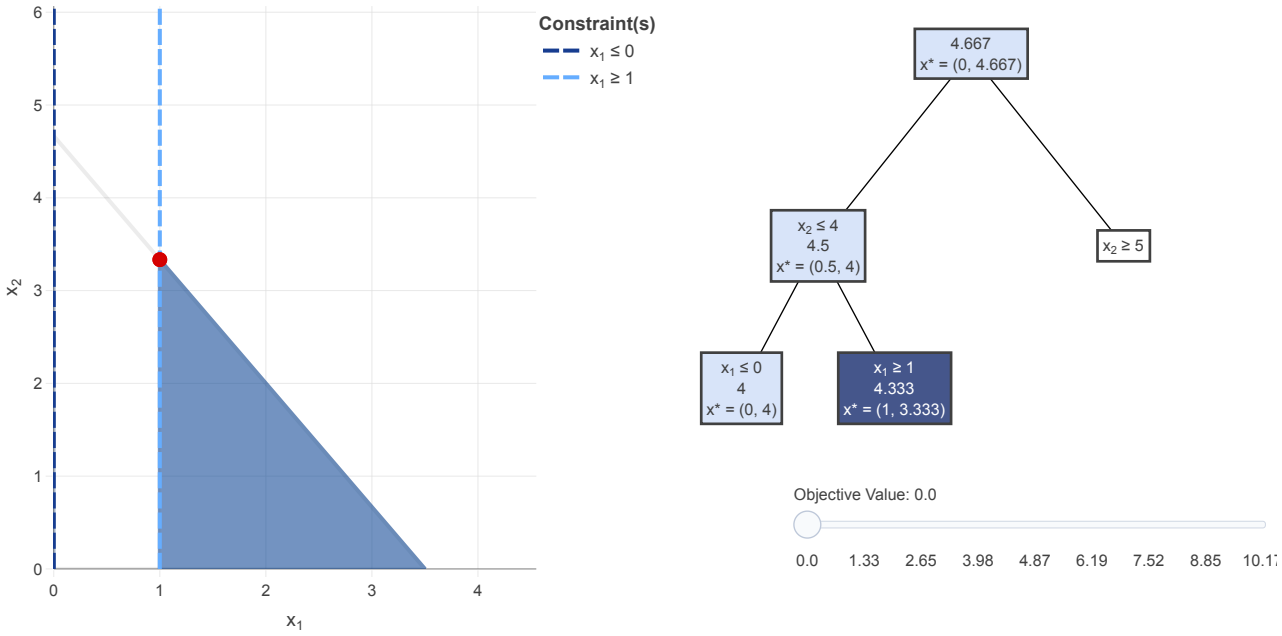
In [3]:
```
nodes[1]
```



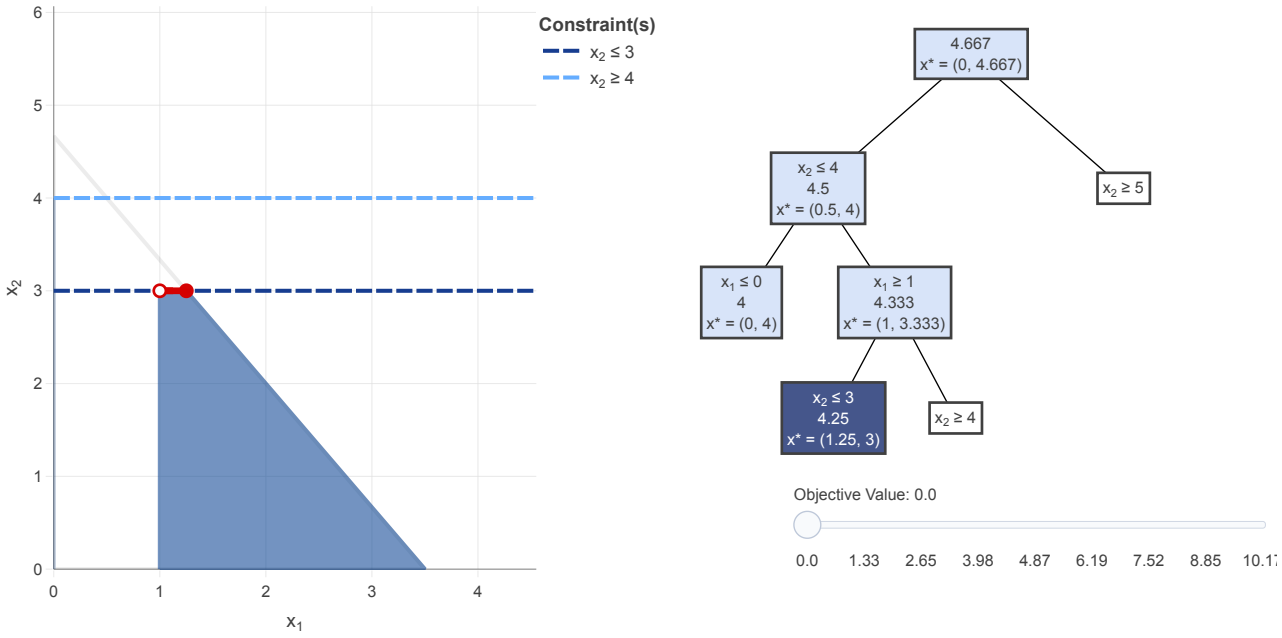
In [4]:
```
nodes[2]
```



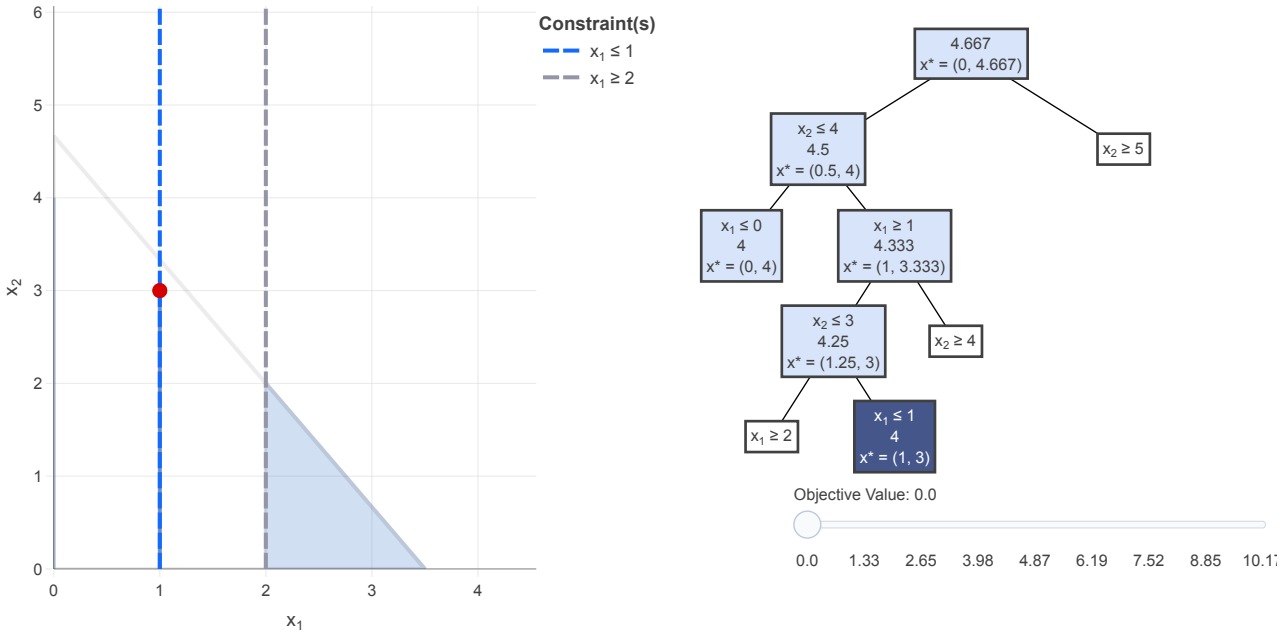
In [5]:
```
nodes[3]
```



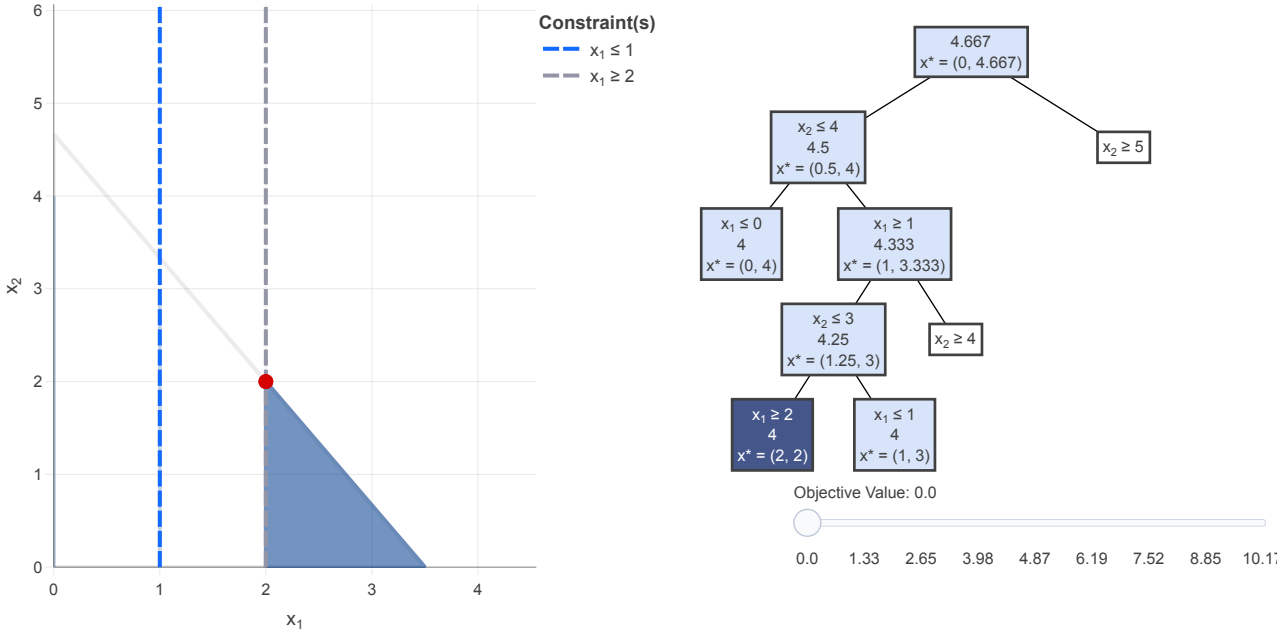
In [6]:
```
nodes[4]
```



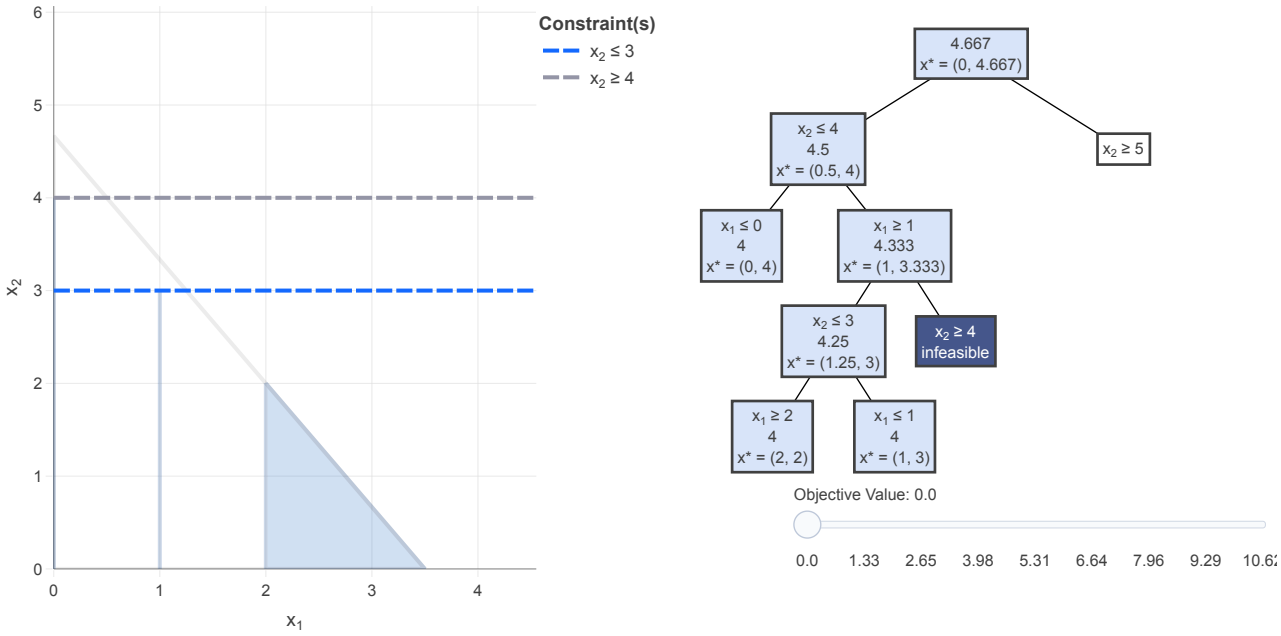
In [7]:
```
nodes[5]
```



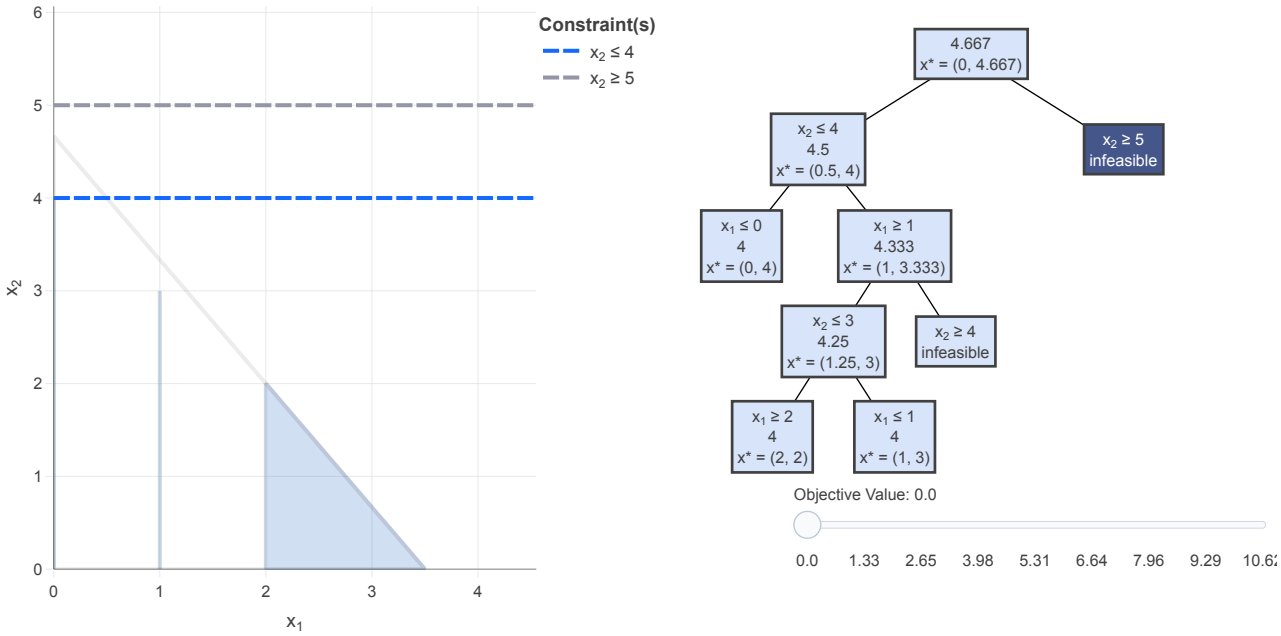
In [8]:
```
nodes[6]
```



In [9]:
```
nodes[7]
```



In [10]:
```
nodes[8]
```



Non-LP based branch and bound

All we need to implement the branch and bound algorithm is a bound. It does not need to be the LP bound. We can use problem-specific bounds if they are known.

A lower bound for the machine scheduling problem

n jobs must be scheduled for processing on a machine. Each job i requires a specified processing time p_i, and has a due date d_i, by which it should be completed, $i = 1, \ldots, n$.

Assume that you set job i_n to be the last job to be executed. Find a lower bound for the solution.

