

# Heuristic Algorithms for Combinatorial Optimisation problems

Ph.D. course in Computer Science

Roberto Cordone  
DI - Università degli Studi di Milano



E-mail: [roberto.cordone@unimi.it](mailto:roberto.cordone@unimi.it)

Web page: <https://homes.di.unimi.it/cordone/courses/2025-haco/2025-haco.html>

# Aims of the course

This course aims to

- 1 show that **heuristic algorithms are not recipes for specific problems:**  
**heuristics and problems can be matched freely**  
*(of course, with different performance)*
- 2 discuss the **common and general aspects** of these algorithms
- 3 teach **how to design** a heuristic for a specific problem
- 4 teach **how to evaluate its performance**

*eurisko = I find*

It is a word derived from Greek

- inspired by the famous story of Archimedes and the golden crown



but it was

- never used by the ancient Greeks
- coined during the 19th century

# Some historical facts

- 4th century CE: Pappus of Alexandria discusses the *analytomenos* (*treasure of analysis*), that is **how to build a mathematical proof**
  - how to move from the hypotheses to the thesis of a theorem
  - how to move from the data to the solution of a geometrical problem
- 17th century: Descartes, Leibnitz *et al.* discuss the *ars inveniendi* (*art of finding*), *i. e.* the **attainment of truth through mathematics**
- 19th century: Bernard Bolzano discusses in detail the most common strategies to **build mathematical proofs** (*Erfindungskunst*)
- 19th-20th century: philosophers, psychologists and economists define **heuristics** as practical and simple **decision rules** that do not aim at an optimal result, but at a **satisficing** one (Simon, 1957)
- 1945: the short essay *How to solve it* by György Pólya comes back to the mathematical meaning of heuristic as an **informal process that leads to prove a thesis or to find a solution**

So, what about *heuristic algorithms*?

# Algorithms and heuristics

Some scientific sectors use the two words as opposites:

- **algorithm** as a **formal, deterministic procedure, consisting of a finite sequence of elementary steps**
- **heuristic** as an **informal, creative, open rule**

One could even say that

- **an algorithm is a correctness proof**
- **a heuristic is a bunch of common sense arguments**

In fact, an algorithm has a correctness proof, a heuristic has none

The phrase **heuristic algorithm** is an oxymoron, in some respects

*Then what does it mean?*

# Heuristic algorithms

A heuristic algorithm is an algorithm which does not guarantee a correct solution

*Then it is useless!*

Quite to the contrary, it can be useful, provided that

- 1 it “costs” much less than a correct algorithm:  
this requires a definition of **computational cost** of an algorithm
  - time
  - space
- 2 it “frequently” yields something “close” to the correct solution:  
this requires to define a solution space endowed with
  - a **metric** to express a “satisfactory distance” from the correct solution
  - a **probabilistic distribution** to express the “satisfactory frequency” of solutions at a satisfactory distance from the correct solutions

# Proofs and algorithms

Mathematical proofs and algorithms are strictly related

- every algorithm has/is a correctness proof
- both are mechanical symbolic transformations from a starting point (hypotheses/data) to an ending point (thesis/solution)
- Turing's undecidability proof mirrors Gödel's incompleteness proof

Heuristics are the construction of both proofs and algorithms

- in case of success, the heuristic is abandoned and the proof preserved
- otherwise, a good heuristic frequently provides a good result, instead of always providing a perfect one

*This is the motivation for heuristic algorithms*

# The focus of this course

The course focuses on heuristic algorithms

- that apply to **Combinatorial Optimisation** problems
- that are **solution-based** (as opposed to **model-based**)

So, we limit

- ① the kind of problem
- ② the kind of algorithm

*It is still a pretty wide field*

Let us further discuss the two limitations



# Problem classification

A problem is a question on a mathematical system

Problems can be classified based on the nature of their solution:

- **decision problems**: their solution is either *True* or *False*
- **search problems**: their solution is *any feasible subsystem* (that is, satisfying certain conditions)
- **optimisation problems**: their solution is the *minimum or maximum value of an objective function defined on the feasible subsystems*
- **counting problems**: their solution is the *number of feasible subsystems*
- **enumeration problem**: their solution is the *collection of all feasible subsystems*
- ...

We address the combination of optimisation and search, that is, we look for the optimal value and a subsystem assuming that value

# Optimisation/search problems

An optimisation/search problem can be represented as

$$\begin{aligned} &\text{opt } f(x) \\ &x \in X \end{aligned}$$

where

- a **solution**  $x$  describes **each subsystem** of the problem
- the **feasible region**  $X$  (**feasible solution space**) is the **set of subsystems which satisfy given conditions**
- the **objective function**  $f : X \rightarrow \mathbb{R}$  **quantitatively measures the quality of each subsystem** ( $\text{opt} \in \{\text{min}, \text{max}\}$ )

The problem consists in determining

- optimisation: the **optimal value**  $f^*$  of the objective function:

$$f^* = \underset{x \in X}{\text{opt}} f(x)$$

- search: at least one **optimal solution**, that is a subsystem

$$x^* \in X^* = \underset{x \in X}{\text{arg opt}} f(x) = \left\{ x^* \in X : f(x^*) = \underset{x \in X}{\text{opt}} f(x) \right\}$$

# Why focusing on optimisation/search problem?

- **applications:** objects with extreme values of an evaluation function tend to be very useful
  - low energy protein structures
  - highly influential/influenciable groups of individuals in social networks
  - low violation partitions of points into regular shapes (classifications)
  - small expressions of logical functions
  - ...

