Université Paris Sud

Laboratoire de Recherche en Informatique

Introduction to optimization

Fabio Martignon

Lecture calendar

- Formulations
- Relaxations
- Lower/upper bounds
- Linear Programming models
- Integer Programming models

Optimization models

- are used to find the best configuration of processes, systems, products, etc.
- rely on a theory developed mostly in the past 50 years
- applied in an industrial, financial, military context, yield a better use of budget/resources or a higher revenue

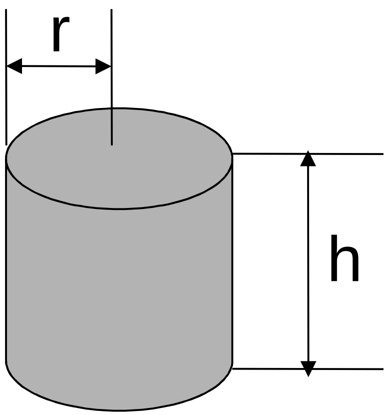
An example

- You work at a company that sells food in tin cans, and are charged with designing the next generation can, which is a cylinder made of tin
- The can must contain V = 20 cu.in. (33 cl)
- Cut and solder tin foil to produce cans
- Tin (foil) is expensive, use as little as possible)
- Design a cylinder with volume V using as little tin (i.e., total area) as possible.

An example

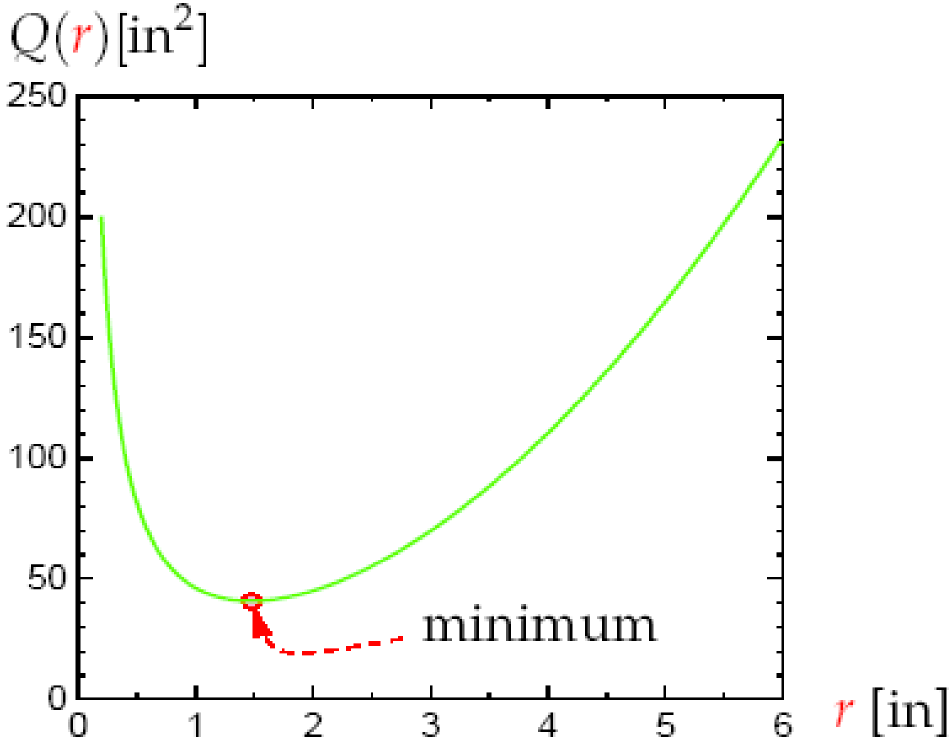
- If we knew radius $\boldsymbol{r}$ and height h,
  - the volume would be $\pi r^2 h$
  - quantity of tin would be $2\pi \boldsymbol{r}^2 + 2\pi \boldsymbol{r} h$
  - $\pi \boldsymbol{r}^2 h$ must be $V = 20$ in^3 => $h = V/\pi \boldsymbol{r}^2$

  - Rewrite the quantity of tin as $Q(\boldsymbol{r}) = 2\pi \boldsymbol{r}^2 + 2\pi \boldsymbol{r}\ V/\pi \boldsymbol{r}^2$
  - $Q(\boldsymbol{r}) = 2\pi \boldsymbol{r}^2 + 2V/\boldsymbol{r}$



Find the minimum of $Q(r)$!
Or minimize the quantity of tin!