Suppose we know that job i_n is the last job in the schedule. Then we know that job in must be scheduled to start at time:

$$\sum_{i \neq i_n} p_i$$

and so is overdue by

$$\max\{p_{i_n} + \sum_{i \neq i_n} p_i - d_{i_n}, 0\}$$

which gives a lower bound on the solution with job i_n at the last position.

This bound suggests a recursive procedure for finding an optimal schedule by instantiating the last job, then the second-last, and so on.

Branch and bound tree

Consider the following data for the machine scheduling problem and solve using branch-and-bound with the lower bound defined above.

Job	Processing time	Due date
1	6	8
2	4	4
3	5	12
4	8	16

Computational implementation

A number of computational decisions need to be taken for the implementation of the branch and bound algorithm. We will briefly explore these questions when we study the Branch and Cut algorithm. Before that, we will examine the Cutting planes method.

Chapter 6

Cutting Planes

A generic approach for MIP

The generic mixed-integer programming solution technique:

Consider the following generic optimisation problem:

$$Z = \max \{c(x) : x \in X \subseteq \mathbb{R}^n\},$$

which can be used, for example, to represent any mixed-integer programming model. In this case, the set X is defined by linear constraints with additional integrality constraints. The function $c(x)$ must be linear and can therefore be written as $c^t x$.

The overall strategy we will use to solve this model is to generate guaranteed bounds for the value of Z. If, we can find an algorithm that ensures the obtention of a sequence of lower bounds $\underline{Z}$ and upper bounds $\overline{Z}$ to the value of Z, respecting: $\underline{Z}_1 < \underline{Z}_2 < \cdots < \underline{Z}_t \leq Z$ and $\overline{Z}_1 > \overline{Z}_2 > \cdots > \overline{Z}_s \geq Z$ we will eventually reach a point where we know the value of Z within any desired precision ϵ.

$$\overline{Z}_s - \underline{Z}_t \leq \epsilon$$

Note that if the problem was $Z = \min \{c(x) : x \in X \subseteq \mathbb{R}^n\}$, $\overline{Z}_i$ above would be a lower bound while $\underline{Z}_i$ would be an upper bound. To avoid confusion, we will talk about primal bounds (given by a feasible solution) and dual bounds (given by a relaxation).

Good and bad formulations

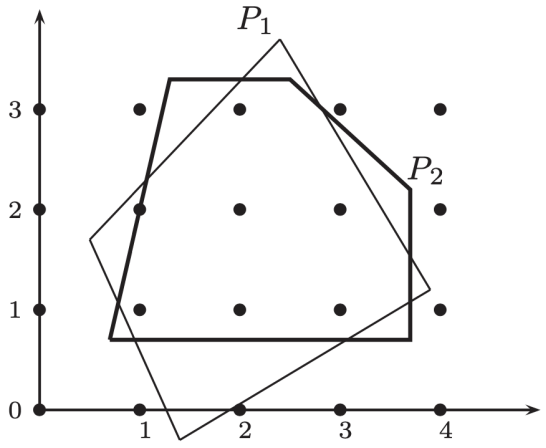
Definition: A subset of $\mathbb{R}^n$ described by a finite set of linear constraints. $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ is a polyhedron.

Definition: A polyhedron $P \subseteq \mathbb{R}^{n+p}$ is a formulation for a set $X \subseteq \mathbb{Z}^n \times \mathbb{R}^p$ if and only if $X = P \cap (\mathbb{Z}^n \times \mathbb{R}^p)$.

Example: two different formulations for the set:

$$X = \{(1,1), (2,1), (3,1), (1,2), (2,2), (3,2), (2,3)\}$$

.



Different models for same IP [Wolsey, 2020]

Strengthening the UFL formulation

Can we make our UFL model get closer to its convex hull?

$$\text{minimise} \quad \sum_{i \in M} \sum_{j \in N} c_{ij} x_{ij} + \sum_{j \in N} f_j x_j$$

s.t.

$$\sum_{j \in N} y_{ij} = 1, \quad \forall i \in M,$$

$$\sum_{i \in M} y_{ij} \leq m x_j, \quad \forall j \in N$$

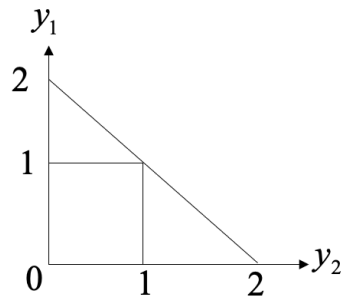
$$y_{ij} \geq 0 \quad \forall i \in M, j \in N,$$

$$x_j \in \{0, 1\} \quad \forall j \in N.$$

Consider the disaggregated version of the second set of constraints:

$$y_{ij} \leq x_j, \quad j \in N, i \in M$$

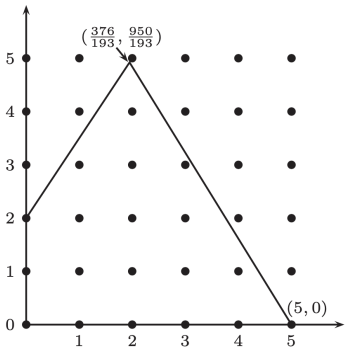
Example: Consider one facility and two clients.



Linear programming relaxation for aggregated and disaggregated versions of UFL constraints

Convex Hull

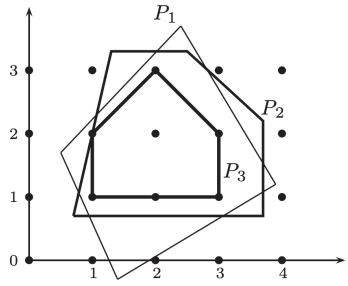
The solution of a LP can be very different from the equivalent IP.



LP and IP solutions [Wolsey, 2020]

Definition: Given a set $X \subseteq \mathbb{R}^n$, the convex hull of X, denoted $\text{conv}(X)$, is defined as follows: $\text{conv}(X) = \left\{x : x = \sum_{i=1}^{t} \lambda_i x^i, \sum_{i=1}^{t} \lambda_i = 1, \lambda_i \geq 0 \text{ for } i = 1, \ldots, t \text{ over all finite subsets } \left\{x^1, \ldots, x^t\right\} \text{ of } X\right\}$.

Figure 1.6 The ideal formulation.



The convex hull formulation [Wolsey, 2020]

Valid inequalities

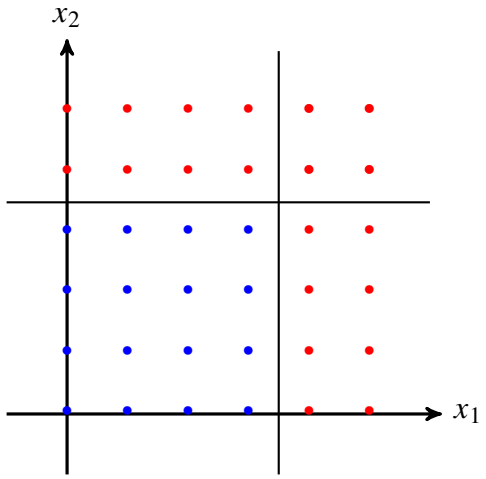
We are following here the notation of chapter 8 of [Wolsey, 2020]:

Consider the general integer program

$$\max_x \{cx : x \in \mathcal{X}\},$$

with $\mathcal{X} = \{x : Ax \leq b, x \in \mathcal{Z}_+^n\}$.

An inequality $\pi x \leq \pi_0$ is a valid inequality for $\mathcal{X} \subseteq \mathcal{R}_+^n$ if $\pi x \leq \pi_0$ for all $x \in \mathcal{X}$.



If $\mathcal{X} = \{x : \tilde{A}x \leq \tilde{b}, x \in \mathcal{R}_+^n\}$ is the convex hull associated with the integer program, all inequalities of the convex hull are valid inequalities for the integer program.

Pure 0-1 set:

Consider the $0-1$ knapsack set:

$$X = \left\{x \in B^5 : 3x_1 - 4x_2 + 2x_3 - 3x_4 + x_5 \leq -2\right\}$$

If $x_2 = x_4 = 0$, the lhs (left-hand side) $= 3x_1 + 2x_3 + x_5 \geq 0$ and the rhs (right-hand side) $= -2$, which is impossible. So all feasible solutions satisfy the valid inequality $x_2 + x_4 \geq 1$.

If $x_1 = 1$ and $x_2 = 0$, the lhs $= 3 + 2x_3 - 3x_4 + x_5 \geq 3 - 3 = 0$ and the rhs $= -2$, which is impossible, so $x_1 \leq x_2$ is also a valid inequality.

Mixed 0-1 set:

Consider the set:

$$X = \left\{(x, y) : x \leq 9999y, 0 \leq x \leq 5, y \in B^1\right\}$$

It is easily checked that the inequality

$$x \leq 5y$$

is valid because $X = \{(0, 0), (x, 1) \text{ with } 0 \leq x \leq 5\}$. As X only involves two variables, it is possible to represent X graphically, so it is easy to check that the addition of the inequality $x \leq 5y$ gives us the convex hull of X

Integer rounding:

Consider the integer region $X = P \cap Z^4$ where

$$P = \left\{x \in R_+^4 : 13x_1 + 20x_2 + 11x_3 + 6x_4 \geq 72\right\}$$

Dividing by 11 gives the valid inequality for P :

$$\frac{13}{11}x_1 + \frac{20}{11}x_2 + x_3 + \frac{6}{11}x_4 \geq 6\frac{6}{11}$$

As $x \geq 0$, rounding up the coefficients on the left to the nearest integer gives $2x_1 + 2x_2 + x_3 + x_4 \geq \frac{13}{11}x_1 + \frac{20}{11}x_2 + x_3 + \frac{6}{11}x_4 \geq 6\frac{6}{11}$, and so we get a weaker valid inequality for P :

$$2x_1 + 2x_2 + x_3 + x_4 \geq 6\frac{6}{11}$$

As x is integer and all the coefficients are integer, the lhs must be integer. An integer that is greater than or equal to $6\frac{6}{11}$ must be at least 7 , and so we can round the rhs up to the nearest integer giving the valid inequality for X :

$$2x_1 + 2x_2 + x_3 + x_4 \geq 7$$

Mixed integer rounding:

Consider the set

$$X = \left\{(x, y) : x \leq 10y, 0 \leq x \leq 14, y \in Z_+^1\right\}$$

It is not difficult to verify the validity of the inequality $x \leq 6 + 4y$, or written another way, $x \leq 14 - 4(2 - y)$. In Figure 8.1 we represent X graphically, and see that the addition of the inequality $x \leq 6 + 4y$ gives the convex hull of X

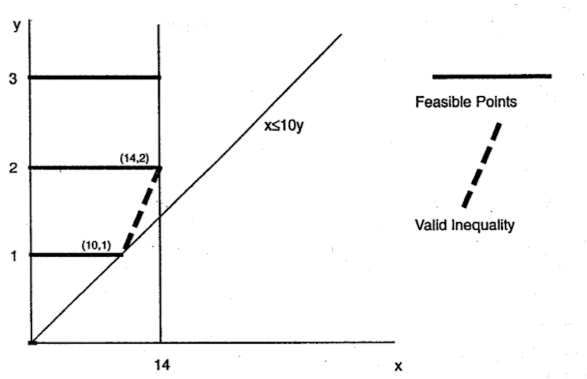


Fig. 8.1 Mixed integer inequality

From [Wolsey, 2020]

For the general case, when C does not divide b, and

$$X = \left\{(x, y) : x \leq Cy, 0 \leq x \leq b, y \in Z_+^1\right\}$$

one obtains the valid inequality $x \leq b - \gamma(K - y)$ where $K = \lceil \frac{b}{C} \rceil$ and $\gamma = b - \left(\lceil \frac{b}{c} \rceil - 1\right) C$

In Gurobi:

MIRCuts

MIR cut generation

Type:	int
Default value:	-1
Minimum value:	-1
Maximum value:	2

Controls Mixed Integer Rounding (MIR) cut generation. Use 0 to disable these cuts, 1 for moderate cut generation, or 2 for aggressive cut generation. The default -1 value chooses automatically. Overrides the Cuts parameter.

Note: Only affects mixed integer programming (MIP) models

Cover inequalities:

For a knapsack problem constraint:

A set $C \subseteq I$ is a cover if $\sum_{i \in C} w_i > K$.

The cover is said to be minimal if $C \backslash \{j\}$ is not a cover for all $j \in C$, i.e., if I remove any element from a minimal cover, it is no longer a cover.

An extended cover $E(C) = C \cup \{j \in I : \ a_j \geq a_i, \forall i \in C\}$, i.e., an extended cover extends a cover by adding all items that have weights larger than or equal to the weight of any element in the original cover.

A knapsack cover inequality is an inequality of the form:

$$\sum_{i \in C} x_i \leq |C| - 1$$

An extended knapsack cover inequality is an inequality of the form:

$$\sum_{i \in E(C)} x_i \leq |C| - 1$$

Note that the right-hand side remains the same as in the inequality for the original cover. This makes this constraint stronger.

Example: cover cuts

Consider the following knapsack problem constraint:

$$X = \left\{x \in \{0,1\}^5 : 5x_1 + 7x_2 + 4x_3 + 4x_4 + 4x_5 \leq 14\right\}$$

Find a cover inequality and an extended cover inequality.

The following is a cover inequality for $C = \{1, 3, 4, 5\}$

$$x_1 + x_3 + x_4 + x_5 \leq 3$$

The following is an extended cover set $E(C) = \{1, 2, 3, 4, 5\}$ as

$$w_2 \geq w_i, i = 1, 3, 4, 5$$

:

$$x_1 + x_2 + x_3 + x_4 + x_5 \leq 3$$

In Gurobi:

CoverCuts

Cover cut generation

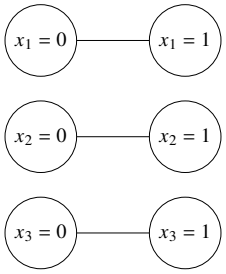
Type:	int
Default value:	-1
Minimum value:	-1
Maximum value:	2

Controls cover cut generation. Use 0 to disable these cuts, 1 for moderate cut generation, or 2 for aggressive cut generation. The default -1 value chooses automatically. Overrides the Cuts parameter.

Clique cuts

Clique cuts identify incompatibilities between variables.

$$x_1 + x_3 \leq 1$$
$$x_1 - x_2 \leq 0$$
$$x_3 - x_2 \leq 0$$



$$x_1 + (1 - x_2) + x_3 \leq 1$$

In Gurobi:

CliqueCuts

Clique cut generation

Type: int

Default value: -1

Minimum value: -1

Maximum value: 2

Controls clique cut generation. Use 0 to disable these cuts, 1 for moderate cut generation, or 2 for aggressive cut generation. The default -1 value choose automatically. Overrides the Cuts parameter.

We have observed that setting this parameter to its aggressive setting can produce a significant benefit for some large set partitioning models.

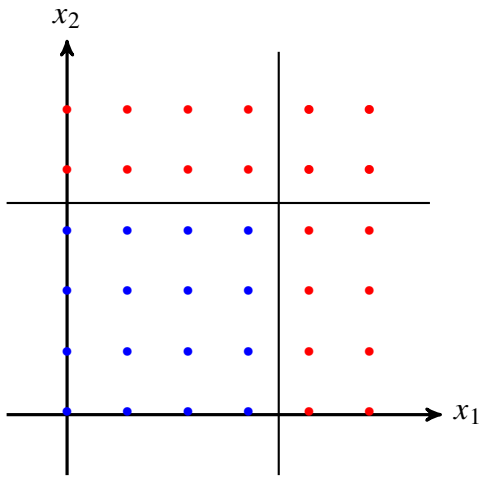
Some other generic cuts

Some other generic cuts implemented in Gurobi:

https://www.gurobi.com/documentation/9.5/refman/parameter_descriptions.html

Cutting planes algorithm

The main idea is to solve the linear programming relaxation and obtain the optimal LP solution. Then generate valid inequalities that cut this solution and iterate.



A general algorithm can be obtained as (also from Wolsey):

1: Initialisation. Set $t = 0$ and $P^0 = P$.
2: Iteration t. Solve the linear program:

$$\bar{z}^t = \max \left\{ cx : x \in P^t \right\}$$

3: Let x^t be an optimal solution.
4: If $x^t \in Z^n$, stop. x^t is an optimal solution for IP
5: If $x^t \notin Z^n$, solve the separation problem for x^t and the family $\mathcal{F}$
6: If an inequality $\left(\pi^t, \pi_0^t\right) \in \mathcal{F}$ is found with $\pi^t x^t > \pi_0^t$ so that it cuts off x^t, set $P^{t+1} = P^t \cap \left\{ x : \pi^t x \leq \pi_0^t \right\}$, and augment t and go to step 2. Otherwise stop.

