42114 Integer Programming

Introductory lecture

David Pisinger, Jesper Larsen

Department of Technology, Management and Economics
Technical University of Denmark

30 August 2022

Welcome to Integer Programming

Jesper Larsen

- MSc in Computer Science at University of Copenhagen
- PhD in Operations Research at DTU in 1999 and at DTU since
- Research interest: integer programming, (public) transport optimization, healthcare planning.

David Pisinger

- MSc in Computer Science from University of Copenhagen
- PhD in Operations Research 1995 from University of Copenhagen
- Research interests: Maritime optimization, vehicle routing, offshore-wind farms, energy investment models

Todays lecture:

- Introduction to the course in general
- Basic modelling with integer variables
- (Very) Short introduction to Julia

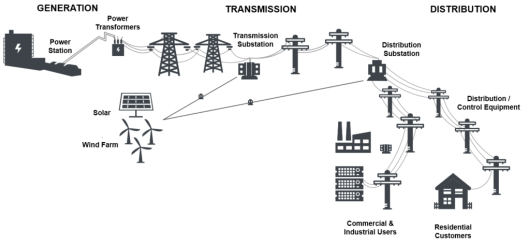
Prerequisites

- Introduction to Operations Research (42101) or a corresponding introductory course to Operations Research.
- If you haven't had an introductory course in Operations Research and do not know linear programming you will have difficulties following the course.

Applications

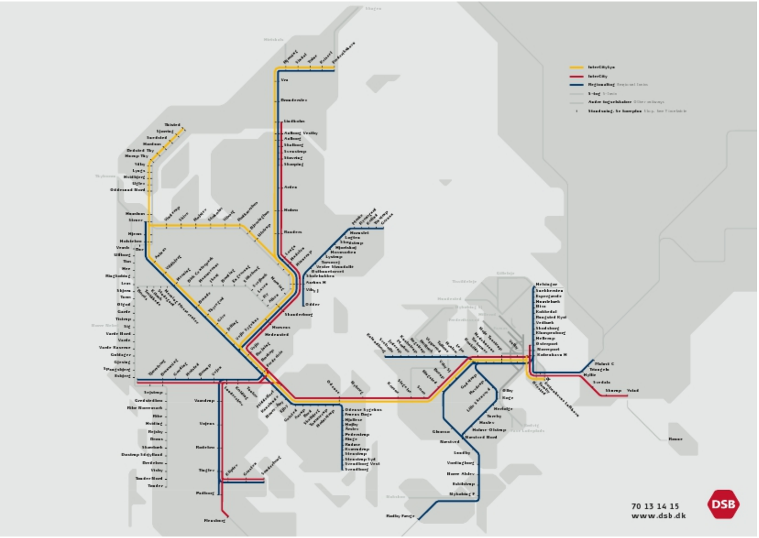
Decide: where to locate turbines

Objective: maximize power, minimize cost



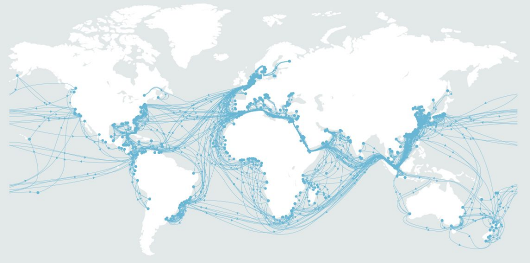
Decide: which energy modes to invest in

Objective: minimize build cost + operational costs

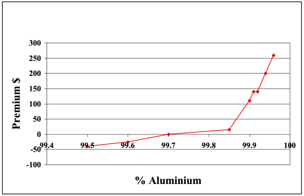


Decide: which train set to assign to which departure

Objective: minimize number of train-sets



Decide: routes, and number of vessels at each route
Objective: minimize operational costs