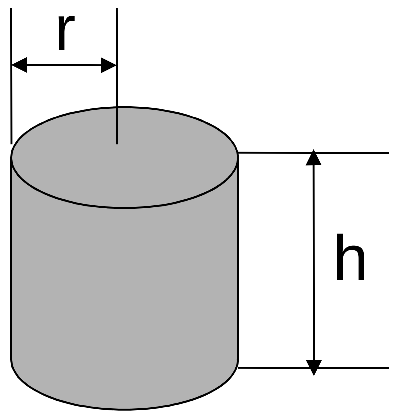
Minimize the quantity of tin



- $r = 1.471$ in
- $h = \dfrac{V}{\pi(1.471)^2} = 2.942$ in

Your First Optimization model

- **Variables:**
  - r: radius of the can's base
  - h: height of the can



- **Objective:**

  $2\pi \boldsymbol{r}^2 + 2\pi \boldsymbol{r}\, h$ (minimize)

- **Constraints:**
  - $\pi \boldsymbol{r}^2 h = V$
  - $h > 0$
  - $r > 0$

Optimization models have

- Variables: Height and radius, number of trucks, . . .

  The unknown (and desired) part of the problem (one thing your boss cares about).

- Constraints: Physical, explicit ($V = 20$ in^3), imposed by law, budget limits . . .

  They define all and only values of the variables that give possible solutions.

- Objective function: what the boss really cares about. Quantity of tin, total cost of trucks, total estimated revenue, . . .

  A function of the variables

The general optimization problem

- Consider a vector $x \in R^n$ of variables.
- An optimization problem can be expressed as:

$$
\begin{aligned}
\textbf{P}: \quad & \text{minimize} \quad f_0(x) \\
& \text{subject to} \quad f_1(x) \le b_1 \\
& \qquad f_2(x) \le b_2 \\
& \qquad f_3(x) \le b_3 \\
& \qquad \ldots \\
& \qquad f_m(x) \le b_m
\end{aligned}
$$

Feasible solutions

- Define $F = \{x \in R^n : f_1(x) \leq b_1, f_2(x) \leq b_2, \ldots, f_m(x) \leq b_m\}$, that is, F is the feasible set of an optimization problem.

- All points $x \in F$ are called feasible solutions.

Local Optimum

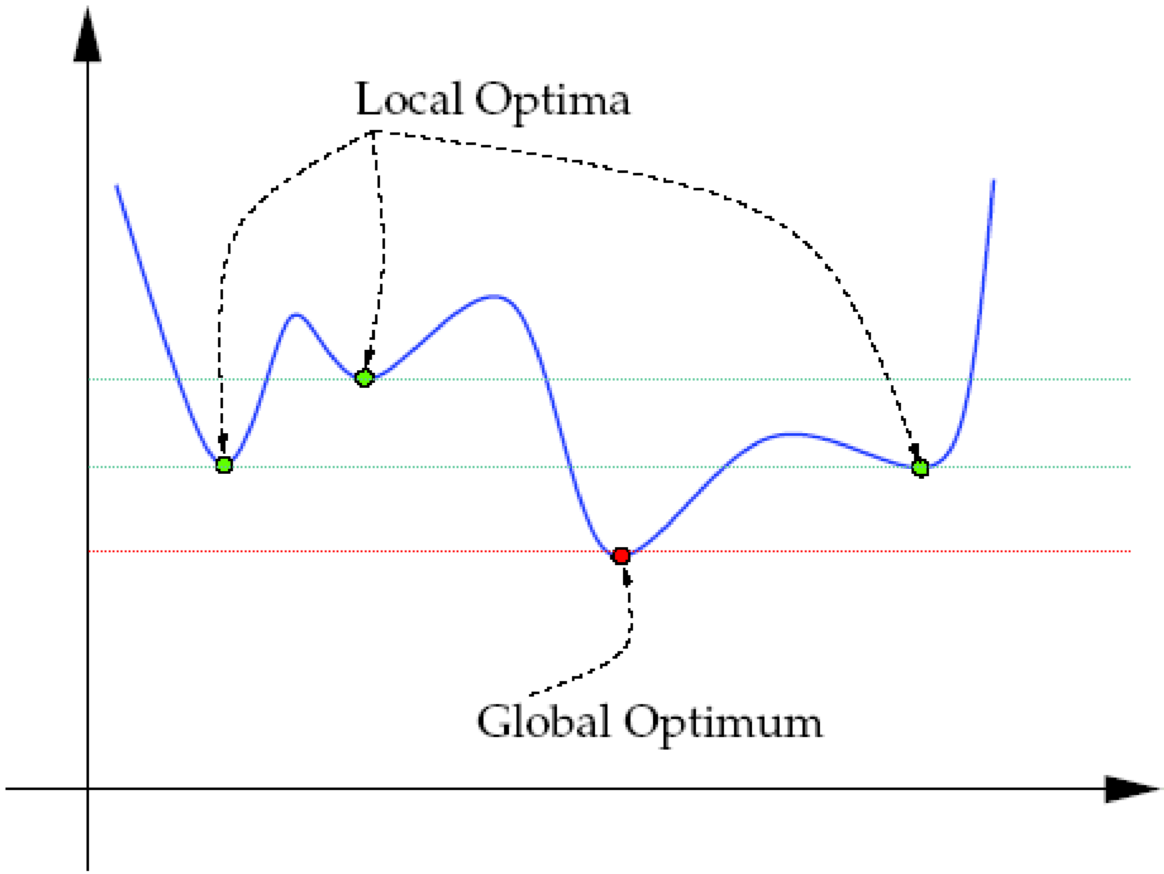
- A vector $x^l \in \mathrm{R}^n$ is a local optimum if
  - $x^l \in F$ *(a feasible solution)*
  - *there is a neighborhood* N *of* x^l *with no better point than* x^l: $f_0(x) \geq f_0(x^l)\ \forall x \in N \cap F$