Exact optimality is costly, not always required, or even desirable  
(*many heuristic solutions could be preferable to a single exact one*)

- **hard decision/search problems reduce to optimisation/search** by **relaxing the complicating constraints**
  - enlarge the feasible region from  $X$  to  $X' \supset X$  to make the search easy;
  - quantify the distance  $d(x)$  of every  $x \in X'$  from  $X$ ;
  - minimise  $d(x)$  in  $X'$ :  $d(x^*) = 0 \Leftrightarrow x^* \in X$
- **enumeration problems of Paretian frontiers** (compromises among conflicting objectives) **directly adapt optimisation/search algorithms**

# Combinatorial Optimisation (CO)

A problem is a CO problem when **the feasible region  $X$  is a finite set**, that is, **it has a finite number of feasible solutions**

*This looks like a very restrictive assumption*

However, the study of CO problems can be useful more in general:

- ① infinite discrete problems can have a finite set of interesting solutions
- ② some continuous problems can be reduced to CO problems  
(e. g., *Linear Programming, Maximum Flow, Minimum Cost Flow*)
- ③ continuous problems can be reduced to discrete ones by sampling  
(*usually not very effective*)
- ④ ideas conceived for CO problems can be extended to other problems  
(*often quite effective*)

# Model-based heuristics

They describe the feasible region  $X$  with a “model”

A typical example is a Mathematical Programming formulation

$$\begin{array}{l} \text{opt } f(x) \\ x \in X \end{array} \quad \longrightarrow \quad \begin{array}{l} \min \phi(\xi) \\ g_i(\xi) \leq 0 \quad i = 1, \dots, m \end{array}$$

where

- $\xi \in \mathbb{R}^n$ , that is, a solution is a vector of  $n$  real values
- $X = \{\xi \in \mathbb{R}^n : g_i(\xi) \leq 0, i = 1, \dots, m\}$ , that is, the feasible region is the set of vectors which satisfy all the inequalities (constraints)

Model-based heuristics exploit the information derived from the model, that is the analytical properties of functions  $\phi$  and  $g_i$  ( $i = 1, \dots, m$ )

Other models can be based on SAT, etc. . .

*We will not use these tools*

# An alternative definition of $CO$

A problem is a  $CO$  problem when:

- ① the number of feasible solutions is finite
- ② the feasible region is  $X \subseteq 2^B$  for a given finite ground set  $B$ , that is, the feasible solutions are all subsets of the ground set that satisfy suitable conditions

The two definitions are equivalent:

$2 \Rightarrow 1$ : if the ground set  $B$  is finite, every collection  $X \subseteq 2^B$  is finite

$1 \Rightarrow 2$ : if the number of feasible solutions is finite, define  $B$  as their set and the feasible region  $X$  as the collection of all singletons of  $B$   
(a “solution” is a set containing a single solution)

In general, the sophisticated definition allows a deeper analysis, because

- $X$  is not simply enumerated
- $X$  is defined in a compact and significant way

Solution-based heuristics consider solutions as subsets of the ground set

① constructive/destructive heuristics:

- they start from an extremely simple subset (respectively,  $\emptyset$  or  $B$ )
- they add/remove elements until they obtain the desired solution

② exchange heuristics:

- they start from a subset obtained in any way
- they exchange elements until they obtain the desired solution

③ recombination heuristics:

- they start from a population of subsets obtained in any way
- they recombine different subsets producing a new population

Heuristic designers can creatively combine elements from different classes

# Randomisation and memory

Two other distinctions concern

- the use of **randomisation**:
  - **deterministic** heuristics, whose input includes only certain information
  - **randomized** heuristics, whose input includes **pseudorandom numbers**  
(*they are deterministic algorithms anyway*)
- the use of **memory**:
  - heuristics whose input includes only current information
  - heuristics whose input also includes **previously generated solutions**

These distinctions are independent from the previous classification

**Metaheuristics** (from the Greek, “beyond heuristics”) is the common name for **heuristic algorithms with randomisation and/or memory**



# Risks to beware of

- 1 **reverential or trendy attitude**, that is choosing an algorithm based on the social context, instead of the problem
- 2 **magic attitude**, that is trusting a method on the basis of an analogy with physical and natural phenomena
- 3 **heuristic integralism**, that is using a heuristic for a problem which admits exact algorithms
- 4 **number crunching**, that is performing sophisticated and complex computations with unreliable numbers
- 5 **SUV attitude**, that is relying on hardware power
- 6 **overcomplication**, that is introducing redundant components and parameters, as if that could only improve the result
- 7 **overfitting**, that is adapting components and parameters of the algorithm to the specific dataset used in the experimental evaluation

It is fundamental to

- free oneself from prejudices
- evaluate the performance of the algorithm in a scientific way
- distinguish the contribution of each component of the algorithm
- efficiently implement each component of the algorithm

$$\begin{aligned} \text{opt } & f(x) \\ & x \in X \end{aligned}$$

where  $X \subseteq 2^B$  and  $B$  finite

We will survey a number of problem classes

- set problems
- logic function problems
- numerical matrix problems
- graph problems

# Why a problem survey?

Reviewing several problems is useful because

- abstract ideas must be concretely applied to different algorithms for different problems
- the same idea can have different effectiveness on different problems
- some ideas only work on problems with a specific structure
- different problems could have nonapparent relations, which could be exploited to design algorithms

So, a good knowledge of several problems teaches how to

- apply abstract ideas to new problems
- find and exploit relations between known and new problems

*Sure, the “Magical Number Seven” risk exists. . .*

To control it, we will make some **interludes** devoted to general remarks

# Weighted set problems: Knapsack Problem ( $KP$ )

Given

- a **set**  $E$  of **elementary objects**
- a **function**  $v : E \rightarrow \mathbb{N}$  describing the **volume** of each object
- a **number**  $V \in \mathbb{N}$  describing the **capacity** of a knapsack
- a **function**  $\phi : E \rightarrow \mathbb{N}$  describing the **value** of each object

select a subset of objects of maximum value that respects the capacity

The **ground set** is trivially the set of the objects:  $B = E$

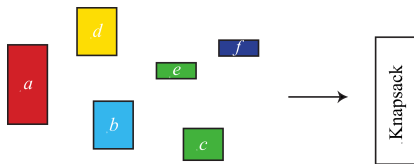
The **feasible region** includes all **subsets** of objects whose **total volume** does not exceed the capacity of the knapsack

$$X = \left\{ x \subseteq B : \sum_{j \in x} v_j \leq V \right\}$$

The **objective** is to **maximise** the total value of the chosen objects

$$\max_{x \in X} f(x) = \sum_{j \in x} \phi_j$$

# Example



$E$	a	b	c	d	e	f
$\phi$	7	2	4	5	4	1
$v$	5	3	2	3	1	1

$$V = 8$$



$$x' = \{c, d, e\} \in X$$
$$f(x') = 13$$



$$x'' = \{a, c, d\} \notin X$$
$$f(x'') = 16$$

# Set problems in metric spaces:

## *Maximum Diversity Problem (MDP)*

Given

- a set  $P$  of points
  - a function  $d : P \times P \rightarrow \mathbb{N}$  providing the distance between point pairs
  - a number  $k \in \{1, \dots, |P|\}$  that is the number of points to select
- select a subset of  $k$  points with the maximum total pairwise distance