Chvatal-Gomory cuts

We saw that we could strengthen our formulations by using valid inequalities. The natural question that arises is: can we obtain a simple procedure that will iteratively generate valid inequalities approximating the convex hull (at least around the area of interest)?

The answer to this question was answered by Gomory. The idea is to use the solution of the linear program to generate a valid inequality that will cut the current optimal fractional solution while leaving the convex hull intact.

Remember that in the optimal solution of a Linear Program, we have a set of basic variables (≥ 0) and a set of non-basic variables ($= 0$). The basic variables can be expressed as a linear expression of the non-basic variables as indicated in the general equation of the system:

$$x_B = B^{-1}b - B^{-1}Nx_N$$

Consider x_{B_u} one of the basic variables with a fractional value in this optimal solution. The associated row reads:

$$x_{B_u} = b_u - \sum_{j \in N} a_{uj} x_{N_j} \tag{6.1}$$

or

$$x_{B_u} + \sum_{j \in N} a_{uj} x_{N_j} = b_u \tag{6.2}$$

where b_u is the u^{th} element of $B^{-1}b$ and a_{uj} is the element in row u, column j of $B^{-1}N$. Let:

$$b_u = \lfloor b_u \rfloor + \beta_u$$

and

$$a_{uj} = \lfloor a_{uj} \rfloor + \alpha_{uj}$$

Using the fact that all variables are positive in a feasible solution, we can change the equality (6.2) into an inequality and round down terms a_{uj}:

$$x_{B_u} + \sum_{j \in N} \lfloor a_{uj} \rfloor x_{N_j} \leq b_u \tag{6.3}$$

Note that the left-hand side is integer for any feasible solution, and therefore, we can round down the right hand side and obtain:

$$x_{B_u} + \sum_{j \in N} \lfloor a_{uj} \rfloor x_{N_j} \leq \lfloor b_u \rfloor \tag{6.4}$$

Now, by subtracting (6.2) from (6.4) we obtain:

$$\sum_{j \in N} \alpha_{uj} x_{N_j} \geq \beta_u \tag{6.5}$$

Since all the operations we used were valid for feasible integer solutions, we know that they this cut does not eliminate any of them. Nevertheless, observe that it eliminates the current optimal solution (since all non-basic variables are zero and $\beta_u > 0$).

This method only works when all variables are integer. There is a version for mixed-integer problems (see Wolsey).

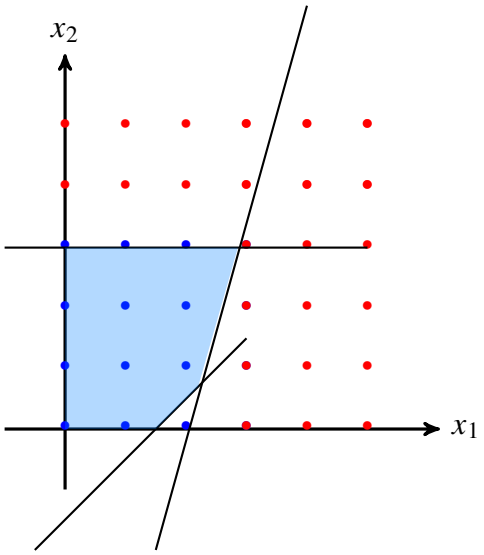
Consider the problem below [Wolsey, 2020]:

z = max $4x_1 - x_2$
s.t.
$7x_1 - 2x_2 \leq 14$
$x_2 \leq 3$
$2x_1 - 2x_2 \leq 3$
$x_1, x_2 \geq 0$ and integers

Solve using cutting planes.

With equality constraints:

$z = \max 4x_1 - x_2$
s.t.
$7x_1 - 2x_2 + x_3 = 14$
$x_2 + x_4 = 3$
$2x_1 - 2x_2 + x_5 = 3$
$x_1, x_2, x_3, x_4, x_5 \geq 0$ and integers



Solving the relaxed version of the problem, we get the following optimal basis:

$$x_B = [x_1, x_2, x_5], x_N = [x_3, x_4]$$

$$B = \begin{bmatrix} 7 & -2 & 0 \\ 0 & 1 & 0 \\ 2 & -2 & 1 \end{bmatrix} \quad N = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

And the general solution of the system is:

$$x_B = B^{-1}b - B^{-1}Nx_N$$

$$\begin{bmatrix} x_1 \\ x_2 \\ x_5 \end{bmatrix} = \begin{bmatrix} 20/7 \\ 3 \\ 23/7 \end{bmatrix} - \begin{bmatrix} 1/7 & 2/7 \\ 0 & 1 \\ -2/7 & 10/7 \end{bmatrix} \begin{bmatrix} x_3 \\ x_4 \end{bmatrix}$$

Both x_1 and x_5 are fractional. Let's use the row of x_1 to generate a Gomory cut:

$$\sum_{j \in N} \alpha_{uj} x_{N_j} \geq \beta_u \rightarrow \frac{1}{7} x_3 + \frac{2}{7} x_4 \geq 6/7.$$

We can rewrite this cut in terms of the original variables by noting that x_3 and x_4 can be written as:

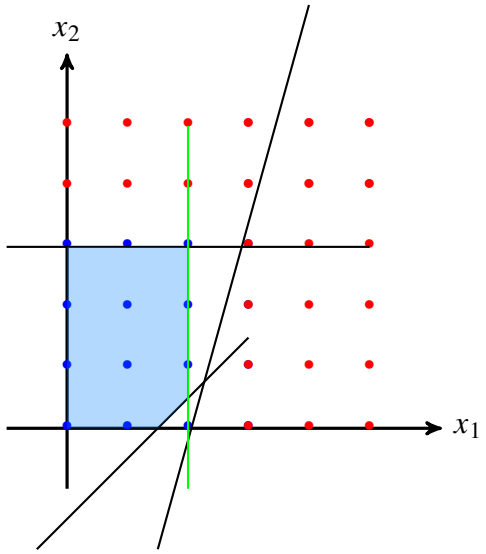
$$x_3 = 14 - 7x_1 + 2x_2,$$
$$x_4 = 3 - x_2.$$

Substituting in the first cut, we get:

$$x_1 \leq 2,$$

yielding the new problem (with the new constraint in green in the picture):

z = max $4x_1 - x_2$
s.t.
$7x_1 - 2x_2 + x_3 = 14$
$x_2 + x_4 = 3$
$2x_1 - 2x_2 + x_5 = 3$
$x_1 + s = 2$
$x_1, x_2, x_3, x_4, x_5, s \geq 0$ and integers



Solving the new linear programming model, we obtain the following optimal basis:

$$x_B = [x_1, x_2, x_3, x_4], x_N = [x_5, s]$$

$$B = \begin{bmatrix} 7 & -2 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 2 & -2 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad N = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

And the general solution of the system is:

$$x_B = B^{-1}b - B^{-1}Nx_N$$

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 2 \\ 1/2 \\ 1 \\ 5/2 \end{bmatrix} - \begin{bmatrix} 0 & 1 \\ 1/2 & -1 \\ 1 & 5 \\ -1/2 & -6 \end{bmatrix} \begin{bmatrix} x_5 \\ s \end{bmatrix}$$

Since basic variables x_2 and x_4 are still fractional in this optimal solution, we can use either of their rows in the system to generate the new Gomory cut. Using row 2 (associated with x_2):

$$\sum_{j \in N} \alpha_{uj} x_{N_j} \geq \beta_u \rightarrow \frac{1}{2} x_5 \geq 1/2.$$

or, in terms of the original variables:

$$x_1 - x_2 \leq 1$$

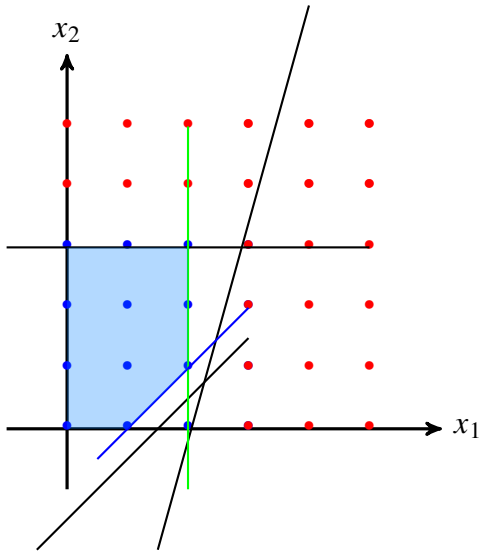
The new program is:

z = max $4x_1 - x_2$
s.t.
$7x_1 - 2x_2 + x_3 = 14$
$x_2 + x_4 = 3$
$2x_1 - 2x_2 + x_5 = 3$

$x_1 + s = 2$

$x_1 - x_2 + t = 1$

$x_1, x_2, x_3, x_4, x_5, s, t \geq 0$ and integers



Which is exactly the convex hull formulation for the original formulation. Solving the linear relaxation will give you the optimal solution $(x_1, x_2) = (2, 1)$.

Note that you do not need to obtain the full convex hull for the problem to be solved. As soon as you get an integer solution for the linear relaxed problem, this solution is optimal for the original integer problem.

In Gurobi:

GomoryPasses

Gomory cut passes

Type:	int
Default value:	-1
Minimum value:	-1
Maximum value:	MAXINT

A non-negative value indicates the maximum number of Gomory cut passes performed. Overrides the Cuts parameter.

Note that the procedure was only valid as all variables (including new slack variables) remained integer.

There are extensions of this cut for mixed-integer problems (see [Wolsey, 2020]).

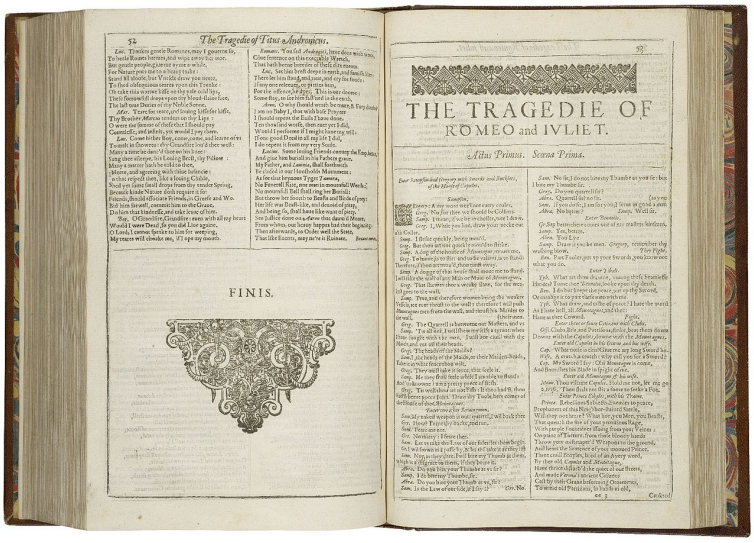
Chapter 7

Branch and Cut

Branch and cut algorithms

At first look, the cutting planes algorithm looks much more effective (and easy to implement) than the branch-and-bound algorithm. You do not need to maintain a list of open problems (you just have a single problem that keeps being enriched at each iteration). However, we often face a tailing-off effect, when the cuts obtained at later iterations tend to cut less and less (fractional) solution space.

The most successful use of the cutting planes algorithm is not as a solo algorithm, but in combination with the branch-and-bound algorithm. Indeed, a valid inequality (such as Chvátal-Gomory cuts) can be used to strengthen your model, improving your dual bounds and fastening the convergence of a branch-and-bound algorithm.



The title page from Romeo and Juliet First Folio, printed in 1623.

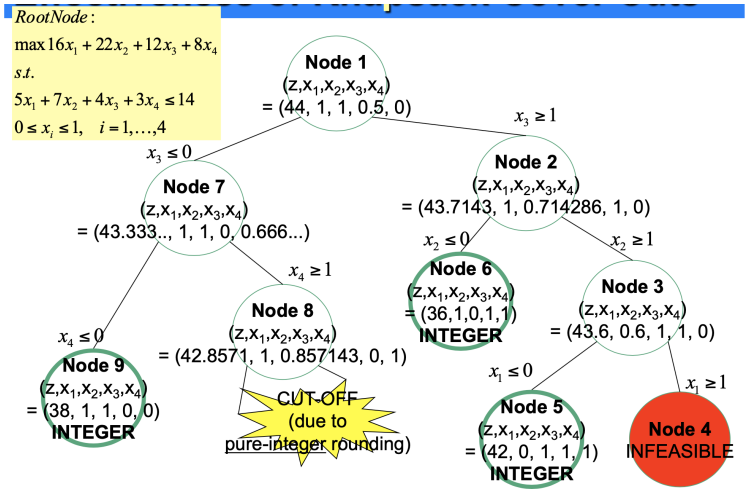
Example: Knapsack cover constraints

Consider again an instance of the knapsack problem:

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

What is the effect of adding cover inequalities?

Solving this problem with the branch-and-bound algorithm, we obtain the following tree:

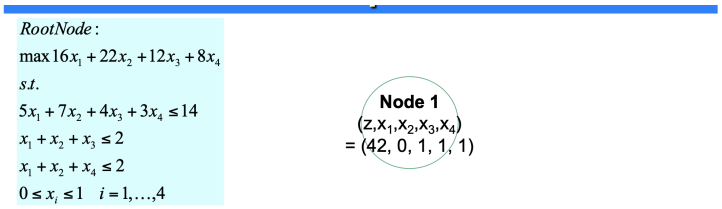


B&B tree for the knapsack example. (figure credit: Heng Soon-Gang)

Now, note that $C = \{1, 2, 3\}$ and $C = \{1, 2, 4\}$ are covers for the knapsack constraint. Therefore, we can have added the following constraints to the original model:

$$x_1 + x_2 + x_3 \leq 2,$$
$$x_1 + x_2 + x_4 \leq 2.$$

By doing so, the branch-and-bound algorithm converges at the root node:



B&B tree for the knapsack example with cover cuts. (figure credit: Heng Soon-Gang)

Beyond the use of cuts to strengthen the linear programming relaxation of models, modern implementations of branch-and-cut algorithms contain also a series of 'add-ons', to improve the performance of the method. We will see the ideas variable selection, node selection, pre-processing and primal heuristics to reduce the size of the trees that are explored.

Computational decisions

Node selection

Which node to explore first?

Depth first:

Explore the leftmost open node of the tree.

It is only possible to prune a node once an integer solution (primal bound) has been found. As these solutions are usually found deep in the tree, this suggests that the depth-first strategy (where we always branch on one side - left or right) should be used.

Another advantage of the depth-first strategy is that the new linear program is exactly the one just solved with the addition of an additional constraint. This allows for a 'hot-start' when solving the new LP.

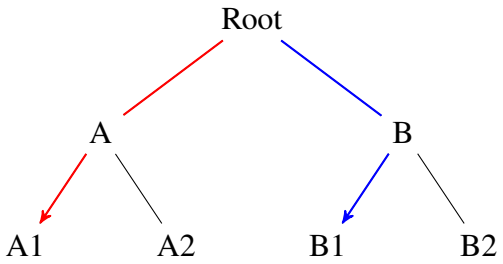
Best bound:

Explore the open node with largest dual bound.

The idea here is to branch on the most promising node. This tends to reduce the size of the explored tree.

Best bound and depth-first rules

Consider the following branch-and-bound tree for a maximisation mixed-integer program.



Propose values for the optimal linear solution value at each node (Root, A, B, A1, A2, B1, B2). These values must be in accordance with the paths generated by a branch and bound approach using both the depth-first approach (red path) and the best bound approach (blue path).

A combination of *depth-first* and *best bound* rules is usually implemented in both commercial (Gurobi, Cplex, Xpress, etc.) and open source (Scip, etc.) solvers. How and why?

Variable selection

When more than one variable has a fractional value at the node, which one should be chosen for branching ?

Random ordering: Ignore influence of ordering: choose any variable that is fractional (randomly, lexicographically,...)

Most fractional: Choose the "most fractional" variable.

For a fractional variable i at a given node, let:

- Down-fractionality: $f_i = x_i^* - \lfloor x_i^* \rfloor$
- Up-fractionality: $1 - f_i$.

Choose variable x_i with $\max(\min(f_i, 1 - f_i))$.