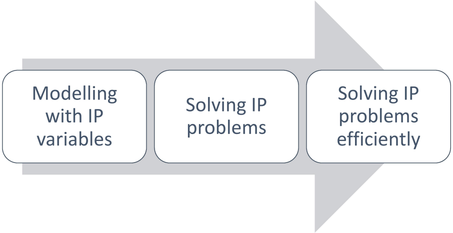


Decide: which alu-cells to combine

Objective: maximize profit

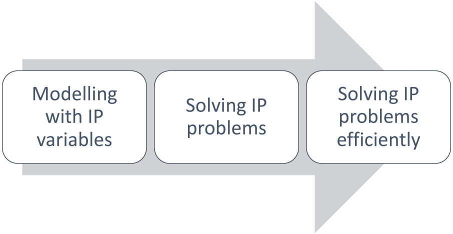
Decide: which surgery to do in which room
Objective: maximize number of patients treated

Course flow



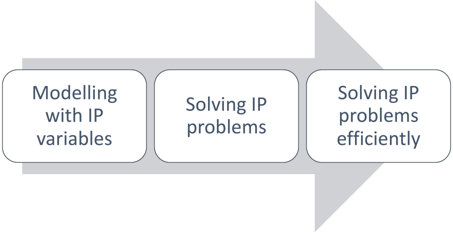
- (1) 30/8 – Introduction/Integer modelling [JLA] (Wolsey Ch 1)
- (1) 6/9 – Formulations of Integer Programs [JLA] (Wolsey Ch 1)
- (1) 13/9 Optimality, relaxation and bounds [JLA] (Wolsey Ch 2)

Course flow



- (2) 20/9 Dynamic Programming [DP] (Wolsey Ch 5)
- (2) 27/9 Branch and Bound I [DP] (Wolsey Ch 7)
- (2) 4/10 Branch and Bound II [DP] (Wolsey Ch 7)
- (2/3) 11/10 Lagrange duality [JLA] (Wolsey Ch 10)

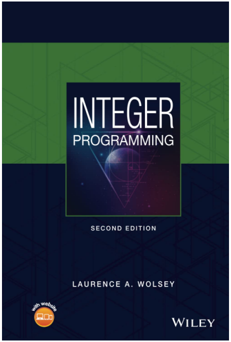
Course flow



- (2/3) 11/10 Lagrange duality [JLA] (Wolsey Ch 10)
- (3) 8/11 Cutting planes [DP] (Wolsey Ch 8)
- (3) 15/11 Strong Valid Inequalities [DP] (Wolsey Ch 9)
- (3) 22/11 Heuristics [JLA] (Wolsey Ch 12)
- (3) 29/11 Branch and Cut for the TSP (Slides) [JLA] (Wolsey Ch 6).

Course material

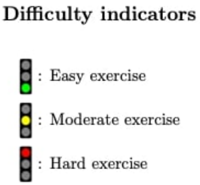
- Laurence A. Wolsey "Integer Programming" (2nd edition)
- Hand-outs and notes
- During the course we will use the programming (modelling) language Julia.
- You should install it on your own computer, and today at the exercises we can help you if you have problems.



julia

Lecture and exercises:

- **Lectures:** Building 303A, lecture hall 42: Tuesdays 15.15 - 17.00
  - There will be no streaming of lectures
  - There is a class before in the lecture hall, so entrance is possible from 15.00
- **Exercises:** Building 358, rooms 060a, 060b, 043 Tuesdays 17.00 - 19.00



Teaching assistants

Valdemar Søgaard

Kenneth Scheel

Siv Hansen

Clara Nielsen

Siv Sørensen

Eléa Prat

Jonathan Thybo

Project and evaluation

- **Project:** During the course there is compulsory project assignments. Groups of max. 3 persons. The project assignment must be **passed** in order to attend the written exam.
  - Project period will begin end September.
  - Lectures 25/10 and 1/11 is set aside for project support (no ordinary lectures)
  - Hand-in will be Friday 4/11.
- **Written exam:** Four hour written exam.
- **Grade:** 7-scale based **only** on the written exam.