The **ground set** is the set of points:  $B = P$

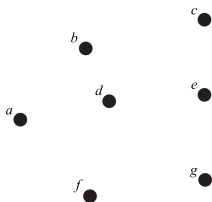
The **feasible region** includes all subsets of  $k$  points

$$X = \{x \subseteq B : |x| = k\}$$

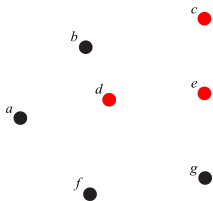
The **objective** is to maximise the sum of all pairwise distances between the selected points

$$\max_{x \in X} f(x) = \sum_{(i,j): i,j \in x} d_{ij}$$

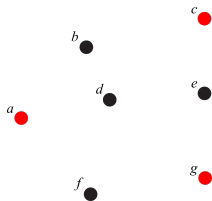
# Example



$k = 3$



$$x' = \{C, D, E\} \in X$$
$$f(x') = 24$$



$$x'' = \{A, C, G\} \in X$$
$$f(x'') = 46$$

# Interlude 1: the objective function

The objective function associates integer values to feasible subsets

$$f : X \rightarrow \mathbb{N}$$

Computing the objective function can be complex (even exhaustive)

We have seen two simple cases

- the *KP* has an **additive objective function** which sums values of an auxiliary function defined on the ground set

$$\phi : B \rightarrow \mathbb{N} \text{ induces } f(x) = \sum_{j \in x} \phi_j : X \rightarrow \mathbb{N}$$

- the *MDP* has a quadratic objective function

Both are defined not only on  $X$ , but on the whole of  $2^B$  (is this useful?)

Both are easy to compute, but the additive functions  $f(x)$  are also fast to recompute if subset  $x$  changes slightly: it is enough to

- sum  $\phi_j$  for each element  $j$  added to  $x$
- subtract  $\phi_j$  for each element  $j$  removed from  $x$

For quadratic functions, this seems more complex (we will talk about it)



# Partitioning set problems: *Bin Packing Problem (BPP)*

Given

- a set  $E$  of elementary objects
- a function  $v : E \rightarrow \mathbb{N}$  describing the volume of each object
- a set  $C$  of containers
- a number  $V \in \mathbb{N}$  that is the volume of the containers

divide the objects into the minimum number of containers respecting the capacity

The ground set  $B = E \times C$  includes all (object,container) pairs

The feasible region includes all partitions of the objects among the containers not exceeding the capacity of any container

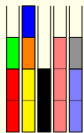
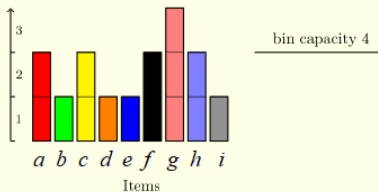
$$X = \left\{ x \subseteq B : |x \cap B_e| = 1 \forall e \in E, \sum_{(e,c) \in x \cap B^c} v_e \leq V \forall c \in C \right\}$$

with  $B_e = \{(i,j) \in B : i = e\}$  and  $B^c = \{(i,j) \in B : j = c\}$

The objective is to minimise the number of containers used

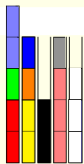
$$\min_{x \in X} f(x) = |\{c \in C : x \cap B^c \neq \emptyset\}|$$

# Example



$$x' = \{(a, 1), (b, 1), (c, 2), (d, 2), (e, 2), (f, 3), (g, 4), (h, 5), (i, 5)\} \in X$$

$$f(x') = 5$$



$$x'' = \{(a, 1), (b, 1), (c, 2), (d, 2), (e, 2), (f, 3), (g, 4), (h, 1), (i, 4)\} \notin X$$

$$f(x'') = 4$$

# Partitioning set problems:

## *Parallel Machine Scheduling Problem (PMSP)*

Given

- a set  $T$  of tasks
- a function  $d : T \rightarrow \mathbb{N}$  describing the time length of each task
- a set  $M$  of machines

divide the tasks among the machines with the minimum completion time

The ground set  $B = T \times M$  includes all (task,machine) pairs

The feasible region includes all partitions of tasks among machines  
(the order of the tasks is irrelevant!)

$$X = \left\{ x \subseteq B : |x \cap B_t| = 1 \forall t \in T \right\}$$

The objective is to minimise the maximum sum of time lengths for each machine

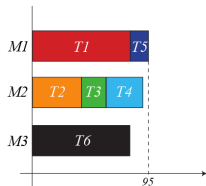
$$\min_{x \in X} f(x) = \max_{m \in M} \sum_{t: (t,m) \in x} d_t$$

# Example

$$T = \{T1, T2, T3, T4, T5, T6\}$$

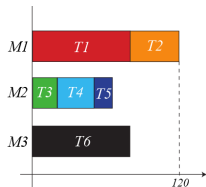
$$M = \{M1, M2, M3\}$$

task	T1	T2	T3	T4	T5	T6
d	80	40	20	30	15	80



$$x' = \{(T1, M1), (T2, M2), (T3, M2), (T4, M2), (T5, M1), (T6, M3)\} \in X$$

$$f(x') = 95$$



$$x'' = \{(T1, M1), (T2, M1), (T3, M2), (T4, M2), (T5, M2), (T6, M3)\} \in X$$

$$f(x'') = 120$$

## Interlude 2: the objective function again

The objective function of the *BPP* and the *PMSP*

- is not additive
- is not trivial to compute (*but not hard, as well*)

Small changes in the solution have a variable impact on the objective

- equal to the time length of the moved tasks (increase or decrease) (e.g., move *T5* on *M1* in  $x''$ )
- zero (e.g., move *T5* on *M3* in  $x''$ )
- intermediate (e.g., move *T2* on *M2* in  $x''$ )

In fact, the impact of a change to the solution depends

- both on the modified elements
- and on the unmodified elements (*contrary to Interlude 1*)