Computational tests show that this strategy is not better than a random ordering choice [Achterberg et al., 2005].

Pseudo-cost branching: Choose variable with best record. If no record is available, use a random rule. The idea here is that, since you can branch multiple times on a single variable, we can use the information on the effect that these earlier branchings had on the objective function to create a 'score'.

Let P_i^-, P_i^+, called pseudo-costs, be historical estimates of the cost per unit of forcing a variable x_i down, respectively up, to become integral.

Then the estimated cost of setting $x_i \rightarrow \lfloor x_i^* \rfloor$ is the down degradation $D_i^- = f_i P_i^-$ and the cost of setting $x_i \rightarrow \lceil x_i^* \rceil$ is $D_i^+ = (1 - f_i) P_i^+$.

Choose a branching variable using either: $\hat{i} = \text{argmax}_i \left[\left(D_i^- + D_i^+ \right) \right]$, or $\hat{i} = \text{argmax}_i \left[\min \left(D_i^-, D_i^+ \right) \right]$.

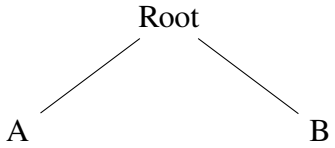
Strong branching Test the effect of branching on each possible fractional variables on the bounds by actually solving the resulting linear programs and branch on the variable with *the most effect*.

Costly but effective: this highlights the importance of reducing the size of the tree and the importance of node selection.

Reliability branching Use strong branching to generate data and then switch to pseudo-cost branching.

Branching rules

Consider the following branch-and-bound tree for an integer program on variables x_1 and x_2.



At the optimal solution of the root node, $x_1 = 0.2$ and $x_2 = 1.7$ ($z = 17.1$). The bound z for nodes A and B depend on the branching constraints applied to the root node and are listed below:

$z = 18.1$ for node A with branching constraint $x_1 \leq 0$,

$z = 20.1$ for node A with branching constraint $x_2 \leq 1$,

$z = 22.1$ for node B with branching constraint $x_1 \geq 1$,

$z = 24.6$ for node B with branching constraint $x_2 \geq 2$.

a) Using a *pseudo-cost branching* rule, which variable should we select for branching at the root node?

b) Using a *strong branching* rule, which variable should we select for branching at the root node?

Pre-processing

Preprocessing is the use of logical rules to determine the infeasibility of a model or to reduce its size, eliminating unnecessary variables and constraints.

Consider the constraint $a_0x_0 + \sum_{j=1}^{n} a_j x_j \leq b$ on variables x_j, $l_j \leq x_j \leq k_j$ for $j \in \{1, \ldots, n\}$.

(i) Bounds on variable x_0.

$$x_0 \leq \left(b - \sum_{j \geq 1 : a_j > 0} a_j l_j - \sum_{j \geq 1 : a_j < 0} a_j k_j \right) / a_0, \qquad a_0 > 0,$$

$$x_0 \geq \left(b - \sum_{j \geq 1 : a_j > 0} a_j l_j - \sum_{j \geq 1 : a_j < 0} a_j k_j \right) / a_0, \qquad a_0 < 0.$$

(ii) Redundancy.

The constraint $a_0x_0 + \sum_{j=1}^{n} a_j x_j \leq b$ is redundant if

$$\sum_{j : a_j > 0} a_j k_j + \sum_{j : a_j < 0} a_j l_j \leq b.$$

(iii) Infeasibility.

The constraint $a_0x_0 + \sum_{j=1}^{n} a_j x_j \leq b$ is infeasible if

$$\sum_{j : a_j > 0} a_j l_j + \sum_{j : a_j < 0} a_j k_j > b.$$

Algorithmic preprocessing

Consider:

Generic MIP constraints:
$$Ax + Gy \leq b \tag{7.1}$$
$$x \in \{0,1\}^n, y \in \mathbb{R}^m \tag{7.2}$$

Let $a^l x + g^l y \leq b_l$ be one of the inequalities of the system and $A'x + G'y \leq b'$ be the system obtained by deleting this inequality from the original system.

We can use this notation to devise several optimisation problems aimed at finding redundancies or infeasibilities in constraints and variables that can be fixed.

Consider:

z = minimise/maximise $a^l x + g^l y$ (7.3)

s.t.

$A'x + G'y \leq b'$ (7.4)

$x \in \{0,1\}^n, y \in \mathbb{R}^m$ (7.5)

a) What can you deduct if for a minimisation problem you obtain $z^* > b_l$?

b) What can you deduct if for a maximisation problem you obtain $z^* < b_l$?

Note on the example: These auxiliary problems are not usually solved to optimality. Instead, heuristics are used to obtain bounds and conclude the results. Refer to [Savelsbergh, 1994] for more examples.

Primal heuristics

We can only prune the tree if we have at least one feasible solution. This is why we often start the exploration with a depth first search.

The better the quality of the bound, the more we can prune. So we have the interest in exploring the partial solution in order to 'complete' an integer solution.

Primal heuristics can play an important role in the user experience and when the solver is used as a heuristic method - i.e., when the user is more interested in finding good solutions quickly than on proving the optimality of the obtained solutions.

Quick and dirty

Solvers implement several of such heuristics. These heuristics are usually simple and fast.

Heuristics for MIP explained in one tweet

Feasibility Pump

The Feasibility Pump heuristic is a local search technique used to find high-quality feasible solutions for mixed-integer programming (MIP) problems. The main idea behind the heuristic is to check if there are high quality integer solutions *around* the relaxed solution provided at a given node of the tree.

1. Solve the LP relaxation of the MIP problem to obtain an initial fractional solution, x^*.
2. Round the fractional solution to obtain an integer solution, $\tilde{x}$.
3. Check if the rounded solution $\tilde{x}$ is feasible for the original MIP problem. If it is, return $\tilde{x}$ and terminate the algorithm.
4. If the rounded solution is not feasible, create a new continuous problem by minimizing the distance between the current fractional solution and $\tilde{x}$, subject to the original constraints.
5. Go back to step 1 and repeat the process until a feasible solution is found or the maximum number of iterations is reached.

An (failed) example:

Consider the following MIP problem:

$$\begin{aligned} \text{Maximise:} \quad & z = x_1 + 2x_2 \\ \text{Subject to:} \quad & 2x_1 + x_2 \geq 4, \\ & x_1 - x_2 \leq 2.5, \\ & x_1, x_2 \in \mathbb{Z}_+. \end{aligned}$$

Suppose we have an solution at a given node of the tree be $(x_1^*, x_2^*) = (3.5, 1)$ with value $z = 5.5$ and apply the FP heuristic:

1. $(x_1^*, x_2^*) = (3.5, 1)$ with value $z = 5.5$.
2. We round the solution to $(\tilde{x}_1 = 4, \tilde{x}_2 = 1)$ which is infeasible.
3. We then solve the problem.

$$\begin{aligned} \text{Minimize:} \quad & z = |x_1 - 4| + |x_2 - 1| \\ \text{Subject to:} \quad & 2x_1 + x_2 \geq 4, \\ & x_1 - x_2 \leq 2.5, \\ & x_1, x_2 \in \mathbb{R}_+. \end{aligned}$$

4. The reduced MIP problem yields the same linear as before solution $(x_1, x_2) = (3.5, 1)$, that is, feasibility pump was not able to find any integer solution. Can you propose strategies that can be used to try again?

   Also, note that the problem is not linear because of the absolute values in the objective function. How can you linearise it?

Example:

Consider the following MIP problem:

$$\begin{aligned} \text{Maximise:} \quad & z = x_1 + 10 * x_2 \\ \text{Subject to:} \quad & 0.25x_1 + x_2 \leq 4, \\ & x_1 \leq 4, \\ & x_2 \leq 3.2, \\ & x_1, x_2 \in \mathbb{Z}_+. \end{aligned}$$

The solution of the linear relaxation is

```
import gurobipy as gp
from gurobipy import GRB

m = gp.Model("FP")
x1 = m.addVar(name = 'x1')
x2 = m.addVar(name = 'x2')

m.addConstr(0.25*x1 + x2 <= 4  )
m.addConstr(x1 <= 4  )
m.addConstr(x2 <= 3.2  )

m.setObjective(x1 + 10*x2, GRB.MAXIMIZE  )

m.optimize()

print(x1.x, " ", x2.x)
```

$(x_1^*, x_2^*) = (3.199, 2.2)$ with value $z = 35.2$. Apply the FP heuristic:

1. $(x_1^*, x_2^*) = (3.199, 2.2)$ with value $z = 3.52$.
2. We round the solution to $(\tilde{x}_1 = 4, \tilde{x}_2 = 4)$ which is infeasible.
3. We then solve the problem.

$$
\begin{aligned}
\text{Minimize:} \quad & z = z_1 + z_2 \\
\text{Subject to:} \quad & 0.25x_1 + x_2 \leq 4, \\
& x_1 \leq 4, \\
& x_2 \leq 3.2, \\
& z_1 \geq x_1 - 4, \\
& z_1 \geq 4 - x, \\
& z_2 \geq x_2 - 4, \\
& z_2 \geq 4 - x_2, \\
& x_1, x_2 \in \mathbb{R}_+.
\end{aligned}
$$

```
import gurobipy as gp
from gurobipy import GRB

m = gp.Model("FP1")
x1 = m.addVar(name = 'x1')
x2 = m.addVar(name = 'x2')
z1 = m.addVar(name = 'z1')
z2 = m.addVar(name = 'z2')

m.addConstr(0.25*x1 + x2 <= 4  )
m.addConstr(x1 <= 4  )
m.addConstr(x2 <= 3.2  )
m.addConstr(z1 >= x1 - 4  )
m.addConstr(z1 >= 4 - x1 )
m.addConstr(z2 >= x2 - 4  )
m.addConstr(z2 >= 4 - x2 )

m.setObjective(z1 + z2, GRB.MINIMIZE  )

m.optimize()

print(x1.x, " ", x2.x)
```

4. The new problem yields the integer solution $(x_1, x_2) = (4, 3)$ with value $z = 34$, that is, feasibility pump was able to find any integer solution.

Relaxation Induced Neighborhood Search (RINS)

The RINS heuristic is a local search technique used to find high-quality feasible solutions for mixed-integer programming (MIP) problems. The main idea behind the RINS heuristic is to identify a small subproblem by fixing some variables, then solving the reduced problem to optimality. This process is repeated multiple times, each time exploring a different neighborhood of the current best solution.

1. Assume you have already found a feasible initial integer solution, $\tilde{x}$.
2. Select a relaxation solution from a node of the branch and bound tree, x^*.
3. Identify a subset of integer variables that have the same values in the integer solution and the relaxation solution.
4. Fix these integer variables at their common values, and solve the reduced MIP problem.
5. If the solution of the reduced MIP problem improves the current best solution, update the best solution.
6. Repeat the process using different relaxation solutions from the branch-and-bound tree.

Example:

Consider the following MIP problem:

$$\begin{aligned} \text{Maximise:} \quad & z = x_1 + 2x_2 \\ \text{Subject to:} \quad & 2x_1 + x_2 \geq 4, \\ & x_1 - x_2 \leq 2.5, \\ & x_1, x_2 \in \mathbb{Z}_+. \end{aligned}$$

Assume you have an initial integer solution $(\tilde{x}_1, \tilde{x}_2) = (2, 1)$ with optimal value $z = 4$ and a solution at a node $(x_1^*, x_2^*) = (3.5, 1)$. Apply the RINS heuristic:

1. $(\tilde{x}_1, \tilde{x}_2) = (2, 1)$ with optimal value $z = 4$.
2. $(x_1^*, x_2^*) = (3.5, 1)$ with value $z = 5.5$.
3. Identify the subset of integer variables with the same values in both the linear relaxation solution at the node and the current integer solution ($x_2 = 1$ in both solutions).
4. Solve the reduced MIP problem with the constraint $x_2 = 1$:

$$\begin{aligned} \text{Maximise:} \quad & z = x_1 + 2x_2 \\ \text{Subject to:} \quad & 2x_1 + x_2 \geq 4, \\ & x_1 - x_2 \leq 2.5, \\ & x_2 = 1, \\ & x_1, x_2 \in \mathbb{Z}_+. \end{aligned}$$

5. The reduced MIP problem yields a solution $(x_1, x_2) = (3, 1)$ with $z = 5$. Since the new solution has a better objective value than the best current one, we update the best solution: it is now $(x_1, x_2) = (3, 1)$ with $z = 5$.

Developing Customized Branch-&-Cut algorithms

Modern solvers allow us to include problem-specific routines into the branch-&-cut algorithm. The way we interact with the solver is via 'Callbacks'.

Callbacks

The Gurobi callback class provides a set of constants that are used within the user callback function. The first set of constants in this class list the options for the where argument to the user callback function. The where argument indicates from where in the optimization process the user callback is being called.

where	Numeric value	Optimizer status
POLLING	0	Periodic polling callback
PRESOLVE	1	Currently performing presolve
SIMPLEX	2	Currently in simplex
MIP	3	Currently in MIP
MIPSOL	4	Found a new MIP incumbent
MIPNODE	5	Currently exploring a MIP node
MESSAGE	6	Printing a log message
BARRIER	7	Currently in barrier
MULTIOBJ	8	Currently in multi-objective optimization

Callbacks available in Gurobi

Example: a strategy is to ignore constraints and add them when violated. These constraints are then called *Lazy constraints*. Lazy constraints are often used for constraints that can not be generated explicitly due to their high number (SECS, e.g.). In this case, we want to get access to the solver every time an integer solution is found, so we can check if any lazy constraint is violated.

Lazy constraints

Consider again the Dovetail problem.

$z = \max 3x_1 + 2x_2$

s.t.

$x_1 + x_2 \leq 9,$

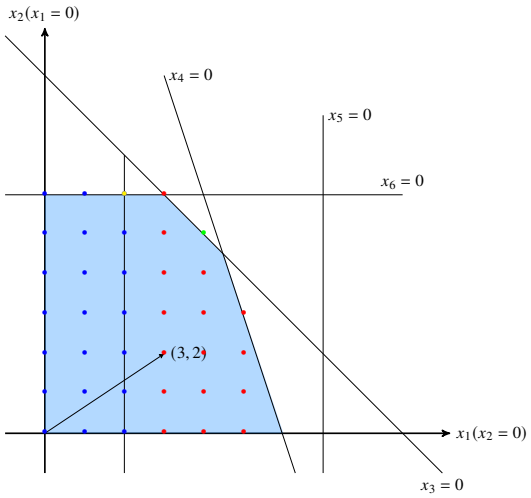
$3x_1 + x_2 \leq 18,$

$x_1 \leq 7,$

$x_2 \leq 6,$

$x_1, x_2 \geq 0.$

Assume now that variables must be integer and there is a lazy constraint $x_2 \leq 2$.



Implement the model.

```
#!/usr/bin/env python3.7

# Example: Imposing lazy constraint on DOVETAIL MODEL

import gurobipy as gp
from gurobipy import GRB


#PART 1 - CALLBACK
def callbackRoutine(model, where):
    if where == GRB.Callback.MIPSOL:
        print("A")
        #Get problem variables
        vals = model.cbGetSolution(model._x)

        #Run some code and add constraint:
        if vals[0]>= 2:
            print("Added constraint: x[1] <= 2")
            model.cbLazy(model._x[0] <= 2.1)


#PART 2 - MODEL

m = gp.Model("dovetail")
c = [3,2]
A = [[1, 1], [3,1], [1,0], [0,1]]
b = [9,18,7,6]
N = range(len(c))
M = range(len(b))

x= m.addVars(N, name='x',vtype=GRB.INTEGER)
m.setObjective( gp.quicksum(x[i]*c[i] for i in N), GRB.MAXIMIZE)
cons= m.addConstrs( gp.quicksum(A[j][i]*x[i] for i in N) <= b[j] for j in M )

#Optimise with lazy constraints
m._x = x
m.Params.LazyConstraints = 1
m.optimize(callbackRoutine)

#Optimise without lazy constraints
# m.optimize()


# Display optimal values of decision variables
for v in m.getVars():
    if v.x > 1e-6:
        print(v.varName, v.x)

# Display optimal solution value
print('Total profit: ', m.objVal)
```

Separation problems

In the tutorials, we see how to find violated subtour elimination constraints for the TSP by simply examining the solution in search for subtours. In that case, we run the branch-and-bound method until an optimal solution was found for the problem and only then looked for the violations.