What is a linear program?

Let us start with a linear program:

$$\max\{cx | Ax \leq b, x \geq 0\}$$

where A is a m by n matrix, c is a vector of size n, b a vector of size m and x is a vector of (decision) variables.

Notice

We notice linear objective function and linear constraints.

Which is equivalent to writing:

$$
\begin{array}{llllll}
\max & c_1x_1 & +c_2x_2 & \dots & +c_nx_n & & \\
\text{s.t.} & a_{11}x_1 & +a_{12}x_2 & \dots & +a_{1n}x_n & \leq & b_1 \\
 & a_{21}x_1 & +a_{22}x_2 & \dots & +a_{2n}x_n & \leq & b_2 \\
 & \vdots & \vdots & & \vdots & & \vdots \\
 & a_{m1}x_1 & +a_{m2}x_2 & \dots & +a_{mn}x_n & \leq & b_m \\
 & x_1 \geq 0, & x_2 \geq 0, & \dots & x_n \geq 0 & &
\end{array}
$$

Mixed Integer Program

Now if *some* but not all variables are integer, we have a **(linear) Mixed Integer Program (MIP)**:

$$\begin{array}{llll} \max & cx & + & hy \\ \text{s.t.} & Ax & + & Gy \leq b \\ & x \geq 0, & & y \geq 0 \text{ and integer} \end{array}$$

where A is a m by n matrix, G is a m by p matrix, c is a vector of size n, h is a vector of size h, b is a vector of size m and x is a vector of (decision) variables, and finaly y is a vector of **integer** (decision) variables.

Integer Program (IP)

If *all* variables are integer, we have an **Integer (Linear) Program**:

$$
\begin{array}{lll}
\max & cx & \\
\text{s.t} & Ax & \leq & b \\
 & x & \geq & 0 \text{ and integer}
\end{array}
$$

Binary Integer Program (BIP)

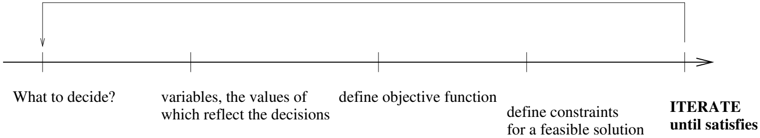
And if all variables are not only integer but restricted to the values 0 or 1, we have a **Binary Integer Program**:

$$
\begin{array}{llll}
\max & cx & & \\
\text{s.t.} & Ax & \leq & b \\
 & x & \in & \{0,1\}^n
\end{array}
$$

Sometimes we write $\mathbf{B}$ instead of $\{0,1\}$.

Modelling

Modelling in integer programming is the **art** of determining an integer programming model for the problem stated.



Characteristics of a good model:

- it is easy to understand the model,
- it is easy to compute the optimal solution.

Example 1: The 0-1-Knapsack Problem

We are going hiking. The knapsack has a capacity of b. Since not all items we would like to bring along fits in the knapsack we assign all items a weight w_i and a "profit" c_i.

- Now the goal is to choose the items that we want to bring along so that the capacity of the knapsack is not exceeded and the "profit" is maximized.

Modelling the 0-1-knapsack problem

- What should be decided?
  - For each item, do we add it to the knapsack or not:
  - For each item i define a binary variable $x_i \in \{0, 1\}$
- Objective function: Add the "profit" of all items selected:
  - $\max c_1x_1 + c_2x_2 + c_3x_3 + \ldots c_nx_n$
  - $\max \sum_{i=1}^{n} c_ix_i$
- Constraint(s)
  - In this case there is only one, the combined weight of all items selected cannot exceed the capacity of the knapsack
  - $w_1x_1 + w_2x_2 + w_3x_3 + \ldots w_nx_n \leq b$
  - $\sum_{i=1}^{n} w_ix_i \leq b$

Example in Julia

Given a 0-1-knapsack problem with five items and $b = 10$ and profit and weight as described below we can make an integer programming model.