The objective function is “flat”: several solutions have the same value  
(*this is a problem when comparing different modifications*)

# Logic function problems: *Max-SAT* problem

Given a *CNF*, assign truth values to its logical variables so as to satisfy the maximum weight subset of its logical clauses

- a set  $V$  of **logical variables**  $x_j$  with values in  $\mathbb{B} = \{0, 1\}$  (*false, true*)
- a **literal**  $\ell_j$  is a **function** consisting of an affirmed or negated variable

$$\ell_j(x) \in \{x_j, \bar{x}_j\}$$

- a **logical clause** is a **disjunction** or logical sum (*OR*) of literals

$$C_i(x) = \ell_{i,1} \vee \dots \vee \ell_{i,n_i}$$

- a **conjunctive normal form** (*CNF*) is a **conjunction** or logical product (*AND*) of logical clauses

$$CNF(x) = C_1 \wedge \dots \wedge C_n$$

- to **satisfy a logical function** means to **make it assume value 1**
- a **function**  $w$  provides the weights of the *CNF* clauses

# Logic function problems: *Max-SAT* problem

The **ground set** is the set of all simple truth assignments

$$B = V \times \mathbb{B} = \{(x_1, 0), (x_1, 1), \dots, (x_n, 0), (x_n, 1)\}$$

The **feasible region** includes all subsets of simple assignments that are

- **complete**, that is include at least a literal for each variable
- **consistent**, that is include at most a literal for each variable

$$X = \{x \subseteq B : |x \cap B_v| = 1 \forall v \in V\}$$

with  $B_{x_j} = \{(x_j, 0), (x_j, 1)\}$

The **objective** is to maximise the total weight of the satisfied clauses

$$\max_{x \in X} f(x) = \sum_{i: C_i(x)=1} w_i$$

# Example

- Variables

$$V = \{x_1, x_2, x_3, x_4\}$$

- Literals

$$L = \{x_1, \bar{x}_1, x_2, \bar{x}_2, x_3, \bar{x}_3, x_4, \bar{x}_4\}$$

- Logical clauses

$$C_1 = \bar{x}_1 \vee x_2 \quad \dots \quad C_7 = x_2$$

- Conjunctive normal form

$$CNF = (\bar{x}_1 \vee x_2) \wedge (\bar{x}_1 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (\bar{x}_2 \vee x_4) \wedge (\bar{x}_2 \vee \bar{x}_4) \wedge x_1 \wedge x_2$$

- Weight function (uniform):

$$w_i = 1 \quad i = 1, \dots, 7$$

$x = \{(x_1, 0), (x_2, 0), (x_3, 1), (x_4, 1)\}$  satisfies  $f(x) = 5$  clauses out of 7

*Complementing a variable does not always change  $f(x)$  ( $x_1$  does,  $x_4$  not)*



# Numerical matrix problems: Set Covering (SCP)

Given

- a binary matrix  $A \in \mathbb{B}^{m,n}$  with row set  $R$  and column set  $C$
- column  $j \in C$  covers row  $i \in R$  when  $a_{ij} = 1$
- a function  $c : C \rightarrow \mathbb{N}$  provides the cost of each column

Select a subset of columns covering all rows at minimum cost

The ground set is the set of columns:  $B = C$

The feasible region includes all subsets of columns that cover all rows

$$X = \left\{ x \subseteq B : \sum_{j \in x} a_{ij} \geq 1 \forall i \in R \right\}$$

The objective is to minimise the total cost of the selected columns

$$\min_{x \in X} f(x) = \sum_{j \in x} c_j$$

# Example

$c$	4	6	10	14	5	6
-----	---	---	----	----	---	---

$A$	0	1	1	1	1	0
	0	0	1	1	0	0
	1	1	0	0	0	1
	0	0	0	1	1	1
	1	1	1	0	1	0

$A$	0	1	1	1	1	0	2
	0	0	1	1	0	0	1
	1	1	0	0	0	1	1
	0	0	0	1	1	1	1
	1	1	1	0	1	0	3

$$x' = \{c_1, c_3, c_5\} \in X$$

$$f(x') = 19$$

$A$	0	1	1	1	1	0	1
	0	0	1	1	0	0	0
	1	1	0	0	0	1	2
	0	0	0	1	1	1	2
	1	1	1	0	1	0	2

$$x'' = \{c_1, c_5, c_6\} \notin X$$

$$f(x'') = 15$$

“Set Covering”: covering a set (rows) with subsets (columns)

## Interlude 3: the feasibility test

Heuristic algorithms often require to solve the following problem

Given a subset  $x$ , is  $x$  feasible or not? In short,  $x \in X$ ?

*It is a decision problem*

The feasibility test requires to compute from the solution and test

- a single number: the total volume ( $KP$ ), the cardinality ( $MDP$ )
- a single set of numbers: values assigned to each variable ( $Max-SAT$ ), number of machines for each task ( $PMSP$ )
- several sets of numbers: number of containers for each object and total volume of each container ( $BPP$ )

The time required can be different if the test is performed

- from scratch on a generic subset  $x$
- on a subset  $x'$  obtained slightly modifying a feasible solution  $x$

Some modifications can be forbidden *a priori* to avoid infeasibility (insertions and removals for  $MDP$ ,  $PMSP$ ,  $Max-SAT$ ), while others require an *a posteriori* test (exchanges)

# Numerical matrix problems: Set Packing

Given

- a binary matrix  $A \in \mathbb{B}^{m,n}$  with row set  $R$  and column set  $C$
- columns  $j'$  e  $j'' \in C$  conflict with each other when  $a_{ij'} = a_{ij''} = 1$
- a function  $\phi : C \rightarrow \mathbb{N}$  provides the value of each column

Select a subset of nonconflicting columns of maximum value

The ground set is the set of columns:  $B = C$

The feasible region includes all subsets of nonconflicting columns

$$X = \left\{ x \subseteq B : \sum_{j \in x} a_{ij} \leq 1 \forall i \in R \right\}$$