Global optimum

- A vector $x^g \in R^n$ is a global optimum if
  - $x^g \in F$ *(a feasible solution)*
  - there is no $x \in F$ better than x^g, i.e.,

$$f_0(x) \geq f_0(x^g) \ \forall x \in F$$

Local optima, global optima



Relaxation of an Optimization problem

- Consider an optimization problem

$$\mathbf{P}: \quad \min \ f_0(x)$$
$$\text{s.t.} \ f_1(x) \le b_1$$
$$f_2(x) \le b_2$$
$$\ldots$$
$$f_m(x) \le b_m,$$

- Let us denote F the set of points x that satisfy all constraints (F is the feasible set):

$$F = \{x \in R^n : \ f_1(x) \le b_1,$$
$$f_2(x) \le b_2,$$
$$\ldots$$
$$f_m(x) \le b_m\}$$

- So we can denote P :min$\{f_0(x) : x \in F\}$ for short.

Relaxation of an Optimization problem

- Consider a problem P : min{ $f_0(x) : x \in F$ }.

- A problem P' :min{$f_0'(x) : x \in F'$} is a relaxation of P if:
  - $F' \supseteq F$
  - $f'_0(x) \leq f_0(x)$ for all $x \in F$.
- If **P'** is a relaxation of a problem **P**, then the global optimum of **P'** is ≤ the global optimum of **P**:

  min{$f_0'(x) : x \in F'$} ≤ min{$f_0(x) : x \in F$}

Restriction of an optimization problem

Consider again a problem

P : $\min\{f_0(x) : f_1(x) \le b_1, f_2(x) \le b_2, \ldots, f_m(x) \le b_m\}$, or

P : $\min\{f_0(x) : x \in F\}$ for short.

- deleting a constraint from P provides a relaxation of P.
- adding a constraint $f_{m+1}(x) \le b_{m+1}$ to a problem P provides a restriction of P, i.e., the opposite:

$$
\begin{aligned}
F'' = \{x \in R^n :\ & f_1(x) \le b_1, \\
& f_2(x) \le b_2, \\
& \ldots, \\
& f_m(x) \le b_m, \\
& f_{m+1}(x) \le b_{m+1}\} \subseteq F
\end{aligned}
$$

and therefore

$\min\{f_0(x) : x \in F''\} \ge \min\{f_0(x) : x \in F\}$

Lower and upper bounds

- Consider an optimization problem P : $\min\{f_0(x) : x \in F\}$: for any feasible solution $x \in F$, the corresponding objective function value $f_0(x)$ is an upper bound.
- the most interesting upper bounds are local optima.
- a lower bound of P is instead a value z such that

$$z \leq \min\{f_0(x) : x \in F\}.$$

Upper vs. Lower bounds

- Situation #1:
- You: “We found a solution that will only cost 572,000 $.”
- Boss: “Ok, that sounds good.”
- Situation #2:
- You: “We found a solution that will only cost 572,000 $.”
- Boss: “That’s too much, find something better.”
- . . .
- You: “We found another solution that costs 554,000 $.”
- Boss: “Can’t you do better than that?”
- You: “I can try again, but here’s the proof that we can’t go below 550,500 $.” ⟸ Lower bound
- Boss: “Ok then, that’s a good solution.”