In general, the process of finding new violated inequalities does not need to wait for an integer solution. Fractional solutions can be used to generate valid inequalities (on the root node or in any other node of the B&B tree). We call the proces of finding these inequalities a *separation problem*.

Separating cover inequalities

Write a model to discover violates cover inequalities.

Consider the set $X = \left\{x \in \{0,1\}^n : \sum_{j=1}^n a_j x_j \leq b\right\}$ [Wolsey, 2020]. Complementing variables if necessary by setting $\bar{x}_j = 1 - x_j$, we assume throughout this section that the coefficients $\{a_j\}_{j=1}^n$ are positive. Also we assume $b > 0$. Let $N = \{1, \ldots, n\}$.

Let $\mathbb{F}$ be the family of cover inequalities for X and let us examine the separation problem for this family. Explicitly we are given a nonintegral point x^* with $0 \leq x_j^* \leq 1$ for all $j \in N$ and we wish to know whether x^* satisfies all the cover inequalities. To formalize this problem, note that the cover inequality can be rewritten as

$$\sum_{j \in C} (1 - x_j) \geq 1$$

Thus, it suffices to answer the question: Does there exist a set $C \subseteq N$ with $\sum_{j \in C} a_j > b$ for which $\sum_{j \in C} \left(1 - x_j^*\right) < 1$?, or put slightly differently: Is $\zeta = \min_{C \subseteq N} \left\{\sum_{j \in C} \left(1 - x_j^*\right) : \sum_{j \in C} a_j > b\right\} < 1$?

As the set C is unknown, this can be formulated as a $0-1$ integer program where the variable $z_j = 1$ if $j \in C$ and $z_j = 0$ otherwise. So the question can be restated yet again:

$$\text{Is } \zeta = \min \left\{\sum_{j \in N} \left(1 - x_j^*\right) z_j : \sum_{j \in N} a_j z_j > b, z \in \{0,1\}^n\right\} < 1?$$

NOTE: Usually we don't want to solve another model in order to obtain cuts. We rely on simpler algorithms that are faster.

Efficient separation problem for SECs

Consider the following formulation for the symmetric TSP on variables $x_{ij} \in \{0, 1\}$:

$$\text{Min } c_{ij}x_{ij} \tag{7.6}$$

subject to:

$$\sum_{j \in N, j \neq i} x_{ij} = 1, \qquad \forall i \in N, \tag{7.7}$$

$$\sum_{j \in N, j \neq i} x_{ji} = 1, \qquad \forall i \in N, \tag{7.8}$$

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1, \qquad \forall S \subseteq N - \{1\}, |S| \geq 2, \tag{7.9}$$

$$x_{ij} \in \{0, 1\}. \tag{7.10}$$

Feasible and infeasible (subtours) solutions:

We can use a max flow / min cut algorithm to find violated inequalities.

Chapter 8

Column generation and Branch-and-Price

Decomposition methods

Mixed-Integer Programs are hard to solve. Even with the impressive evolution on algorithms and hardware of the past few decades, MIPs are NP-hard problems and, as the size of the problems grows, eventually there is a moment when even the best solvers running on the most powerful machines will run out of memory or time.

In the following chapters, we will extend the applicability of MIP by introducing decomposition methods, which divide the problem to be solved in complementary parts that work iteratively, substituting the solution of a huge MIP for the solution of multiple smaller MIPs or LPs. We start with the column generation method.

Symmetry

Simmetry in MIP happens when a feasible solution of a problem is represented by multiple solutions of the model.

Example:

Consider the unidimensional cutting stock problem presented in the modelling chapter. A compact formulation for the problem was introduced as:

1-dim cutting: $\text{z} = \min \sum_{j \in J} y_j$

s.t.

$$\sum_{j \in J} x_{ij} \geq d_i, \quad \forall i \in I$$

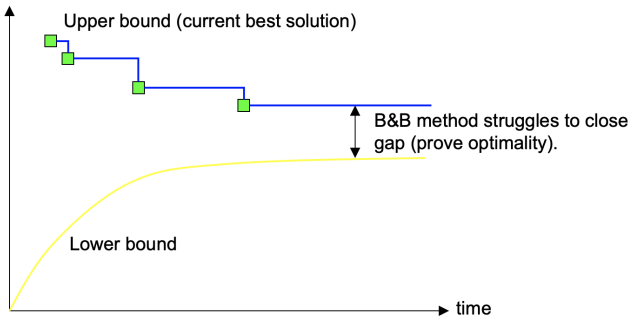
$$\sum_{i \in I} l_i x_{ij} \leq L y_j, \quad \forall j \in J$$

$$x_{ij} \in \mathbb{Z}^+, \quad \forall i \in I, j \in J$$

$$y_i \in \{0, 1\} \quad \forall j \in J$$

One problem of this formulation is that it contains multiple representations for the same solution. This is due to the fact that we differentiate between the pieces of raw material $j \in J$.

This leads to the following behaviour on the branch and bound algorithm.



Eliminating symmetry

A strategy for eliminating symmetry is to use *extensive formulations* over an *exponentially large* set of variables.

Example: Patterns for a bin-packing problem

Consider the 1-d bin packing problem.

Data:

- Paper rolls are of 17m lengths
- Customer demands:

  25 pieces of length 3m,

  20 pieces of length 5m,

  15 pieces of 9m.

Propose cutting patterns.

We define cutting patterns as possible configurations for bins.

Items ↓ / Patterns →	1	2	3	4	5	6	7
1	5	4	2	2	1	0	0
2	0	1	2	0	1	3	0
3	0	0	0	1	1	0	1

An extensive formulation

Let:

- P : set of possible cutting patterns,
- I : set of items to be cut,
  d_i : the demand for item $i \in I$,
- a_{ip} : units of item $i \in I$ cut in pattern $p \in P$,

and define variables x_p indicating the number of patterns of type $p \in P$ to cut.

An extensive formulation can be written as:

$$z = \min \sum_{p \in P} c_p x_p$$

s.t.

$$\sum_{p \in P} a_{ip} x_p \geq d_i, \quad i \in I,$$

$$x_p \in \mathbb{Z}_+$$

The extensive formulation can only be written explicity for very small problems as the number of patterns grows exponentially with the number of items.

Relaxing the master problem

To cope withg the large number of variables, we can start with a relaxed version of the problem which considers only a few patterns.

A relaxed master problem

Consider the data in the previous example and initialise your problem with patterns of type 1, 6 and 7. The problem reads:

$$z = \min \sum_{p \in P} x_1 + x_6 + x_7$$

s.t.

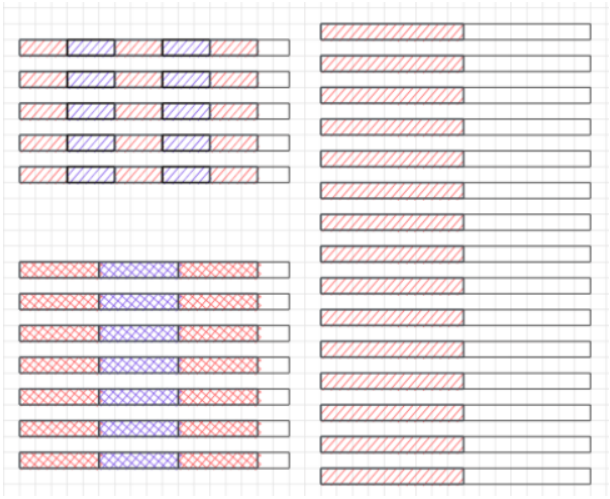
$$5x_1 \geq 25$$

$$3x_2 \geq 20$$

$$x_3 \geq 15$$

$$x_1, x_2, x_3 \in \mathbb{R}_+$$

The solution of the problem in the example provides the following (inefficient) solution:



Example of solution for the bin-packing problem in the example

Pricing new columns

Consider the linear-relaxation version of the master problem.

$$
\begin{aligned}
z = \min & \sum_{p \in P} x_1 + x_6 + x_7 \\
& \text{s.t.} \\
& 5x_1 \geq 25 \\
& 3x_2 \geq 20 \\
& x_3 \geq 15 \\
& x_1, x_2, x_3 \in \mathbb{R}_+
\end{aligned}
$$

Given a primal solution, the duals associated with the functional constraints (say, u_1, u_2 and u_3) indicate the 'local' cost of providing one unit of item 1, 2 and 3, respectively.

Now imagine a new pattern producing a_1, a_2 and a_3 units of items 1, 2 and 3, respectively. Introducing one unit of this pattern in the solution will have the following local effect on the cost:

$$\text{Price of new column} = 1 - u_1 a_1 - u_2 a_2 - u_3 a_3$$

A pricing problem can then be written as:

$$
\begin{aligned}
z = \min & \; 1 - u_1 a_1 - u_2 a_2 - u_3 a_3 \\
& \text{s.t.} \\
& l_1 a_1 + l_2 a_2 + l_3 a_3 \leq L \\
& a_1, a_2, a_3 \in \mathbb{Z}_+
\end{aligned}
$$

The column generation algorithm

Algorithm 3 Column generation main idea

1: Find a feasible set of patterns.
2: Solve the relaxed master problem and obtain the dual values
3: Use the dual values to price the best new column. If no column with negative cost can be found, STOP (the solution is optimal for the LP problem). Otherwise, add the column to the master problem and return to 2.

The column generation algorithm solves Linear Programs. A simple heuristic is to solve the linear relaxation problem with column generation and then use the columns generated to find an integer solution. This heuristic does not necessarily provide feasible solutions.

Solving the bin-packing problem using CG

Consider again the 1-d bin packing problem with paper rolls are of 17m lengths and customer demands are 25 pieces of length 3m, 20 pieces of length 5m and 15 pieces of 9m.

Solve the linear relaxation of the problem using column generation.

Pattern-based Formulation

Step 0:

Choose subset of columns corresponding to cutting patterns $(5\ 0\ 0)^T$, $(0\ 3\ 0)^T$, $(0\ 0\ 1)^T$, i.e.

$$A^s = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$P = \{(5,0,0),(0,3,0),(0,0,1)\}$$

	# of 3m	# of 5m	# of 9m
Pattern 1	5	0	0
Pattern 2	4	1	0
Pattern 3	2	2	0
Pattern 4	2	0	1
Pattern 5	1	1	1
Pattern 6	0	3	0
Pattern 7	0	0	1

1

Pattern-based Formulation

Step 1:

Solve

$$\text{Restricted LP}: \begin{cases} \min & y_1 + y_2 + y_3 \\ s.t. & \\ 5y_1 \geq 25 \\ 3y_2 \geq 20 \\ y_3 \geq 15 \\ y \geq 0 \end{cases}$$

$P = \{(5,0,0),(0,3,0),(0,0,1)\}$

$$\begin{cases} \min & \sum_{p \in P} x_p \\ s.t. & \\ \sum_{p \in P} a_{ip} x_p \geq b_i, & i = 1,2,3 \\ x \geq 0 \end{cases}$$

Optimal basic solution is

$y_1 = 5,\ y_2 = 6\ 2/3,\ y_3 = 15$

and optimal dual variables

$u = (1/5,\ 1/3,\ 1)$

2

Pattern-based Formulation

Step 2:

Solve column generation subproblem:

$$\begin{cases} \min & 1 - (1/5, 1/3, 1) a_p \\ s.t. & \\ a_p \text{ is a column of the Master LP} \end{cases}$$

But what does it mean for a_p to be a column of the Master LP?

$a_p = (a_{1p}, a_{2p}, a_{3p})^T$ is integer, ≥ 0 and satisfies

$3a_{1p} + 5a_{2p} + 9a_{3p} \leq 17$

3

Pattern-based Formulation

Therefore column generation subproblem is

$$z^{sub} = \begin{cases} \min & 1 - \left(\frac{1}{5}a_{1p} + \frac{1}{3}a_{2p} + a_{3p}\right) \\ s.t. & \\ 3a_{1p} + 5a_{2p} + 9a_{3p} \leq 17 \\ a_{1p}, a_{2p}, a_{3p} \geq 0, \text{integer} \end{cases}$$

Déjà vu again?

$$= 1 - \begin{cases} \max & \frac{1}{5}a_{1p} + \frac{1}{3}a_{2p} + a_{3p} \\ s.t. & \\ 3a_{1p} + 5a_{2p} + 9a_{3p} \leq 17 \\ a_{1p}, a_{2p}, a_{3p} \geq 0, \text{integer} \end{cases}$$

Optimal solution:
$a_{1p} = a_{2p} = a_{3p} = 1$

z^{sub}
$= 1 - (1/5 + 1/3 + 1)$
$= -8/15 < 0$

4

Pattern-based Formulation

Minimum reduced cost is negative, so column $(1\ 1\ 1)^T$ should be added to A^s.

We return to Step 1 and solve the new restricted LP:

$$\text{Restricted LP}: \begin{cases} \min & y_1 + y_2 + y_3 + y_4 \\ s.t. & \\ 5y_1 + y_4 \geq 25 \\ 3y_2 + y_4 \geq 20 \\ y_3 + y_4 \geq 15 \\ y \geq 0 \end{cases}$$

$P = \{(5,0,0),(0,3,0),(0,0,1),(1,1,1)\}$

$$\begin{cases} \min & \sum_{p \in P} x_p \\ s.t. & \\ \sum_{p \in P} a_{ip} x_p \geq b_i, & i = 1,2,3 \\ x \geq 0 \end{cases}$$

Optimal basic solution is

$y_1 = 2,\ y_2 = 1\ 2/3,\ y_3 = 0,\ y_4 = 15$

and optimal dual variables

$u = (1/5,\ 1/3,\ 7/15)$

5

Pattern-based Formulation

Column generation subproblem is

$$z^{sub} = 1 - \begin{cases} \max & \frac{1}{5}a_{1p} + \frac{1}{3}a_{2p} + \frac{7}{15}a_{3p} \\ s.t. & \\ 3a_{1p} + 5a_{2p} + 9a_{3p} \leq 17 \\ a_{1p}, a_{2p}, a_{3p} \geq 0, \text{integer} \end{cases}$$

Optimal solution to subproblem is

$a_1 = a_2 = a_3 = 1$

Therefore $z^{sub} = 1 - (1/5+1/3+7/15) = 0$

6

Pattern-based Formulation

Thus there can be no negative reduced cost columns and optimal solution to Master LP is to use

2 x $(5\ 0\ 0)^T$, **1 2/3** x $(0\ 3\ 0)^T$, **15** x $(1\ 1\ 1)^T$

Note:

This is not integer!!

But LP value = 2 + 1 2/3 + 15 = 18 2/3 is lower bound on problem.

Observation: Now

2 x $(5\ 0\ 0)^T$, **2** x $(0\ 3\ 0)^T$, **15** x $(1\ 1\ 1)^T$

is an integer feasible solution with value 19.

Is this the optimal solution?

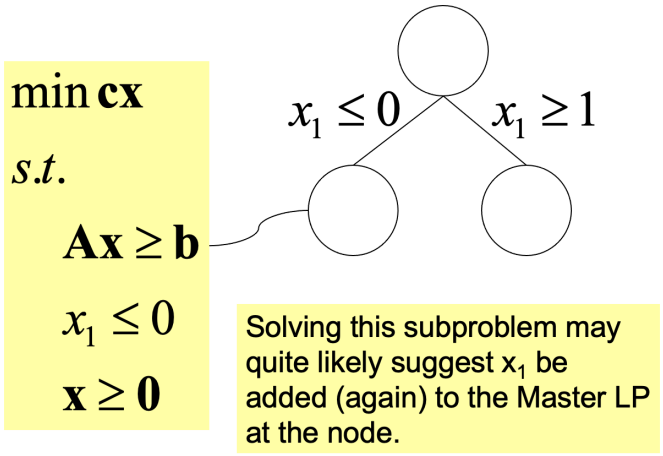
7

A brief introduction to Branch-and-price

If the problem contains integer variables, branching on fractional values is needed for the obtention of provable optimal solutions. This technique is called Branch-and-Price.

Branch-and-price extends column-generation by branching on fractional variables, such as in the Branch-and-bound method. Note, however, that now we need to solve a column generation approach at each node of the tree.

Branching constraints might affect the structure of the problem to be solved at each node. The main design aspect of a branch-and-price algorithm is often the obtention of good branching rules.



The problem with classical branching rules when applied in a branch-and-price algorithm.