	1	2	3	4	5
c_i	5	3	2	7	4
w_i	2	8	4	2	5

$$\begin{aligned} \max \quad & 5x_1 + 3x_2 + 2x_3 + 7x_4 + 4x_5 \\ \text{s.t.} \quad & 2x_1 + 8x_2 + 4x_3 + 2x_4 + 5x_5 \leq 10 \\ & x_1, x_2, x_3, x_4, x_5 \in \{0, 1\} \end{aligned}$$

Code

- Tell Julia we a using the modelling component and solving using the GLPK solver

```
using JuMP, GLPK
```

- Describe data for the problem

```
profit = [5, 3, 2, 7, 4]
weight = [2, 8, 4, 2, 5]
capacity = 10
```

- Define a mathematical model `model` and what solver should be used to solve it

```
model = Model(GLPK.Optimizer)
```

More code

- Define the five binary variables `x` for the problem
  `@variable(model, x[1:5], Bin)`
- Define the objective of the problem as the sum of profits times the value of the `x` variables

```
@objective(model, Max,
profit[1]*x[1]+profit[2]*x[2]+profit[3]*x[3]+
profit[4]*x[4]+profit[5]*x[5])
```

  It does become tedious to write large sums like this. Therefore Julia has an abbreviation for theses long sums. It becomes

```
@objective(model, Max, sum(profit[i]*x[i] for i=1:5))
```

  Here `i` takes on every integer value between 1 and 5 resulting in exactly the same expression as my original one. Same trick can be used when defining the single constraint of the 0-1-knapsack problem.

And a bit more code...

- Define the constraint prohibiting the solution to exceed the capacity
```
@constraint(model, sum(weight[i]*x[i] for i=1:5) <=
capacity)
```
- Solve the problem!
```
JuMP.optimize!(model)
```
- Write the values of the solution
```
println("Objective is: ", JuMP.objective_value(model))
println("Solution is:")
for i in 1:5
print(JuMP.value(x[i]), )
end
```

Running the program in Julia

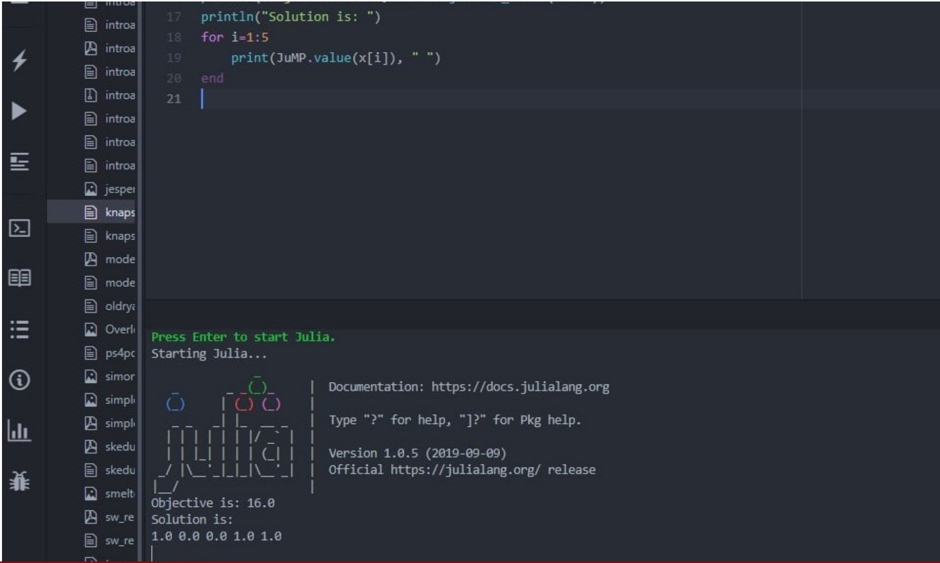
Workspace — C:\Users\jesla\Dropbox\education\02713\lectureIntroModelling — Atom