What relaxations are for

- If P' is a relaxation of a problem P, then the global optimum of P' is $\leq$ the global optimum of P.
- Hence, any relaxation P' of P provides a lower bound on P.

➡ If a problem P is difficult but a relaxation P' of P is easier to solve than P itself, we can still try and solve P' :

(i) we get a lower bound and

(ii) the solution of P' may help solve P.

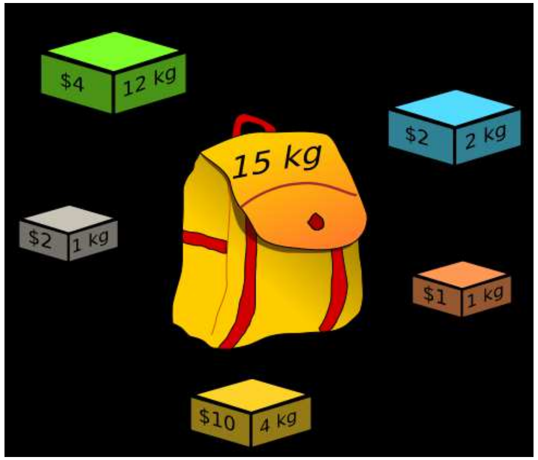
The Knapsack problem

At a flea market in Rome, you spot *n* objects (old pictures, a vessel, rusty medals . . .) that you could re-sell in your antique shop for about *double* the price.

You want these objects to pay for your flight ticket to Rome, which cost *C*.

Also, your knapsack can carry all of them, but you don't want it heavy, so you want to buy the objects that will load your knapsack as little as possible.

How do you solve this problem?



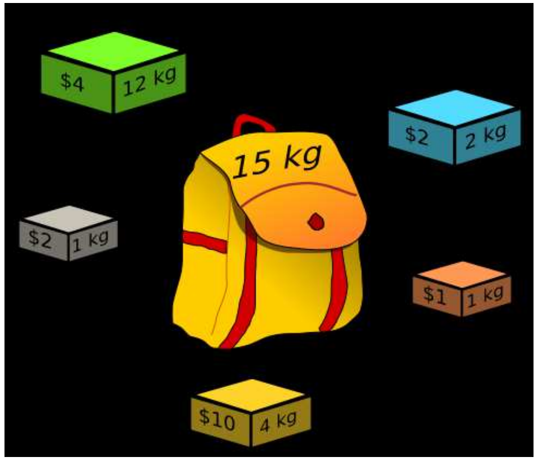
The Knapsack problem

Each object $i = 1, 2, \ldots, n$ has a price p_i and a weight w_i.

➔ Variables: one variable x_i for each $i = 1, 2 \ldots, n$. This is a "yes/no" variable, i.e., either you take the i-th object or not.

➔ Constraint: total revenue must be at least C (As you'll double the price when selling them at your store, the revenue for each object is exactly p_i)

➔ Objective function: the total weight



Your first (non-trivial) optimization model

$$\begin{array}{ll} \min & \sum_{i=1}^{n} w_i x_i \\ & \sum_{i=1}^{n} p_i x_i \geq C \\ & x_i \in \{0,1\} \quad \forall i = 1, 2, \ldots, n \end{array}$$

Let's try a relaxation: delete the only linear constraint.

$$\begin{array}{ll} \min & \sum_{i=1}^{n} w_i x_i \\ & x_i \in \{0,1\} \quad \forall i = 1, 2, \ldots, n \end{array}$$

Variables x_i are integer

What is the optimal solution of this problem? Does it give us a lower bound?

Relaxing the Knapsack problem

$$\min \sum_{i=1}^{n} w_i x_i$$
$$\sum_{i=1}^{n} p_i x_i \geq C$$
$$0 \leq x_i \leq 1 \quad \forall i = 1, 2, \ldots, n$$

x_i have real values

Relaxing *integrality* of the variables gives a relaxation where we admit fractions of objects.
It doesn't make sense physically (and monetarily . . .), but it's a relaxation, and it **does** give us a better lower bound.