In order to avoid generating the same column for the classical branching strategy, we would need a pricing problem such as:

$$
\begin{aligned}
& z = \min 1 - \sum_{i=1}^{N} u_i a_i \\
& \text{s.t.} \\
& \sum_{i=1}^{N} l_i a_i \leq L \\
& a_i \in \mathbb{Z}_+, \quad i = 1, \ldots N, \\
& (a_1, a_2, \ldots, a_N) \neq (a_{11}, a_{21}, \ldots, a_{N1})
\end{aligned}
$$

The last constraint destroys the subproblem structure. It requires the subproblem to find the 2nd best knapsack solution.

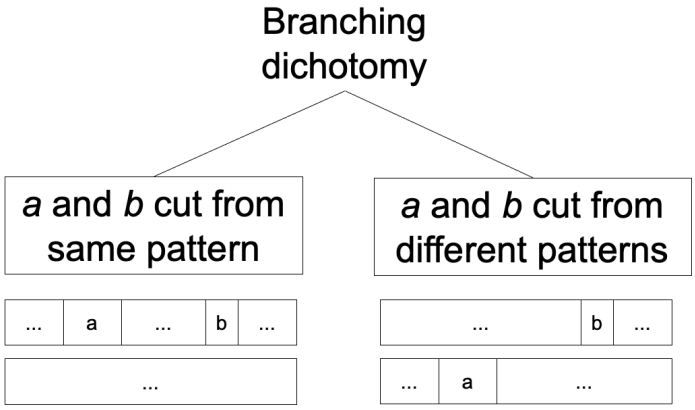
A better branching rule for the binary cutting problem

Binary cutting problem: all demands are equal to 1 and must be supplied at equality.

Let:

- P : set of all cutting patterns.
- P_i : set of cutting patterns that has piece with width i.

Consider also two fractional variables, $0 \leq x_1, x_2 \leq 1$ associated with two patterns $P_1, P_2 \in P_a$. And an item b such that $P_1 \in P_b$, $P_2 \notin P_b$. A branching rule can be described as:



A branching rule for the binary cutting stock problem.

Pricing problems

The pricing problems are not as complicated as before:

Left branch:

$$
\begin{aligned}
& z = \min 1 - \sum_{i=1}^{N} u_i a_i \\
& \text{s.t.} \\
& \sum_{i=1}^{N} l_i a_i \leq L \\
& a_i \in \mathbb{Z}_+, \quad i = 1, \ldots N, \\
& a_a = a_b
\end{aligned}
$$

Right branch:

$$
\begin{aligned}
& z = \min 1 - \sum_{i=1}^{N} u_i a_i \\
& \text{s.t.} \\
& \sum_{i=1}^{N} l_i a_i \leq L \\
& a_i \in \mathbb{Z}_+, \quad i = 1, \ldots N, \\
& a_a = 1 - a_b
\end{aligned}
$$

Chapter 9

Benders decomposition

Mixed-Integer Programs are hard to solve. Even with the impressive evolution on algorithms and hardware of the past few decades, MIPs are NP-hard problems and, as the size of the problems grows, eventually there is a moment when even the best solvers running on the most powerful machines will run out of memory or time.

In this chapter we will extend the applicability of MIP by introducing decomposition methods, which divide the problem to be solved in complementary parts that work iteratively, substituting the solution of a huge MIP for the solution of multiple smaller MIPs or LPs.

Benders decomposition

Benders decomposition was first proposed by Benders in 1962 [Benders, 1962]. For many years it was a nice theoretical result with little applicability in practice. In 1974, Geoffrion and Graves [Geoffrion and Graves, 1974] used the technique to plan the distribution of items in a supply chain with intermediate depots between production and consumption sites, fostering the application and the theoretical study of the method. Nowadays, Benders decomposition is one of the most successful decomposition techniques in mixed-integer programming.

Numerische Mathematik 4, 238—252 (1962)

Partitioning procedures for solving mixed-variables programming problems*

By

J. F. BENDERS**

I. Introduction

In this paper two slightly different procedures are presented for solving mixed-variables programming problems of the type

$$\max\{c^T x + f(y) \mid A x + F(y) \leqq b,\ x \in R_p,\ y \in S\}, \tag{1.1}$$

where $x \in R_p$ (the p-dimensional Euclidean space), $y \in R_q$, and S is an arbitrary subset of R_q. Furthermore, A is an (m, p) matrix, $f(y)$ is a scalar function and $F(y)$ an m-component vector function both defined on S, and b and c are fixed vectors in R_m and R_p, respectively.

An example is the mixed-integer programming problem in which certain variables may assume any value on a given interval, whereas others are restricted to integral values only. In this case S is a set of vectors in R_q with integral-valued components. Various methods for solving this problem have been proposed by BEALE [1], GOMORY [9] and LAND and DOIG [11]. The use of integer variables, in particular for incorporating in the programming problem a choice from a set of alternative discrete decisions, has been discussed by DANTZIG [4].

MANAGEMENT SCIENCE
Vol. 20, No. 5, January, 1974
Printed in U.S.A.

MULTICOMMODITY DISTRIBUTION SYSTEM DESIGN BY BENDERS DECOMPOSITION*†

A. M. GEOFFRION AND G. W. GRAVES§

University of California, Los Angeles

A commonly occurring problem in distribution system design is the optimal location of intermediate distribution facilities between plants and customers. A multicommodity capacitated single-period version of this problem is formulated as a mixed integer linear program. A solution technique based on Benders Decomposition is developed, implemented, and successfully applied to a real problem for a major food firm with 17 commodity classes, 14 plants, 45 possible distribution center sites, and 121 customer zones. An essentially optimal solution was found and proven with a surprisingly small number of Benders cuts. Some discussion is given concerning why this problem class appears to be so amenable to solution by Benders' method, and also concerning what we feel to be the proper professional use of the present computational technique.

The Benders reformulation

We will follow the 2005 survey of the method applied to network design problems [Costa, 2005] to introduce the main ideas of the method. Consider the following MIP:

General MIP problem:
$$z = \text{Min } c^t x + d^t y \tag{9.1}$$
s.t.
$$Ax + By \geq b, \tag{9.2}$$
$$Dy \geq e, \tag{9.3}$$
$$x \geq 0, y \geq 0 \text{ and integer.} \tag{9.4}$$

Vectors x and y are the continuous and integer variables, respectively, while c and d are the vectors of the associated costs. Matrices A, B and D and vectors b and e have the appropriate dimensions. This problem can be expressed as:

$$\min_{y \in Y} \{ d^t y + \min_{x \geq 0} \{ c^t x : Ax \geq b - By \} \},$$

where $Y = \{y \mid Dy \geq e,\ y \geq 0 \text{ and integer}\}$. The inner minimization is a linear program. Associating dual variables u to constraints $Ax \geq b - B\overline{y}$, we can write the dual version of this problem as

$$\max_{u \geq 0} \{ u^t (b - B\overline{y}) : u^t A \leq c \}.$$

This is the Benders decomposition subproblem.

Using duality theory, the primal and dual formulations can be interchanged. Therefore, (9.1) – (9.3) can be rewritten as:

$$\min_{\overline{y} \in Y} \{ d^t \overline{y} + \max_{u \geq 0} \{ u^t (b - B\overline{y}) : u^t A \leq c \} \}.$$

Note that the feasible space of the subproblem (inner maximization) is independent of the choice made for variables y. Let $F = \{u \mid u \geq 0\ ;\ u^t A \leq c\}$ represent this feasible space. We assume that F is not empty for it would correspond to a primal problem either infeasible or unbounded. F is therefore composed of extreme points u^p (for $p = 1 \ldots P$) and extreme rays r^q (for $q = 1 \ldots Q$).

The solution of the subproblem can be either bounded or unbounded. In the first case, the solution is one of the extreme points u^p $(p = 1 \ldots P)$. In the latter situation, there is a direction r^q for which $r^q(b - B\overline{y}) > 0$.[1]

The unbounded situation results in an unfeasible primal problem and must be avoided. We must therefore eliminate the values of $\overline{y}$ that would yield an unbounded inner dual problem. This is done by explicitly considering the restrictions:

$$r^q(b - B\overline{y}) \leq 0, \qquad q = 1 \ldots Q. \tag{9.5}$$

With this restrictions in the external formulation, the maximum value of the inner problem is the value of one of the extreme points of F. The Problem becomes:

$$\begin{aligned}
&\text{z = Min } d^t y + \max\{u^p(b - B)y : p = 1, \ldots, P\} \\
&\text{s.t.} \\
&r^q(b - By) \leq 0, \quad \forall q = 1, \ldots, Q, \\
&y \in Y.
\end{aligned}$$

or, with the use of an auxiliary continuous variable z:

$$\begin{aligned}
&\text{z = Min } d^t y + z && (9.6)\\
&\text{s.t.} \\
&r^q(b - By) \leq 0, \quad \forall q = 1, \ldots, Q, && (9.7)\\
&z \geq u^p(b - By), \quad \forall p = 1, \ldots, P, && (9.8)\\
&y \in Y, z \text{ free} && (9.9)
\end{aligned}$$

Formulation (9.6) – (9.9) is called the Benders reformulation.

[1] To simplify the notation in the following, assume that r^q and u^p are row vectors.

A delayed generation of constraints

The idea of the Benders decomposition algorithm is to ignore most of the initial constraints of the Benders reformulation and generate them as needed. We start with a relaxed version of the reformulation called the *relaxed master problem.*

$$
\begin{aligned}
MasterProblem \quad & \text{z = Min } d^t y + z && (9.10)\\
& \text{s.t.} \\
& z \geq -M && (9.11)\\
& y \in Y, z \text{ free} && (9.12)
\end{aligned}
$$

This gives us tentative values for the integer variables, $\overline{y}$, that are sent to the subproblem:

$$
\begin{aligned}
Subproblem \quad & \text{z = Max } u(b - B\overline{y}) && (9.13)\\
& \text{s.t.} \\
& uA \leq c && (9.14)\\
& u \geq 0 && (9.15)
\end{aligned}
$$

which returns us an extreme ray r^q or an extreme point u^p that can be used to generate a feasibility or optimality cut for the master problem (and the method iterates).

End of the method

Every time the Master problem is run, we have a dual bound for the problem (i.e., we have solved a relaxation of the problem). Every time the subproblem finds an extreme point, we have found a primal bound for the problem (i.e.,we have found a feasible solution). The method converges when the last dual bound found is equal (up to a tolerance) to the best primal bound found during the process.

Example: Consider the following MIP problem.

$z = \text{Max } x_1 - x_2 + y_1 + y_2$

s.t.

$x_1 + x_2 + y_2 \leq 2$

$-x_1 - x_2 + y_1 \leq -1$

$x_1, x_2 \geq 0, y_1, y_2 \in \{0, 1\}$

Solve using Benders decomposition.

Implementation - Direct

```
import gurobipy as gp
from gurobipy import GRB

#master
m = gp.Model()
y = m.addVars(2, vtype = GRB.BINARY, name="y")
z = m.addVar(name="z", ub=1000)
m.setObjective(y[0]  + y[1] + z, GRB.MAXIMIZE)
m.Params.OutputFlag= 0

#sub
s = gp.Model()
s.Params.InfUnbdInfo = 1
s.Params.OutputFlag= 0
x = s.addVars(2, name="x")
cons1 = s.addConstr(x[0] + x[1] <= 0, name="cons1")
cons2 = s.addConstr(-x[0] - x[1] <= 0, name="cons2")
s.setObjective(x[0] - x[1],GRB.MAXIMIZE)

#Benders loop

LB = -100000
UB = 100000

while(UB - LB >= 0.00001):

  #optimize Master
  m.optimize()
  UB = m.objVal

  #update sub
  cons1.rhs = 2 - y[1].x
  cons2.rhs = -1 - y[0].x

  #optimize sub
  s.optimize()

  #generate cuts
  if s.status == 3:
    print("Infeasibility constraint")
    u1 = cons1.getAttr('FarkasDual')
    u2 = cons2.getAttr('FarkasDual')
    expr = u1*(2 - y[1]) + u2*(-1  - y[0])
    m.addConstr(expr >= 0)
    print(expr, ">=", "0")
  else:
    print("Feasibility constraint")
    u1 = cons1.getAttr('Pi')
    u2 = cons2.getAttr('Pi')
```

```
    expr = u1*(2 - y[1]) + u2*(-1  - y[0])
    print(expr, ">=", "z")
    m.addConstr( expr >= z)
    if   y[0].x  + y[1].x  + s.objVal > LB:
      LB = y[0].x  + y[1].x  + s.objVal

  print(LB, " ", UB)

# # Display optimal values of decision variables
for v in m.getVars():
    if v.x > 1e-6:
        print(v.varName, v.x)

# Display optimal solution value
print('Total profit: ', m.objVal)
```

Implementation - Via Lazy Constraints

```
#!/usr/bin/env python3.7
import gurobipy as gp
from gurobipy import GRB


#Initial Subproblem:
def initial_sub():
  s = gp.Model()
  s.Params.InfUnbdInfo = 1
  s.Params.OutputFlag= 0
  x = s.addVars(2, name="x")
  cons1 = s.addConstr(x[0] + x[1] <= 0, name="cons1")
  cons2 = s.addConstr(-x[0] -x[1] <= 0, name="cons2")
  s.setObjective(x[0] - x[1],GRB.MAXIMIZE)
  return s

#Update subproblem
def update_subproblem(s,y1,y2):
  cons1 = s.getConstrByName("cons1")
  cons2 = s.getConstrByName("cons2")
  cons1.rhs = 2 - y2
  cons2.rhs = -1 - y1

def generate_cuts(model,where):
  if where == GRB.Callback.MIPSOL:
    valsy = model.cbGetSolution(model._y)
    print(valsy)
    update_subproblem(s,valsy[0],valsy[1])
    s.optimize()
    if s.status == 3:
      print("Infeasibility constraint")
      cons1 = s.getConstrByName("cons1")
      cons2 = s.getConstrByName("cons2")
      u1 = cons1.getAttr('FarkasDual')
      u2 = cons2.getAttr('FarkasDual')
      expr = u1*(2 - y[1]) + u2*(-1  - y[0])
      print(expr, ">=", "0")
      model.cbLazy( expr >= 0)
    else:
      print("Feasibility constraint")
      cons1 = s.getConstrByName("cons1")
      cons2 = s.getConstrByName("cons2")
      u1 = cons1.getAttr('Pi')
      u2 = cons2.getAttr('Pi')
      expr = u1*(2 - y[1]) + u2*(-1  - y[0])
      print(expr, ">=", "z")
      model.cbLazy( expr >= m._z)

s = initial_sub()
```

```
s.write("model.lp")

#master
m = gp.Model()
y = m.addVars(2, vtype = GRB.BINARY, name="y")
z = m.addVar(name="z", ub=1000)

m.setObjective(y[0]  + y[1] + z, GRB.MAXIMIZE)

m.Params.LazyConstraints = 1
m._y = y
m._z = z
m.Params.OutputFlag= 0
m.optimize(generate_cuts)

s.write("model.lp")


# Display optimal values of decision variables
for v in m.getVars():
    if v.x > 1e-6:
        print(v.varName, v.x)

# Display optimal solution value
print('Total profit: ', m.objVal)
```

Chapter 10

Lagrangian Relaxation

Relaxations

Consider optimisation problems P and RP defined as:

$$(P) \quad Z_P = \max \{c(x) : x \in X \subseteq \mathbb{R}^n\}$$
$$(RP) \quad Z_{RP} = \max \{f(x) : x \in T \subseteq \mathbb{R}^n\}$$

RP is a relaxation of P if:

(i) $X \subseteq T$,

(ii) $f(x) \geq c(x)$ for all $x \in X$.

Relations between P and RP

- $Z_{RP} \geq Z_P$:

  If x^* is an optimal solution of P, $x^* \in X \subseteq T$ and $Z_P = c(x^*) \leq f(x^*)$. As $x^* \in T$, $f(x^*)$ is a lower bound on Z_{RP} and therefore $Z_P \leq f(x^*) \leq Z^{RP}$.

- If relaxation RP is infeasible, the original problem P is also infeasible:

  RP infeasible means $T = \emptyset$ and since $X \subseteq T$, X is also empty $\Rightarrow$ P is infeasible.

- Let x^* be an optimal solution of RP. If $x^* \in X$ and $f(x^*) = c(x^*)$, then x^* is an optimal solution of P:

  As $x^* \in X$, $c(x^*)$ is a lower bound on P: $c(x^*) \leq Z_P$. As RP is a relaxation of P, $Z_{RP} = f(x^*)$ is an upper bound on P: $c(x^*) = f(x^*) \geq Z_P$. As the primal and dual bounds are the same, we know the solution is optimal.