The objective is to maximise the total value of the selected columns

$$\max_{x \in X} f(x) = \sum_{j \in x} \phi_j$$

# Example

$$\phi \quad \begin{array}{|c|c|c|c|c|c|} \hline 4 & 6 & 10 & 14 & 5 & 6 \\ \hline \end{array}$$

$$A \quad \begin{array}{|c|c|c|c|c|c|} \hline 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ \hline \end{array}$$

$$A \quad \begin{array}{|c|c|c|c|c|c|} \hline 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ \hline \end{array} \quad \begin{array}{l} 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{array}$$

$$x' = \{c_2, c_4\} \in X$$

$$f(x') = 20$$

$$A \quad \begin{array}{|c|c|c|c|c|c|} \hline 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ \hline \end{array} \quad \begin{array}{l} 1 \\ 0 \\ 2 \\ 2 \\ 1 \end{array}$$

$$x'' = \{c_1, c_5, c_6\} \notin X$$

$$f(x'') = 15$$

“Set Packing”: packing disjoint subsets (columns) of a set (rows)

# Numerical matrix problems: Set Partitioning (*SPP*)

Given

- a **binary matrix**  $A \in \mathbb{B}^{m,n}$  with a set of rows  $R$  and a set of columns  $C$
- a **function**  $c : C \rightarrow \mathbb{N}$  that provides the **cost of each column**

select a minimum cost subset of nonconflicting columns covering all rows

The **ground set** is the **set of columns**:  $B = C$

The **feasible region** includes all **subsets of columns that cover all rows and are not conflicting**

$$X = \left\{ x \subseteq B : \sum_{j \in x} a_{ij} = 1 \forall i \in R \right\}$$

The **objective** is to **minimise the total cost of the selected columns**

$$\min_{x \in X} f(x) = \sum_{j \in x} c_j$$

# Example

$$c \quad \begin{array}{|c|c|c|c|c|c|} \hline 4 & 6 & 10 & 14 & 5 & 6 \\ \hline \end{array}$$

$$A \quad \begin{array}{|c|c|c|c|c|c|} \hline 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ \hline \end{array}$$

$$A \quad \begin{array}{|c|c|c|c|c|c|} \hline 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ \hline \end{array} \quad \begin{array}{l} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{array}$$

$$x' = \{c_2, c_4, c_6\} \in X$$

$$f(x') = 26$$

$$A \quad \begin{array}{|c|c|c|c|c|c|} \hline 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ \hline \end{array} \quad \begin{array}{l} 1 \\ 0 \\ 2 \\ 1 \\ 1 \end{array}$$

$$x'' = \{c_1, c_5, c_6\} \notin X$$

$$f(x'') = 15$$

“Set Partitioning”: partition a set (rows) into subsets (columns)

# Graph problems: Travelling Salesman Problem (*TSP*)

Given

- a **directed graph**  $G = (N, A)$
- a **function**  $c : A \rightarrow \mathbb{N}$  that provides the **cost of each arc**

select a circuit visiting all the nodes of the graph at minimum cost

The **ground set** is the **arc set**:  $B = A$

The **feasible region** includes the **circuits that visit all nodes in the graph** (Hamiltonian circuits)

*How to determine whether a subset is a feasible solution?*

*And a modification of a feasible solution?*

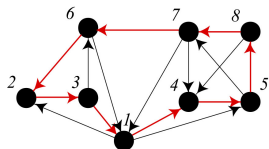
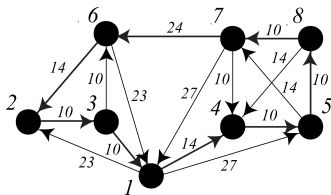
*Can we rule out some modifications?*

The **objective** is to **minimise the total cost of the selected arcs**

$$\min_{x \in X} f(x) = \sum_{j \in x} c_j$$

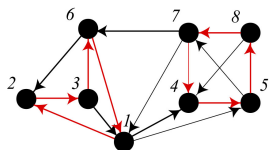


# Example



$$x' = \{(1, 4), (4, 5), (5, 8), (8, 7), (7, 6), (6, 2), (2, 3), (3, 1)\} \in X$$

$$f(x') = 102$$



$$x'' = \{(4, 5), (5, 8), (8, 7), (7, 4), (1, 2), (2, 3), (3, 6), (6, 1)\} \notin X$$

$$f(x'') = 106$$

## Interlude 4: the search for feasible solutions

Heuristic algorithms often require to solve the following problem

Find a feasible solution  $x \in X$

*It is a search problem*

The search for a feasible solution is trivial or easy for some problems:

- some sets are always feasible, such as  $x = \emptyset$  (*KP*, *Set Packing*) or  $x = B$  (feasible instances of *SCP*)
- random subsets satisfying a constraint, such as  $|x| = k$  (*MDP*)
- random subsets satisfying consistency constraints, such as assigning one task to each machine (*PMSP*), one value to each logic variable (*Max-SAT*), etc. . .

## Interlude 4: the search for feasible solutions

But **it is hard for other problems**:

- in the *BPP* the problem is **easy if the number of containers is large** (e. g., one container for each object)
- in the *SPP* **no polynomial algorithm is known** to solve the problem
- in the *TSP* the problem is **easy for complete graphs**

One can apply a **relaxation**, i. e. **enlarge the feasible region** from  $X$  to  $X'$

- the objective  $f$  must be extended from  $X$  to  $X'$  (see *Interlude 1*)
- but often  $X' \setminus X$  includes better solutions (... *how about that?*)

# Graph problems: Vertex Cover (VCP)

Given an **undirected graph**  $G = (V, E)$ , select a subset of vertices of minimum cardinality such that each edge of the graph is incident to it

The **ground set** is the **vertex set**:  $B = V$

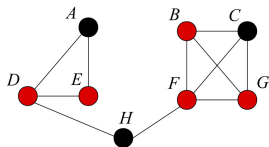
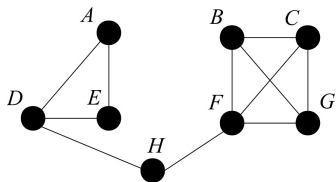
The **feasible region** includes all **vertex subsets** such that all the edges of the graph are incident to them

$$X = \left\{ x \subseteq V : x \cap (i, j) \neq \emptyset \forall (i, j) \in E \right\}$$