Linear Programming

Canonical LP Formulation

- Consider an optimization problem

$$\mathbf{P}: \quad \min \ f_0(x)$$
$$\text{s.t.} \ f_1(x) \le b_1$$
$$f_2(x) \le b_2$$
$$\ldots$$
$$f_m(x) \le b_m,$$

P is a *linear programming problem* (LP) if $f_0 : R^n \rightarrow R$, $f_i : R^n \rightarrow R^m$ are **linear forms.**

LP in *canonical form is*:

$$\left.\begin{array}{ll} \min_x & c^\top x \\ \text{s.t.} & Ax \le b \\ & x \ge 0 \end{array}\right\} [C]$$

A is the ($m \times n$) *coefficient matrix*, *b* is the *right-hand side vector*, and *c* is the *objective coefficient vector*.

Canonical LP Formulation

We can reformulate inequalities to equations by adding a non-negative *slack variable* $x_{n+1} \geq 0$:

$$\sum_{j=1}^{n} a_j x_j \leq b \;\Rightarrow\; \sum_{j=1}^{n} a_j x_j + x_{n+1} = b \;\wedge\; x_{n+1} \geq 0$$

Standard form

A LP formulation in standard form is the following (with all inequalities transformed to equations):

$$\left.\begin{array}{rl} \min_x & (c')^\mathsf{T} x \\ \text{s.t.} & A'x = b \\ & x \geq 0 \end{array}\right\} [S]$$

Where $x = (x_1, \ldots, x_n, x_{n+1}, \ldots, x_{n+m})$, $A' = (A, I_m)$,
$c' = (c, 0, \ldots, 0)$

Of size m

The standard form is useful because linear systems of equations are computationally easier to deal with than systems of inequalities

This form is used in simplex algorithm (solver CPLEX)

Maximization problems

They are not so different from their minimization counterpart.

$$\max\{\mathbf{f}(\mathbf{x}) : \mathbf{x} \in \mathbf{F}\} \quad = \quad -\min\{-\mathbf{f}(\mathbf{x}) : \mathbf{x} \in \mathbf{F}\}$$

we should take the opposite of the objective function **only**.

Example:

max{2x − 3 : x in [4, 5]} = −min {−2x + 3 : x in [4, 5]}

7 = −(−7)

Example

Consider this problem:

$$\left.\begin{array}{ll} \max\limits_{x_1,x_2} & x_1 + x_2 \\ \text{s.t.} & x_1 + 2x_2 \leq 2 \\ & 2x_1 + x_2 \leq 2 \\ & x \geq 0 \end{array}\right\}$$

In standard form:

$$\left.\begin{array}{ll} -\min_x & -x_1 - x_2 \\ \text{s.t.} & x_1 + 2x_2 + x_3 = 2 \\ & 2x_1 + x_2 + x_4 = 2 \\ & x \geq 0 \end{array}\right\}$$