The Lagrangian Relaxation

Consider an integer programming model in which we divide the constraints in two sets.

$$
\begin{aligned}
IP \quad & z = \max c^t x \\
& \text{s.t.} \\
& Ax \leq b, \\
& Dx \leq d, \\
& x \in \mathbb{Z}^n_+.
\end{aligned}
$$

Let $\mathcal{X} = \{x | Dx \leq d, x \in \mathbb{Z}^n_+\}$ and the problem can be rewritten as:

$$
\begin{aligned}
IP \quad & z = \max c^t x \\
& \text{s.t.} \\
& Ax \leq b, \\
& x \in \mathcal{X}.
\end{aligned}
$$

This partition is not arbitrary. We are interested in dividing the problem in a way such that the $A_{(m\times n)}x_{(n\times 1)} \leq b_{(m\times 1)}$ constraints can be seen as the *complicating* ones, i.e., the remaining problem that we obtain if these constraints are somehow ignored is much easier to solve than the complete problem.

Considering positive multipliers $u = (u_1, \ldots, u_m)$, to obtain:

$$
\begin{aligned}
IP(u) \quad & z(u) = \max c^t x + u^t(b - Ax) \\
& \text{s.t.} \\
& x \in \mathcal{X}.
\end{aligned}
$$

Theorem 4 *$IP(u)$ is a relaxation of IP*

Proof:

The first condition for it to be a relaxation is that $X \subseteq T$, i.e., the feasible space of the relaxation is at least as large as that of the original problem. This is true since we have only removed constraints from IP to $IP(u)$.

The second condition for it to be a relaxation is that $f(x) \geq c(x)$ for all $x \in X$. That is, for any feasible solution of IP, $z(u) \geq z$. This is true since for a feasible solution of (IP), $b - Ax \geq 0$ and since the multipliers are also ≥ 0, we are adding a positive value to the objective of $z(u)$ with respect to z.

The Lagrangian dual problem

As we did when introducing duality, we are interested in finding the multipliers u that will give us the best possible bound to (IP). As we defined (P) as a maximisation problem, this implies solving the following problem in the u variables:

$$W_{LD} = \min_{u}\{z(u) : u \geq 0\}$$

That is:

$$W_{LD} = \min_{u \geq 0}\{\max_{x \in X} c^t x + u^t(b - Ax)\}$$

We now show that a solution $x(u)$ of IP(u) is optimal for IP if:

(i): $x(u)$ is optimal for $IP(u)$

(ii): $Ax(u) \leq b$

(iii): $(Ax(u))_i = b_i$ whenever $u_i \geq 0$.

Proof [Wolsey, 2020]:

- By (i): $W_{\text{LD}} \leq z(u) = cx(u) + u(b - Ax(u))$.
- By (iii): $c^t x(u) + u(b - Ax(u)) = cx(u)$.
- By (ii): $x(u)$ is feasible in IP and so $c^t x(u) \leq z$. Thus, $W_{\text{LD}} \leq cx(u) + u(b - Ax(u)) = c^t x(u) \leq z$.

But as $W_{\text{LD}} \geq z$, equality holds throughout and $x(u)$ is optimal in IP.

LR for the uncapacitated facility location problem [Wolsey, 2020].

Consider the following version of the uncapacitated facility location problem.

$$z = \max \sum_{i \in M} \sum_{j \in N} c_{ij} x_{ij} - \sum_{j \in N} f_j x_j$$

s.t.

$$\sum_{j \in N} y_{ij} = 1, \quad \forall i \in M,$$

$$y_{ij} \leq x_j, \quad \forall i \in M, j \in N$$

$$y_{ij} \geq 0 \quad \forall i \in M, j \in N,$$

$$x_j \in \{0, 1\} \quad \forall j \in N.$$

What is the difference between this model and the one we studied earlier ? Dualise the demand constraints and ropose a Lagrangian relaxation for this model.

Dualising the demand constraints, we obtain:

$$z = \max \sum_{i \in M} \sum_{j \in N} (c_{ij} - u_i) y_{ij} - \sum_{j \in N} f_j x_j + \sum_{i \in N} u_i$$

s.t.

$$y_{ij} \leq x_j, \quad \forall i \in M, j \in N$$

$$y_{ij} \geq 0 \quad \forall i \in M, j \in N,$$

$$x_j \in \{0, 1\} \quad \forall j \in N.$$

Now note that the LR can be decomposed by facility:

$$z_j = \max \sum_{i \in M} \sum_{j \in N} (c_{ij} - u_i) y_{ij} - \sum_{j \in N} f_j x_j$$

s.t.

$y_{ij} \leq x_j, \quad \forall i \in M,$

$y_{ij} \geq 0 \quad \forall i \in M, j \in N,$

$x_j \in \{0, 1\} \quad \forall j \in N.$

Each of which is easily solved by inspection (how?).

Numerical example:

Consider an instance with $m = 6$ clients and $n = 5$ potential locations, fixed location costs $f = (2, 4, 5, 3, 3)$ and the client-location profit matrix:

$$(c_{ij}) = \begin{pmatrix} 6 & 2 & 1 & 3 & 5 \\ 4 & 10 & 2 & 6 & 1 \\ 3 & 2 & 4 & 1 & 3 \\ 2 & 0 & 4 & 1 & 4 \\ 1 & 8 & 6 & 2 & 5 \\ 3 & 2 & 4 & 8 & 1 \end{pmatrix}$$

Taking $u = (5, 6, 3, 2, 5, 4)$, we obtain the revised profit matrix:

$$\left(c_{ij} - u_i\right) = \begin{pmatrix} 1 & -3 & -4 & -2 & 0 \\ -2 & 4 & -4 & 0 & -5 \\ 0 & -1 & 1 & -2 & 0 \\ 0 & -2 & 2 & -1 & 2 \\ -4 & 3 & 1 & -3 & 0 \\ -1 & -2 & 0 & 4 & -3 \end{pmatrix}$$

Therefore:

Facility 1: We open the facility at cost 2 and serve clients 2, 5, 6 yielding a net value of $2-2-4-1 = -5$.

Facility 2: We open the facility at cost 4 and serve clients $1, 3, 4, 6$ yielding a net value of $4 - 3 - 1 - 2 - 2 = -4$.

Facility 3: We open the facility at cost 5 and serve clients $1, 2$ yielding a net value of $5-4-4 = -3$.

Facility 4: We open the facility at cost 3 and serve clients $1, 3, 4, 5$ yielding a net value of $3 - 2 - 2 - 1 - 3 = -5$.

Facility 5: We open the facility at cost 3 and serve clients 2, 6 yielding a net value of $3-5-3 = -5$.

To obtain the bound, we add the term $\sum_{i \in M} u_i = 25$: $LB = 25 - 5 - 4 - 3 - 5 - 5 = 3$

The strength of the Lagrangian dual

The Lagrangian dual $W_{LD} = \min\{z(u) : u \geq 0\}$, with $z(u) = \max c^t x + u^t(b - Ax)$ and subject to $x \in X$ can be written in explicit terms as:

$$W_{LD} = \min_{u \geq 0} \left\{ \max_{x \in X} [c^t x + u^t (b - Ax)] \right\}$$

Now consider all feasible points of X as $\{x_1, \ldots, x^P\}$. We can therefore write the problem as:

$$W_{LD} = \min_{u \geq 0} \left\{ \max_{p=1,\ldots,P} \left[c^t x^p + u\,(b - Ax^p) \right] \right\}$$

or, in a format we are more used to:

$W_{\text{LD}} = \min \eta$
s.t.
$\eta \geq c^t x^p + u\,(b - Ax^p) \quad \forall p = 1, \ldots, P,$
$u \in \mathbb{R}^m_+, \eta \in \mathbb{R}^1.$

In other words, this is now a linear programming problem and we can obtain its dual associating dual variables μ_p to the functional constraints:

$$W_{\text{LD}} = \max \sum_{p=1}^{P} \mu_p \left(c^t x^p\right)$$

s.t.

$$\sum_{p=1}^{P} \mu_p \left(Ax^p - b\right) \leq 0$$

$$\sum_{p=1}^{P} \mu_p = 1$$

$$\mu \in \mathbb{R}_+^P$$

which can be rewritten as:

$$W_{\text{LD}} = \max \sum_{p=1}^{P} c^t \left(\mu_p x^p\right)$$

s.t.

$$\sum_{p=1}^{P} A\mu_p x^p \leq b$$

$$\sum_{p=1}^{P} \mu_p = 1$$

$$\mu \in \mathbb{R}_+^P$$

Or, observing that $x = \mu_p x^p$ is a convex combination of the feasible points of X since $\sum_{p=1}^{P} \mu_p = 1$:

$$W_{\text{LD}} = \max\{c^t x : Ax - b \leq 0, x \in \text{conv}(X)\}$$

Now compare with the original problem and notice that the Lagrangian dual has the effect of *convexifying* the constraints in X.

Example: The strength of the Lagrangian dual

What is the effect of dualising the second constraint in the problem:

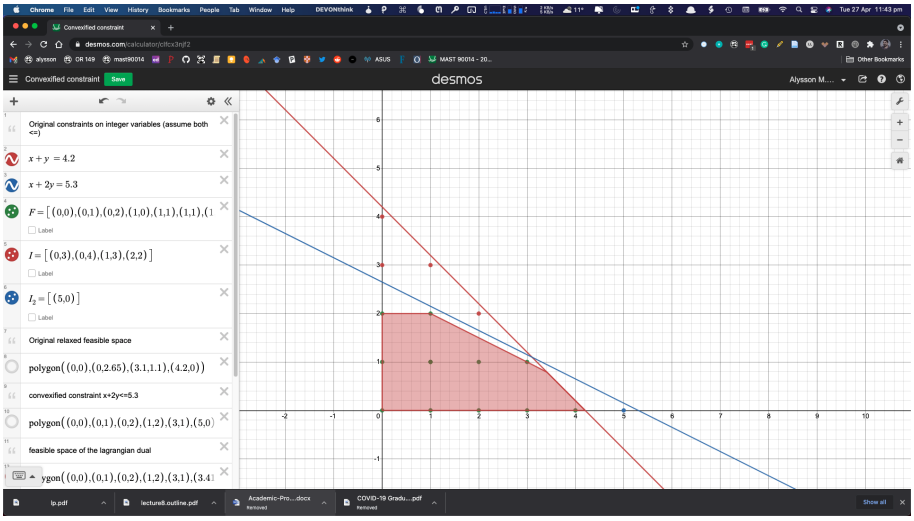
$$\max x + 1.1y$$

s.t

$$x + y \leq 4.2$$
$$x + 2y \leq 5.3$$
$$x, y \in \mathbb{Z}^+$$

$$W_{LD} = \min_{u \geq 0} \left\{ \max_{x,y \in X} x + 1.1y + u(5.3 - x - 2y) \right\}$$

with $X = x, y \in \mathbb{Z}^+ | x + y \leq 4.2$



Effect of the convexification of $x + y \leq 4.2$ (blue).

Note:: If the constraints in X *integrality property*, the Lagrangian relaxation will yield the same value as the linear relaxation.

Solving the Lagrangian dual

Consider again the Lagrangian dual problem:

$$
\begin{aligned}
& W_{\mathrm{LD}} = \min \eta \\
& \text{s.t.} \\
& \eta \geq c^t x^p + u\,(b - Ax^p) \quad \forall p = 1, \ldots, P, \\
& u \in \mathbb{R}^m_+, \eta \in \mathbb{R}^1.
\end{aligned}
$$

And the problem can be written explicitly considering all integer points in X (or only those in the convex hull).

feasible point (x^p, y^p)	associated constraint
(0,0)	$\eta \geq 0.0 + u(5.3)$
(0,1)	$\eta \geq 1.1 + u(3.3)$
(0,2)	$\eta \geq 2.2 + u(1.3)$
(0,3)	$\eta \geq 3.3 + u(-0.7)$
(0,4)	$\eta \geq 4.4 + u(-2.7)$
(1,0)	$\eta \geq 1.0 + u(4.3)$
(1,1)	$\eta \geq 2.1 + u(2.3)$
(1,2)	$\eta \geq 3.2 + u(0.3)$
(1,3)	$\eta \geq 4.3 + u(-1.7)$
(2,0)	$\eta \geq 2.0 + u(3.3)$
(2,1)	$\eta \geq 3.1 + u(1.3)$
(2,2)	$\eta \geq 4.2 + u(-0.7)$
(3,0)	$\eta \geq 3.0 + u(2.3)$
(3,1)	$\eta \geq 4.1 + u(0.3)$
(4,0)	$\eta \geq 4.0 + u(1.3)$

All points of X for the example

The subgradient method

The approach above is only possible for small problems. Consider again the Lagrangian dual problem:

$$W_{LD} = \min_{u \geq 0} \left\{ \max_{x \in X} [c^t x + u^t (b - Ax)] \right\}$$

or, with the explicit use of the feasible points of X:

$$W_{LD} = \min_{u \geq 0} \left\{ \max_{p=1,\ldots,P} \left[c^t x^p + u \, (b - Ax^p) \right] \right\}$$

Leading to the subgradient algorithm:

Algorithm 4 Subgradient algorithm

1: Let $k = 0$,
2: $u = u^k$
3: Solve the Lagrangian problem $IP(u^k)$ and obtain the optimal solution $x(u^k)$
4: Let $u^{k+1} = max\{u_i^k - \mu_k (b - Ax(u^k)_i, 0\}, i = 1 \ldots, m$,
5: k=k+1

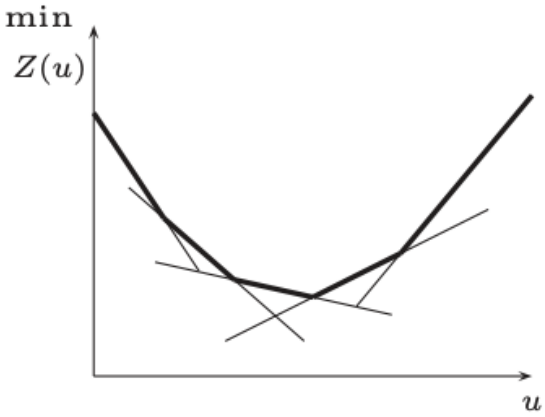
μ_k is the step size at iteration k.

Theorem 10.2 from [Wolsey, 2020] gives some ideas for how to chose its value:

(a) If $\sum_k \mu_k \rightarrow \infty$, and $\mu_k \rightarrow 0$ as $k \rightarrow \infty$, then $Z\left(u^k\right) \rightarrow W_{LD}$ the optimal value of LD.

(b) If $\mu_k = \mu_0 \rho^k$ for some parameter $\rho < 1$, then $Z\left(u^k\right) \rightarrow W_{LD}$ if μ_0 and ρ are sufficiently large.

(c) If $\bar{W} \geq W_{LD}$ and $\mu_k = \epsilon_k \left[Z\left(u^k\right) - \bar{W}\right] / \left\|b - Ax\left(u^k\right)\right\|^2$ with $0 < \epsilon_k < 2$, then $Z\left(u^k\right) \rightarrow \bar{W}$, or the algorithm finds u^k with $\bar{W} \geq Z\left(u^k\right) \geq W_{LD}$ for some finite k.



The Lagrangian dual on a single u variable

Example: subgradient algorithm

Dualise the second constraint in the problem and solve the Lagrangian dual using the subgradient algorithm.

$$\max x + 1.1y$$

s.t

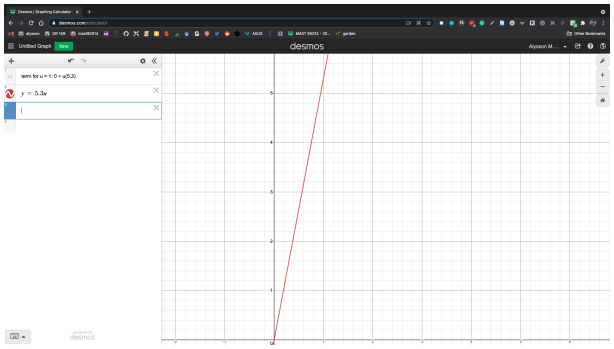
$$x + y \leq 4.2$$
$$x + 2y \leq 5.3$$
$$x, y \in \mathbb{Z}^+$$

Initialisation: $u = u_1 = 1$ and $k = 1$. Let us use a naïve $\mu_1 = 0.1$

Iteration 1:

Solving the Lagrangian dual for u_1 yields $x(u) = (0, 0)$ with $z(u) = 5.3$.