The **objective** is to **minimise the number of selected vertices**

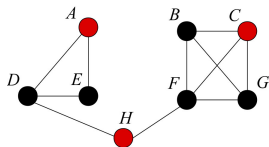
$$\min_{x \in X} f(x) = |x|$$

# Example



$$x' = \{B, D, E, F, G\} \in X$$

$$f(x') = 5$$



$$x'' = \{A, C, H\} \notin X$$

$$f(x'') = 3$$

# Graph problems: Maximum Clique Problem

Given

- an undirected graph  $G = (V, E)$
- a function  $w : V \rightarrow \mathbb{N}$  that provides the weight of each vertex

select the subset of pairwise adjacent vertices of maximum weight

The ground set is the vertex set:  $B = V$

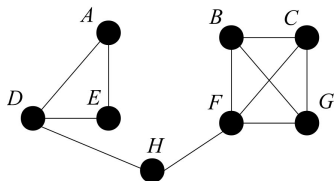
The feasible region includes all subsets of pairwise adjacent vertices

$$X = \{x \subseteq V : (i, j) \in E \forall i \in x, \forall j \in x \setminus \{i\}\}$$

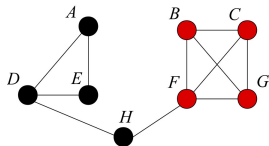
The objective is to maximise the weight of the selected vertices

$$\max_{x \in X} f(x) = \sum_{j \in x} w_j$$

# Example

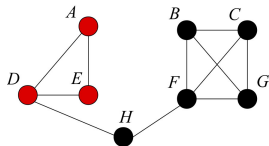


Uniform weights:  $w_i = 1$  for each  $i \in V$



$$x' = \{B, C, F, G\} \in X$$

$$f(x') = 4$$



$$x'' = \{A, D, E\} \in X$$

$$f(x'') = 3$$

# Graph problems: Maximum Independent Set Problem

Given

- an undirected graph  $G = (V, E)$
- a function  $w : V \rightarrow \mathbb{N}$  that provides the weight of each vertex

select the subset of pairwise nonadjacent vertices of maximum weight

The ground set is the vertex set:  $B = V$

The feasible region includes the subsets of pairwise nonadjacent vertices

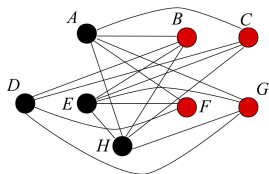
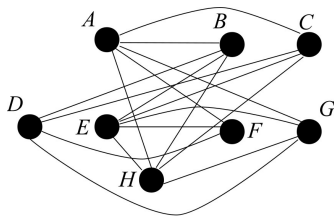
$$X = \{x \subseteq B : (i, j) \notin E \forall i \in x, \forall j \in x \setminus \{i\}\}$$

The objective is to maximise the weight of the selected vertices

$$\max_{x \in X} f(x) = \sum_{j \in x} w_j$$

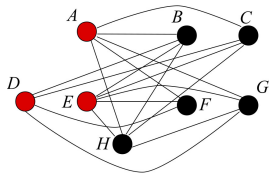


# Example



$$x' = \{B, C, F, G\} \in X$$

$$f(x') = 4$$



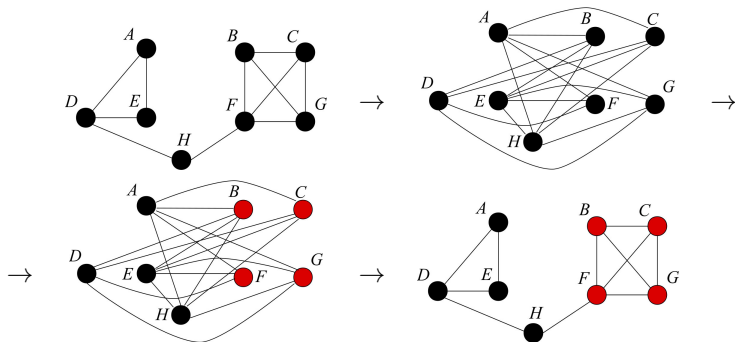
$$x'' = \{A, D, E\} \in X$$

$$f(x'') = 3$$

# Interlude 5: the relations between problems (1)

Each instance of the *MCP* is equivalent to an instance of the *MISP*

- 1 start from the *MCP* instance, that is graph  $G = (V, E)$
- 2 build the **complementary graph**  $\bar{G} = (V, (V \times V) \setminus E)$
- 3 find a solution of the *MISP* on  $\bar{G}$  (**optimal or heuristic**)
- 4 **the corresponding vertices give a solution of the *MCP* on  $G$**   
(optimal or heuristic, **according to the original one**)

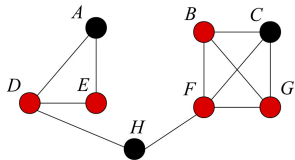


The process can be applied also in the opposite direction

## Interlude 5: the relations between problems (2)

The *VCP* and the *SCP* are also related, but in a different way; **each instance of the *VCP* can be transformed into an instance of the *SCP***:

- each edge  $i$  corresponds to a row of the covering matrix  $A$
- each vertex  $j$  corresponds to a column of  $A$
- if edge  $i$  touches vertex  $j$ , set  $a_{ij} = 1$ ; otherwise  $a_{ij} = 0$
- **an optimal solution of the *SCP* gives an optimal solution of the *VCP***  
(a heuristic *SCP* solution gives a heuristic *VCP* solution)



	A	B	C	D	E	F	G	H
(A, D)	1	0	0	1	0	0	0	0
(A, E)	1	0	0	0	1	0	0	0
(B, C)	0	1	1	0	0	0	0	0
(B, F)	0	1	0	0	0	1	0	0
(B, G)	0	1	0	0	0	0	1	0
(C, F)	0	0	1	0	0	1	0	0
(C, G)	0	0	1	0	0	0	1	0
(D, E)	0	0	0	1	1	0	0	0
(D, H)	0	0	0	1	0	0	0	1
(F, G)	0	0	0	0	0	1	1	0
(F, H)	0	0	0	0	0	1	0	1

*It is not simple to do the reverse*

# Interlude 5: the relations between problems (3)

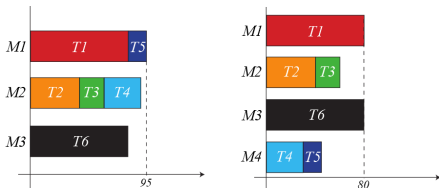
The *BPP* and the *PMSP* are equivalent, but in a more sophisticated way:

- the tasks correspond to the objects
- the machines correspond to the containers, but
  - *BPP*: minimise the number of containers, given the capacity
  - *PMSP*: given the number of machines, minimise the completion time

Start from a *BPP* instance

- 1 make an assumption on the optimal number of containers (e.g., 3)
- 2 build the corresponding *PMSP* instance
- 3 compute the optimal completion time (e.g., 95)
  - if it exceeds the capacity (e.g., 80), increase the assumption (4 or 5)
  - if it does not, decrease the assumption (2 or 1)

*(using heuristic PMSP solutions leads to a heuristic BPP solution)*



*The reverse process is possible*

The two problems are equivalent, but each one must be solved several times

# Kernelisation (“problem reduction”)

Kernelisation transforms all instances of  $P$  into simpler instances of  $P$ , instead of instances of another problem  $Q$

*This is also known as problem reduction*

Quite often, in fact, useful properties allow to prove that

- there exists an optimal solution not including certain elements of  $B$   
( $\Rightarrow$  such elements can be removed)
- there exists an optimal solution including certain elements of  $B$   
( $\Rightarrow$  such elements can be set apart and added later)

In short, remove elements of  $B$  without affecting the solution

Possible useful outcomes are

- an exact algorithm polynomial in  $n$  (parameterized complexity)
- faster exact and heuristic algorithms
- better heuristic solutions
- heuristic kernelisation: apply relaxed conditions sacrificing optimality

# Kernelisation of the *VCP*

Useful property: each vertex  $v$  of degree  $\delta_v \geq k + 1$  must belong to any feasible solution of value  $\leq k$

Otherwise,  $k + 1$  edges should be covered each by a different vertex

*Kernelisation* algorithm:

- start at step  $t = 0$  with  $k_0 = k$  and an empty vertex subset  $x_t := \emptyset$
- set  $t = t + 1$  and add to the solution the vertices of degree  $\geq k_t + 1$

$$\delta_v \geq k_t + 1 \Rightarrow x_t := x_{t-1} \cup \{v\}$$

- update  $k_t$ :  $k_t := k_0 - |x|$
- remove the vertices of zero degree, those of  $x$  and the covered edges

$$V := \{v \in V : \delta_v > 0\} \setminus x_t \quad E := \{e \in E : e \cap x_t = \emptyset\}$$

- if  $|E| > k_t^2$ , there is no feasible solution ( $k_t$  vertices are not enough)
- if  $|E| \leq k_t^2 \Rightarrow |V| \leq 2k_t^2$ ; apply the exhaustive algorithm

The complexity is  $T(n, k) \in \Theta(n + m + 2^{2k^2} k^2)$

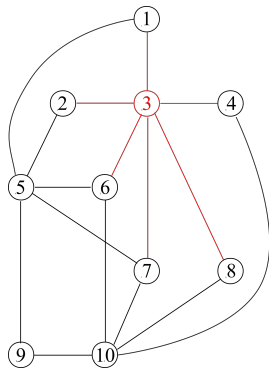
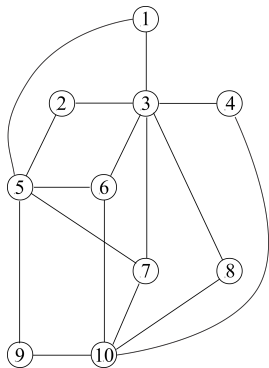
# Kernelisation of the *VCP*

Given the following graph, is there a solution with  $k \leq k_0 = 5$ ?

Exhaustive algorithm:  $T(n) \in \Theta(2^n(m+n))$

Since  $n = 10$  and  $m = 16$ ,  $T(n) \approx 2^{10}(10+16) = 26\,624$

$\delta_3 = 6 \geq k_0 + 1 \Rightarrow x_1 := \{3\}$ , remove the incident edges and  $k_1 = 4$



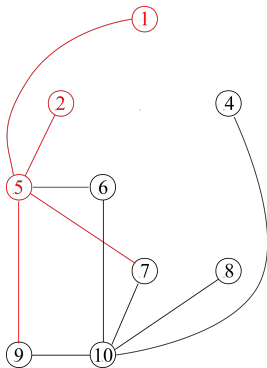
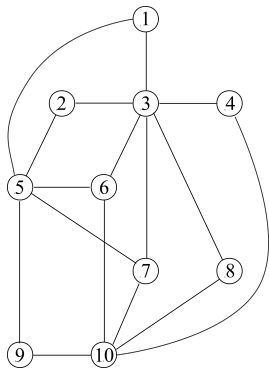
# Kernelisation of the *VCP*

Given the following graph, is there a solution with  $k \leq k_0 = 5$ ?

Exhaustive algorithm:  $T(n) \in \Theta(2^n(m+n))$

Since  $n = 10$  and  $m = 16$ ,  $T(n) \approx 2^{10}(10+16) = 26\,624$

$\delta_5 = 5 \geq k_1 + 1 \Rightarrow x_2 := \{3, 5\}$ , remove the incident edges and  $k_2 = 3$





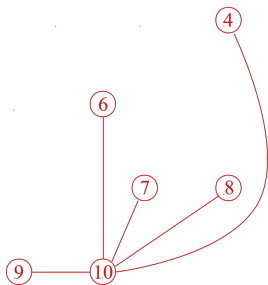
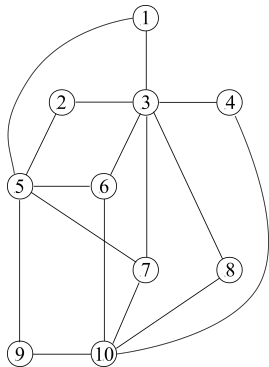
# Kernelisation of the *VCP*

Given the following graph, is there a solution with  $k \leq k_0 = 5$ ?