Objective function: *max f* ⟷ *-min (-f)*

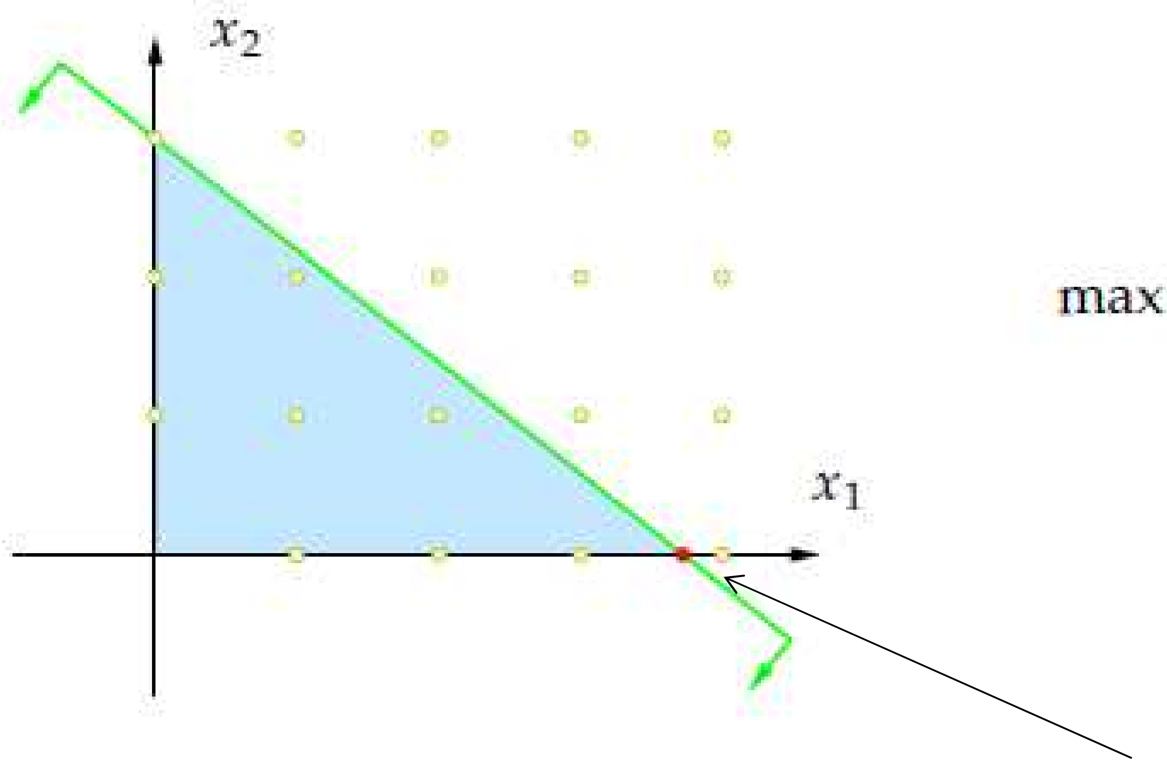
Integer Programming

Mixed-Integer Linear Programming

$$
\begin{array}{llllll}
\min & c_1x_1 & +c_2x_2 & \dots & +c_nx_n & \\
 & a_{11}x_1 & +a_{12}x_2 & \dots & +a_{1n}x_n & \leq b_1 \\
 & a_{21}x_1 & +a_{22}x_2 & \dots & +a_{2n}x_n & \leq b_2 \\
 & \vdots & & & & \\
 & a_{m1}x_1 & +a_{m2}x_2 & \dots & +a_{mn}x_n & \leq b_m \\
 & & & & x_i & \in \mathbb{Z} \quad \forall i \in J \subseteq \{1,2\dots,n\}
\end{array}
$$

➔A much more powerful modeling tool than LP:
➔yes/no decisions variables : x_i in {0,1}
➔Much more difficult than LP models.

Why can't we just round numbers up/down?



$$\begin{array}{lll} \max & x_1 & +1.01x_2 \\ & \frac{x_1}{3.7} & +\frac{x_2}{3} \leq 1 \\ & & x_1, x_2 \in \mathbb{Z}_+ \end{array}$$

- Optimal solution of the LP relaxation: (3.7, 0), obj. f.: 3.7
- Instead, the optimal solution of the original problem: (0, 3), obj. f.: 3.03
- Hence, the LP relaxation solution is completely different from the integer solution

Binary variables, logical operators

- model yes/no decisions: x_i in {0, 1}
- x_i = 0 if the decision is “no”,
- x_i = 1 if it is “yes”
- can use logical operators: implications, disjunctions, etc.:
- Kevin or Daniel will have ice cream, but not both:
- $x_{Kevin} + x_{Daniel} <= 1$
- At least one among Kevin and Daniel will have ice cream:
- $x_{Kevin} + x_{Daniel} >= 1$
- If Kevin has ice cream, then Michel will have one too:
- $x_{kevin} <= x_{Michel}$
- Daniel gets ice cream if and only if Mario does not get any:
- $x_{Daniel} = 1 - x_{Mario}$

Binary variables and operations with sets

Binary variables are useful to model problems on sets. E.g.:

- Choose a subset S of a set A of elements such that S has certain properties (e.g. not more than K elements, etc.)
- Each element i in A has a cost c_a
- The cost of a solution S is $\sum_{i \in S} c_i$
- Define variable x_i:

$$x_i = \begin{cases} 1 & \text{if } i \in S \\ 0 & \text{otherwise} \end{cases}$$

- Now the cost of a solution S is: $$\sum_{i \in A : x_i = 1} c_i = \sum_{i \in A} c_i x_i$$