File Edit View Juno Selection Find Packages Help

knapsack.jl

```julia
using JuMP, GLPK

profit = [5, 3, 2, 7, 4]
weight = [2, 8, 4, 2, 5]
capacity = 10

model = Model(GLPK.Optimizer)

@variable(model, x[1:5], Bin)

@objective(model, Max, sum(profit[i]*x[i] for i=1:5))
@constraint(model, sum(weight[i]*x[i] for i=1:5) <= capacity)

JuMP.optimize!(model)

println("Objective is: ", JuMP.objective_value(model))
println("Solution is: ")
for i=1:5
    print(JuMP.value(x[i]), " ")
end
```

Reading the results



Example 2: The Assignment Problem

There are n people available to carry out n jobs. Each person is assign to carry out exactly one job, and each job need exactly one person assigned. There is an estimated cost c_{ij} if person i is assigned to job j.

- The goal is to the assignment with the minimum cost.

Modelling the assignment problem

- What decisions have to be made?
  - Which job is person 1 going to perform? which job is person 2 going to perform? etc.
  - Can also be put in another way: Is person 1 doing job 1? Yes/no etc.
  - For each person i and job j define a binary variable $x_{ij} \in \{0, 1\}$
- How would the objective function look like?
  - If person i is assigned job j, that is, $x_{ij} = 1$ then we incur the cost c_{ij}
  - We need to check that for all combinations of i and j
  - $\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$
- Constraints:
  - Make sure each person is only assigned one job:
  - $\sum_{j=1}^{n} x_{ij} = 1$ for each i
  - But that is not enough!!!

Last part of the assignment problem

- If we stop here we could still have a job j assigned to several persons, og a job remaining unassigned
  - We need a constraint that ensures that each job is assigned to exactly one person.
  - $\sum_{i=1}^{n} x_{ij} = 1$ for each j
- And now we are done!

Innovative use of variables in IP

- Big-M notation
- Functions with N possible values
- Or constraints
- The Travelling Salesman Problem

Big-M method
Example 3:The Uncapacitated Lot Sizing Problem

We need to schedule production over n time periods for a single product. We need to fullfill a demand of d_t in time period t. The basic model has additional data:

- f_t is the fixed cost of producing in period t
- p_t is unit production cost in period t
- h_t is unit storage cost in period t
- The goal is now to schedule our production as cheaply as possible.

Modelling the ULS problem

- Variables: x_t units produced in time period t, s_t units transfered to storage in time period t, y_t are if we produce or not.
- Objective:

$$\min \sum_{t=1}^{n} f_t y_t + p_t x_t + h_t s_t$$

- Flow balance: $s_{t-1} + x_t = d_t + s_t$ for every time period $t = 1, \ldots, n$
- But how do we model that if $y_t = 0$ then $x_t = 0$, but if $y_t = 1$ then x_t can be any (positive integer) value?
  - Let M be a "huge" number. A number larger than we would ever assign to x_t. Such a value is often not hard to find.
  - Now we can model what we want with $x_t \leq M y_t$ for every time period $t = 1, \ldots, n$

At least k out of N constraints must hold (I)

$$\left.\begin{array}{rcl} f_1(x_1, x_2, \ldots, x_n) & \leq & d_1 \\ f_2(x_1, x_2, \ldots, x_n) & \leq & d_2 \\ \vdots & \vdots & \vdots \\ f_N(x_1, x_2, \ldots, x_n) & \leq & d_N \end{array}\right\} \text{at least } k \text{ must hold}$$

This can be mastered by the following change

At least k out of N constraints must hold (II)

Use $y_i = 1$ to indicate that constraint i does hold

$$
\begin{array}{ccc}
f_1(x_1, x_2, \ldots, x_n) & \leq & d_1 + M(1 - y_1) \\
f_2(x_1, x_2, \ldots, x_n) & \leq & d_2 + M(1 - y_2) \\
\vdots & \vdots & \vdots \\
f_N(x_1, x_2, \ldots, x_n) & \leq & d_N + M(1 - y_N) \\
\sum_{i=1}^{N} y_i & = & k \\
y_i \in \{0, 1\} & &
\end{array}
$$

Or-constraints (I)

Consider a case where at least one of two constraints must hold:

$$\begin{aligned} & 3x_1 + 2x_2 \leq 18 \\ \text{or} \quad & x_1 + 4x_2 \leq 16 \end{aligned}$$

We need to reformulate into a mathematical model where all constraints specified must hold.