$u_2 = \max\{u_1 - 0.1(5.3 - 1 \times 0 - 2 \times 0), 0\} = 0.47$

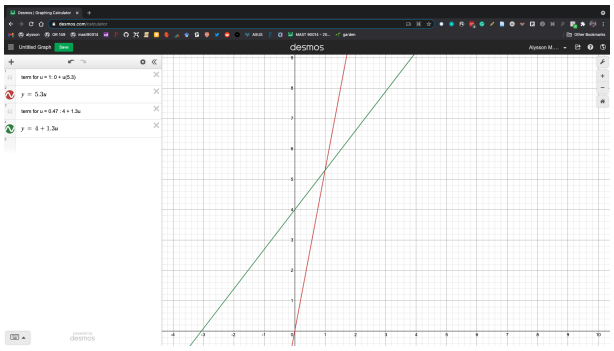


Iteration 1

Iteration 2:

Solving the Lagrangian dual for u_2 yields $x(u) = (4, 0)$ with $z(u) = 4.611$

$u_3 = \max\{0.57 - 0.1(5.3 - 1 \times 4 - 2 \times 0), 0\} = 0.04$

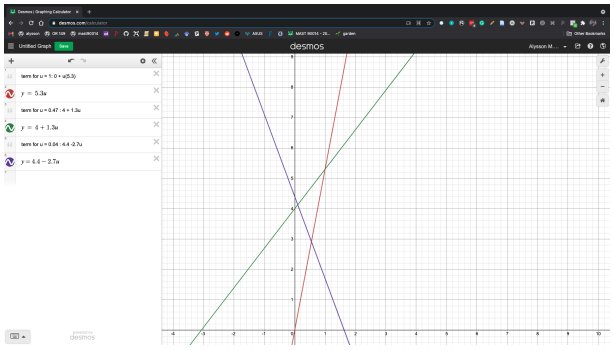


Iteration 2

Iteration 3:

Solving the Lagrangian dual for u_3 yields $x(u) = (0, 4)$ with $z(u) = 4.292$

$u_4 = \max\{0.04 - 0.1(5.3 - 1 \times 0 - 2 \times 8), 0\} = 0.31$



Iteration 3

Iteration 4:

Solving the Lagrangian dual for u_4 yields $x(u) = (4, 0)$ with $z(u) = 4.403$ (we already had that point)

$u_5 = \max\{0.31 - 0.1(5.3 - 1 \times 4 - 2 \times 0), 0\} = 0.18$

and so on. Note that we are getting closer to the optimal value of

$$u = 0.1.$$

It is probably time to halve the value of μ.

Chapter 11

A primer on Stochastic Programming

Uncertainty is a term used in subtly different ways in a number of fields, including philosophy, physics, statistics, economics, finance, insurance, psychology, sociology, engineering, and information science. It applies to predictions of future events, to physical measurements that are already made, or to the unknown. Uncertainty arises in partially observable and/or stochastic environments, as well as due to ignorance and/or indolence (Wikipedia).

In Industry, uncertainty is common: we are usually not sure about demands, resource prices, etc.

Until now, we have assumed that all data is known with certainty. In this chapter, we briefly describe how MIP can be extended to cope with uncertainty using the stochastic programming framework.

The farmer's problem [Birge and Louveaux, 2011]

Consider a European farmer who specializes in raising wheat, corn, and sugar beets on his 500 acres of land. During the winter, he wants to decide how much land to devote to each crop. (We refer to the farmer as "he" for convenience and not to imply anything about the gender of European farmers.)

The farmer knows that at least 200 tons (T) of wheat and 240 T of corn are needed for cattle feed. These amounts can be raised on the farm or bought from a wholesaler. Any production in excess of the feeding requirement would be sold. Over the last decade, mean selling prices have been \$170 and \$150 per ton of wheat and corn, respectively. The purchase prices are 40% more than this due to the wholesaler's margin and transportation costs.

Another profitable crop is sugar beet, which he expects to sell at \$36/T; however, the European Commission imposes a quota on sugar beet production. Any amount in excess of the quota can be sold only at \$10/T. The farmer's quota for next year is 6000 T. Based on past experience, the farmer knows that the mean yield on his land is roughly 2.5 T, 3 T, and 20 T per acre for wheat, corn, and sugar beets, respectively. Table 1 summarizes these data and the planting costs for these crops.

Variables:

- x_1: acres of land devoted to wheat,
- x_2: acres of land devoted to corn,
- x_3: acres of land devoted to sugar beets,
- w_1: tons of wheat sold,
- y_1: tons of wheat purchased,
- w_2: tons of corn sold,
- y_2: tons of corn purchased,
- w_3: tons of sugar beets sold at the favorable price,
- w_4: tons of sugar beets sold at capped price,

Model:

$$z = \min 150x_1 + 230x_2 + 260x_3 + 238y_1 - 170w_1 + 210y_2 - 150w_2 - 36w_3 - 10w_4$$

s.t.

$$x_1 + x_2 + x_3 \leq 500$$
$$2.5x_1 + y_1 - w_1 \geq 200$$
$$3x_2 + y_2 - w_2 \geq 240$$
$$w_3 + w_4 \leq 20x_3$$
$$w_3 \leq 6000$$
$$x_1, x_2, x_3, y_1, y_2, w_1, w_2, w_3, w_4 \geq 0.$$

The deterministic solution based on average yields:

Culture	Wheat	Corn	Sugar Beets
Surface (acres)	120	80	300
Yield (T)	300	240	6000
Sales (T)	100	–	6000
Purchase (T)	–	–	–
Overall profit: $\$118,600$			

Bad and good years

After thinking about this solution, the farmer becomes worried. He has indeed experienced quite different yields for the same crop over different years mainly because of changing weather conditions. Most crops need rain during the few weeks after seeding or planting, then sunshine is welcome for the rest of the growing period. Sunshine should, however, not turn into drought, which causes severe yield reductions. Dry weather is again beneficial during harvest. From all these factors, yields varying 20 to 25% above or below the mean yield are not unusual.

Solving the model for this data yields:

Culture	Wheat	Corn	Sugar Beets
Surface (acres)	183.33	66.67	250
Yield (T)	550	240	6000
Sales (T)	350	–	6000
Purchase (T)	–	–	–
Overall profit: $\$167,667$			

The deterministic solution based on good yields

Culture	Wheat	Corn	Sugar Beets
Surface (acres)	100	25	375
Yield (T)	200	60	6000
Sales (T)	–	–	6000
Purchase (T)	–	180	–
Overall profit: $\$59,950$			

The deterministic solution based on good yields

These solutions are called **the perfect information solution (PI)**.

We don't know what the future holds

Neither does the farmer. He needs to decide on what to plant now, before knowing what the yields will be.

A solution would be to plant according to the average, good or bad year (maybe according to how optimistic he is). What would be the long-term performance using this strategy?

```
import gurobipy as gp
from gurobipy import GRB

m = gp.Model("farmer")
x = m.addVars(range(3), name = 'x')
w = m.addVars(range(4), name = 'w')
y = m.addVars(range(2), name = 'y')

#Yields
yields = [[2.5, 3, 20], #average year
    [3, 3.6, 24], #good year +20%
    [2, 2.4, 16]] #bad year -20%


yearType = 0 # 0 average / 1 good, 2 bad

m.addConstr(x[0] + x[1] + x[2] <= 500  )
m.addConstr(yields[yearType][0]*x[0] + y[0] - w[0] >= 200  )
m.addConstr(yields[yearType][1]*x[1] + y[1] - w[1] >= 240  )
m.addConstr(w[2] + w[3] <= yields[yearType][2]*x[2]    )
m.addConstr(w[2]  <= 6000)

m.setObjective(150*x[0] + 230*x[1] + 260*x[2] + 238*y[0] - 170*w[0] + 210*y[1]
     -150*w[1] -36*w[2] -10*w[3]  )

m.optimize()

plantingAreas = [x[0].x, x[1].x, x[2].x]


#Now fix the planting variables and evaluate the result for each kind of year

solutions = []
for yearType in range(3):
  m = gp.Model("farmer")
```

```
  x = m.addVars(range(3), name = 'x')
  w = m.addVars(range(4), name = 'w')
  y = m.addVars(range(2), name = 'y')
  m.addConstrs( x[i] == plantingAreas[i] for i in range(3))


  m.addConstr(x[0] + x[1] + x[2] <= 500  )
  m.addConstr(yields[yearType][0]*x[0] + y[0] - w[0] >= 200  )
  m.addConstr(yields[yearType][1]*x[1] + y[1] - w[1] >= 240  )
  m.addConstr(w[2] + w[3] <= yields[yearType][2]*x[2]    )
  m.addConstr(w[2]  <= 6000)

  m.setObjective(150*x[0] + 230*x[1] + 260*x[2] + 238*y[0] - 170*w[0] + 210*y
    [1]  -150*w[1] -36*w[2] -10*w[3]  )

  m.optimize()
  solutions.append(m.objVal)


print(solutions)
```

Results

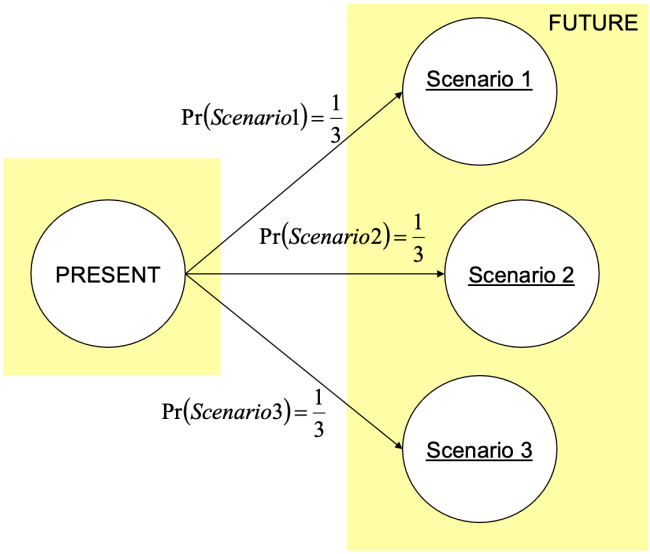
Planning for:	Profit for average	Profit for good	Profit for bad
Average	118,600	148,000,	55,120
Good	107,683	167,666	47,700
Bad	86,600	113,250	59,950

The solutions in the first row of the table are obtained when the optimal planning for the average data is implemented in all scenarios. This is sometimes called **the average solution (AV)**.

In the diagonal, we have the 'perfect information' solution.

The stochastic programming solution

In Stochastic programming, we represent the future uncertainty in scenarios, each containing a possible realisation of the problem parameters.



A two-stage scenario tree

The idea of stochastic programming is to allow for different variables to represent future decisions. The scenarios are linked via the first stage decisions (that must be taken here and now).

The stochastic programming model

Let S be the set of scenarios and p_{is} be the yields per planted acre of product i in scenario s.

Variables:

- First stage variables:

  x_1: acres of land devoted to wheat,

  x_2: acres of land devoted to corn,

  x_3: acres of land devoted to sugar beets,

- Second stage variables:

  w_{1s}: tons of wheat sold in scenario $s \in S$,

  y_{1s}: tons of wheat purchased in scenario $s \in S$,

  w_{2s}: tons of corn sold in scenario $s \in S$,

  y_{2s}: tons of corn purchased in scenario $s \in S$,

  w_{3s}: tons of sugar beets sold at the favorable price in scenario $s \in S$,

  w_{4s}: tons of sugar beets sold at capped price in scenario $s \in S$,

Model:

$$z = \min 150x_1 + 230x_2 + 260x_3 + \frac{1}{3}\sum_{s\in S}(238y_{1s} - 170w_{1s} + 210y_{2s} - 150w_{2s} - 36w_{3s} - 10w_{4s})$$

s.t.

$$x_1 + x_2 + x_3 \leq 500$$
$$p_{1s}x_1 + y_{1s} - w_{1s} \geq 200, \qquad s \in S$$
$$p_{2s}x_2 + y_{2s} - w_{2s} \geq 240, \qquad s \in S$$
$$w_{3s} + w_{4s} \leq p_{3s}x_3, \qquad s \in S$$
$$w_{3s} \leq 6000, \qquad s \in S$$
$$x_1, x_2, x_3 \geq 0$$
$$y_{1s}, y_{2s}, w_{1s}, w_{2s}, w_{3s}, w_{4s} \geq 0, \qquad s \in S.$$

This solution is known as **the stochastic solution (SP)**.

		Wheat	Corn	Sugar Beets
First Stage	Area (acres)	170	80	250
$s = 1$ Above	Yield (T)	510	288	6000
	Sales (T)	310	48	6000 (favor. price)
	Purchase (T)	–	–	–
$s = 2$ Average	Yield (T)	425	240	5000
	Sales (T)	225	–	5000 (favor. price)
	Purchase (T)	–	–	–
$s = 3$ Below	Yield (T)	340	192	4000
	Sales (T)	140	–	4000 (favor. price)
	Purchase (T)	–	48	–
	Overall profit: $108,390			

The stochastic solution [Birge and Louveaux, 2011]

Metrics for stochastic programming

As we saw before:

- SP: Solution of the stochastic model.
- PI: Average solution when each scenario is run as an independent model using the 'correct' information.
- AV: Average solution when the model once using averaged data over all scenarios.

Estimated value of perfect information (max)

$$EVPI = PI - SP$$

Estimated value of the stochastic solution (max)

$$VSS = SP - AV$$

Bibliography

[Achterberg et al., 2005] Achterberg, T., Koch, T., and Martin, A. (2005). Branching rules revisited. *Operations Research Letters*, 33(1):42–54.

[Benders, 1962] Benders, J. F. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252.

[Birge and Louveaux, 2011] Birge, J. R. and Louveaux, F. (2011). *Introduction to Stochastic Programming*. Springer Science & Business Media.

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

[Borges, 1946] Borges, J. L. (1946). On exactitude in science. *Los Anales de Buenos Aires, año 1, no. 3*.

[Costa, 2005] Costa, A. M. (2005). A survey on Benders decomposition applied to fixed-charge network design problems. *Computers & Operations Research*, 32:1429–1450.

[Dantzig et al., 1954] Dantzig, G., Fulkerson, R., and Johnson, S. (1954). Solution of a Large-Scale Traveling-Salesman Problem. *Journal of the Operations Research Society of America*, 2(4):393–410.

[Geoffrion and Graves, 1974] Geoffrion, A. M. and Graves, G. W. (1974). Multicommodity Distribution System Design by Benders Decomposition. *Management Science*, 20:822–844.

[Land and Doig, 1960] Land, A. H. and Doig, A. G. (1960). An automatic method for solving discrete programming problems. *Econometrica*, 28:497–520.

[Laporte, 2024] Laporte, G. (2024). A seminal contribution of Ailsa Land and Alison Doig Harcourt to the field of mathematical programming. *Aust. N. Z. J. Stat.*

[Miller et al., 1960] Miller, C., Zemlin, R., and Tucker, A. (1960). Integer Programming Formulation of Traveling Salesman Problems. *Journal of the ACM (JACM)*, 7(4):326–329.

[Nielsen, 1999] Nielsen, H. (1999). Algorithms for linear optimization. Technical report, Technical University of Denmark.

[Öncan et al., 2009] Öncan, T., Altınel, İ. K., and Laporte, G. (2009). A comparative analysis of several asymmetric traveling salesman problem formulations. *Computers & Operations Research*, 36(3):637–654.

[Savelsbergh, 1994] Savelsbergh, M. W. P. (1994). Preprocessing and Probing Techniques for Mixed Integer Programming Problems. *ORSA Journal on Computing*, 6(4):445–454.

[Sierksma and Zwols, 2015] Sierksma, G. and Zwols, Y. (2015). *Linear and Integer Optimization: Theory and Practice*. Chapman and Hall/CRC, 3rd edition edition.

[Toth and Vigo, 2002] Toth, P. and Vigo, D. (2002). An Overview of Vehicle Routing Problems. In *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics.

[Wolsey, 2020] Wolsey, L. A. (2020). *Integer Programming*. John Wiley & Sons.

[Zandin and Maynard, 2001] Zandin, K. B. and Maynard, H. B. (2001). *Maynard's Industrial Engineering Handbook*. McGraw-Hill Education, 5th edition edition.