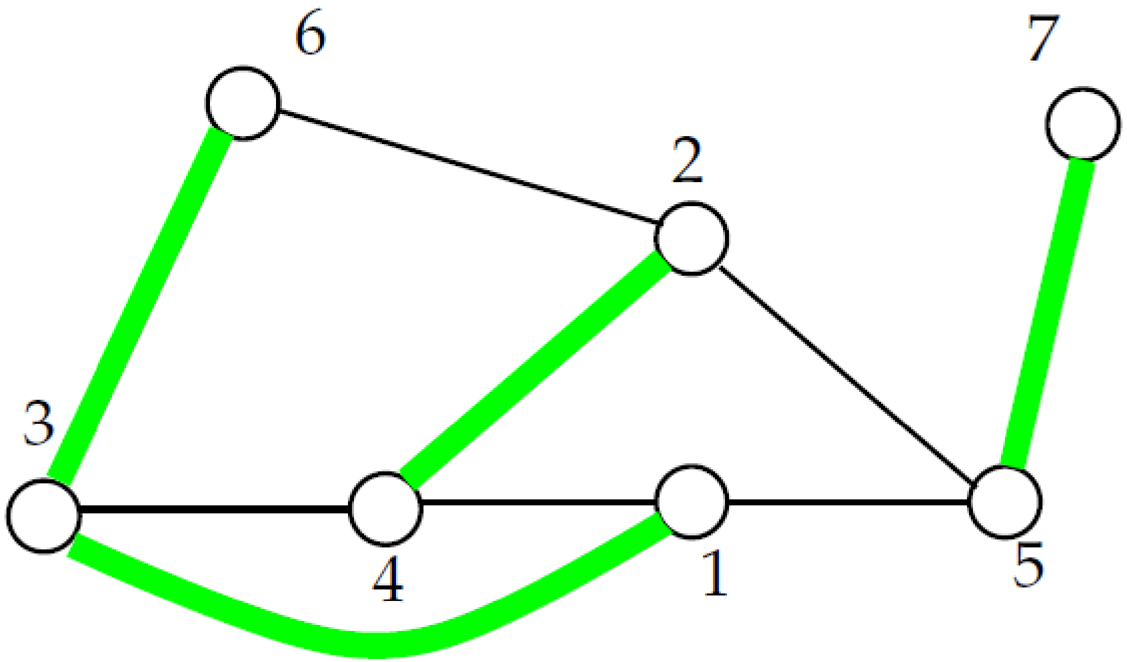
Binary variables and operations with sets

→define properties similarly, e.g.