Or-constraints (II)

Requirements can be rewritten as:

$$
\begin{aligned}
3x_1 + 2x_2 &\leq 18 \\
x_1 + 4x_2 &\leq 16 + M
\end{aligned}
$$

or

$$
\begin{aligned}
3x_1 + 2x_2 &\leq 18 + M \\
x_1 + 4x_2 &\leq 16
\end{aligned}
$$

where M is a very very large positive number.

Or-constraints (III)

This is equivalent to

$$\begin{aligned} 3x_1 + 2x_2 &\leq 18 + My \\ x_1 + 4x_2 &\leq 16 + M(1-y) \end{aligned}$$

where y is a binary auxiliary variable. By using one binary variable for each constraint this idea can be generalised to more constraints.

Functions with N possible solutions

We have

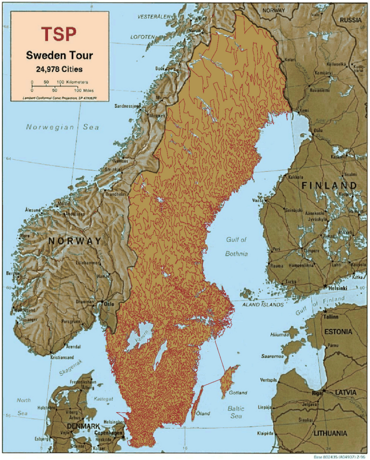
$$f(x_1, x_2, \ldots, x_n) = d_1 \text{ or } d_2 \text{ or } \ldots \text{ or } d_N$$

$f()$ could be either a variable or a constraint.

Equivalent IP is

$$\begin{aligned} f(x_1, x_2, \ldots, x_n) &= \textstyle\sum_{i=1}^{N} d_i y_i \\ \textstyle\sum_{i=1}^{N} y_i &= 1 \end{aligned}$$

The Travelling Salesman



- Tour of Sweden have 24978 nodes.
- TSP record: 528,280,881 nodes.
- Real-life applications of TSP are VLSI design and DNA sequencing.
- For more info see `www.tsp.gatech.edu`

The Travelling Salesman

- d_{ij} distance from city i to j
- binary variable x_{ij} if travel directly from i to j
- minimize

$$\sum_i \sum_j d_{ij} x_{ij}$$

The Travelling Salesman

- Leave each city i once

$$\sum_j x_{ij} = 1 \quad \text{for all } i$$

- Enter each city j once

$$\sum_i x_{ij} = 1 \quad \text{for all } j$$

The Travelling Salesman

- Subtour elimination

$$\sum_{i \in S} \sum_{i \notin S} x_{ij} \geq 1 \qquad \text{for } S \subset N, S \neq \emptyset$$

The Travelling Salesman

Alternative idea: Assign sequence number $s_i \in \{1, \ldots, n\}$

- $s_1 = 1$
- constraint:

$$x_{ij} = 1 \quad \Rightarrow \quad s_j = s_i + 1$$

- sufficient to write:

$$x_{ij} = 1 \quad \Rightarrow \quad s_j \geq s_i + 1$$

- MIP constraint:

$$s_j \geq s_i + 1 - M(1 - x_{ij})$$

- For all i, j where $j \neq 1$

Topics we have been through:

- Introduction to the course in general
- Basic modelling with integer variables
- A (very) Short introduction to Julia