$$|S| \leq K \text{ is } \sum_{i \in A} x_i \leq K$$

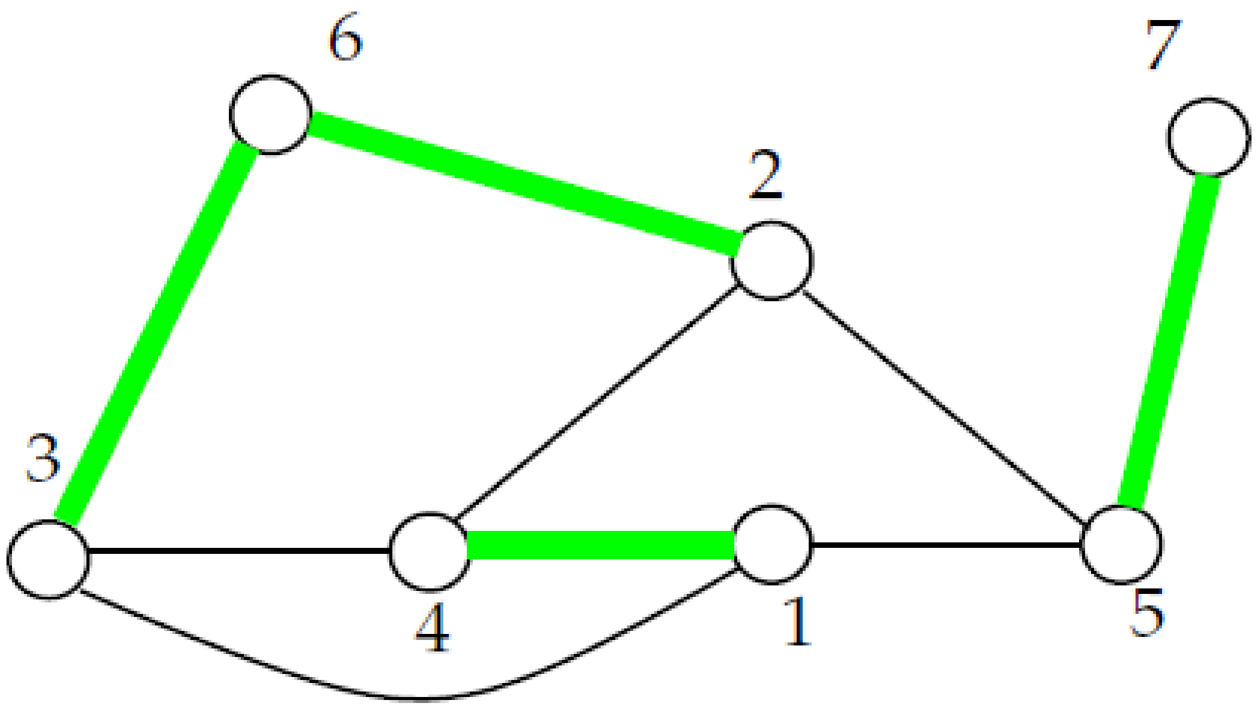
Example: the edge covering problem

→In a graph *G* = *(V, E)* as in the figure, choose a subset *S* of edges such that all nodes are "covered" by at least one edge in *S*. Minimize the number of edges used.



Example: the edge covering problem

→In a graph $G = (V, E)$ as in the figure, choose a subset S of edges such that all nodes are "covered" by at least one edge in S. Minimize the number of edges used.



Mathematical formulation of the edge covering problem

min	x_{13}	$+x_{14}$	$+x_{15}$	$+x_{24}$	$+x_{25}$	$+x_{26}$	$+x_{34}$	$+x_{36}$	$+x_{57}$	
Node 1	x_{13}	$+x_{14}$	$+x_{15}$							≥ 1
Node 2				x_{24}	$+x_{25}$	$+x_{26}$				≥ 1
	x_{13}						$+x_{34}$	$+x_{36}$		≥ 1
		x_{14}		$+x_{24}$			$+x_{34}$			≥ 1
			x_{15}		$+x_{25}$				$+x_{57}$	≥ 1
						$+x_{26}$		$+x_{36}$		≥ 1
Node 7									x_{57}	≥ 1
	$x_{13},$	$x_{14},$	$x_{15},$	$x_{24},$	$x_{25},$	$x_{26},$	$x_{34},$	$x_{36},$	x_{57}	$\in \{0, 1\}$

X_{13} =1 means that the edge (1,3) is selected

Mathematical formulation of the edge covering problem

$$\min \sum_{\{i,j\}\in E} x_{\{i,j\}}$$

$$\sum_{j\in V:\{i,j\}\in E} x_{\{i,j\}} \geq 1 \qquad \forall i \in V$$

$$x_{\{i,j\}} \in \{0,1\} \qquad \forall \{i,j\} \in E$$

AMPL (A Modeling Language for Mathematical Programming)

http://www.ampl.com/

edge_covering.model:

```
param n; # number of nodes
set V=1..n; # set of nodes
set E within {i in V, j in V: i<j};# subset of set of node pairs
var x {E} binary;
minimize numEdges: sum {(i,j) in E} x[i,j];
subject to covering_constraint {i in V}:
sum {j in V: (i,j) in E} x[i,j] + sum {j in V: (j,i) in E} x[j,i] >= 1;
```

edge_covering.data:

```
param n := 7;
set E := (1,3) (1,4) (1,5) (2,4) (2,5) (2,6) (3,4) (3,6) (5,7);
```

Cplex (a Mathematical Programming Solver)

runfile.run

```
model edge_covering.mod;
data edge_covering.dat;
option solver 'cplexamp';
option log_file 'ffile.log';
option ...;
solve;
# This is a comment.

#display the processing time (in seconds) to get the optimal solution
display _solve_user_time > results_processingTime.out;

display numEdges > results_numEdges.out;

# display the edges chosen by the solution
display x > results_chosenEdges.out;
quit;
```

Solution

```
martignon@XXX:~/ExampleAMPL$ cat results_processingTime.out
_solve_user_time = 0.008

martignon@XXX:~/ExampleAMPL$ cat results_numEdges.out
numEdges = 4

martignon@XXX:~/ExampleAMPL$ cat results_chosenEdges.out
x :=
1 3   0
1 4   1
1 5   0
2 4   0
2 5   0
2 6   1
3 4   0
3 6   1
5 7   1;
```