Exhaustive algorithm:  $T \in \Theta(2^n(m+n))$

Since  $n = 10$  and  $m = 16$ ,  $T \approx 2^{10}(10+16) = 26\,624$

$\delta_{10} = 5 \geq k_2 + 1 \Rightarrow x_3 := \{3, 5, 10\}$ , remove the incident edges and  
 $k_3 = 2$



Kernelisation:  $T \approx m + n = 10 + 16 = 26$

# Graph problems: Capacitated Min. Spanning Tree Problem

Given

- an undirected graph  $G = (V, E)$  with a root vertex  $r \in V$
- a function  $c : E \rightarrow \mathbb{N}$  that provides the cost of each edge
- a function  $w : V \rightarrow \mathbb{N}$  that provides the weight of each vertex
- a number  $W \in \mathbb{N}$  that is the subtree appended to the root (branch)

select a spanning tree of minimum cost such that each branch respects the capacity

The ground set is the edge set:  $B = E$

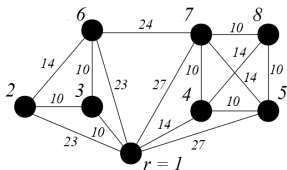
The feasible region includes all spanning trees such that the weight of the vertices spanned by each branch does not exceed  $W$

*The feasibility test requires to visit the subgraph*

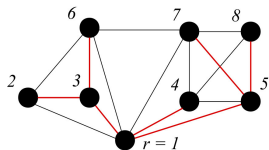
The objective is to minimise the total cost of the selected edges

$$\min_{x \in X} f(x) = \sum_{j \in x} c_j$$

# Example

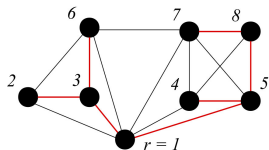


Uniform weight ( $w_i = 1$  for each  $i \in V$ ) and capacity:  $W = 3$



$$x' = \{(r, 3), (3, 2), (3, 6), (r, 4), (r, 5), (5, 7), (5, 8)\} \in X$$

$$f(x') = 95$$



$$x'' = \{(r, 3), (3, 2), (3, 6), (r, 5), (5, 4), (5, 8), (8, 7)\} \notin X$$

$$f(x'') = 87$$

*It is easy to evaluate the objective, less easy the feasibility*

# Cost of the main operations

The objective function is

- fast to evaluate: sum the edge costs
- fast to update: sum the added costs and subtract the removed ones

but it is easy to obtain subtrees that span vertices in a nonoptimal way

The feasibility test is

- not very fast to perform:
  - visit to check for connection and acyclicity
  - visit to compute the total weight of each subtree
- not very fast to update:
  - show that the removed edges break the loops introduced by the added ones
  - recompute the weights of the subtrees

This also holds when the graph is complete

*What if we described the problem in terms of vertex subsets?*

# An alternative description

Define a **set of branches**  $T$  *(as the containers in the BPP)*

*One for each vertex in  $V \setminus \{r\}$ : some can be empty*

The **ground set** is the **set of the (vertex,branch) pairs**:  $B = V \times T$

The **feasible region** includes all **partitions of the vertices into connected subsets** (*visit, trivial on complete graphs*) of **weight  $\leq W$**  (*as in the BPP*)

$$X = \left\{ x \subseteq B : |x \cap B_v| = 1 \forall v \in V \setminus \{r\}, \sum_{(i,j) \in B^t} w_i \leq W \forall t \in T, \dots \right\}$$

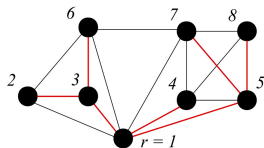
with  $B_v = \{(i,j) \in B : i = v\}$ ,  $B^t = \{(i,j) \in B : j = t\}$

The **objective** is to **minimise the sum of the costs of the branches spanning each subset of vertices and appending it to the root**

*It is a combination of minimum spanning tree problems*

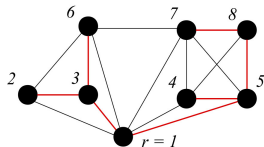
# Example

The previously considered solutions now have a different representation



$$x' = \{(2, T1), (3, T1), (6, T1), (4, T2), (5, T3), (7, T3), (8, T3)\} \in X$$

$$f(x') = 95$$



$$x'' = \{(2, T1), (3, T1), (6, T1), (4, T2), (5, T2), (7, T2), (8, T2)\} \notin X$$

$$f(x'') = 87$$

*The feasibility test only requires to sum the weights,  
computing the objective requires to solve a MST problem*

# Cost of the main operations

The objective function is

- slow to evaluate: compute a *MST* for each subset
- slow to update: recompute the *MST* for each modified subset

but the subtrees are optimal by construction

If the graph is complete, the feasibility test is

- fast to perform:
  - sum the weights of the vertices for each subtree
- fast to update:
  - sum the added weights and subtract the removed ones

*Advantages and disadvantages switched places*

# Graph problems: Vehicle Routing Problem (VRP)

Given

- a directed graph  $G = (N, A)$  with a depot node  $d \in N$
- a function  $c : A \rightarrow \mathbb{N}$  that provides the cost of each arc
- a function  $w : N \rightarrow \mathbb{N}$  that provides the weight of each node
- a number  $W \in \mathbb{N}$  that is the capacity of each circuit

select a set of circuits of minimum cost such that each one visits the depot and respects the capacity

The ground set could be

- the arc set:  $B = A$
- the set of all (node,circuit) pairs:  $B = N \times C$

The feasible region could include

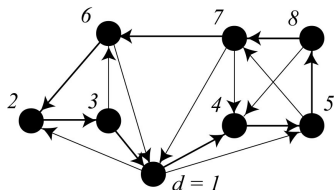
- all arc subsets that cover all nodes with circuits visiting the depot and whose weight does not exceed  $W$  (*again the visit of a graph*)
- all partitions of the nodes into subsets of weight non larger than  $W$  and admitting a spanning circuit (*NP-hard problem!*)

The objective is to minimise the total cost of the selected arcs

$$\min_{x \in X} f(x) = \sum_{j \in x} c_j$$

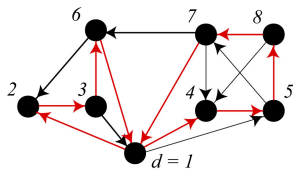


# Example



Uniform weight ( $w_i = 1$  for each  $i \in N$ ) and capacity:  $W = 4$

The solutions could be described as



- arc subsets

$$x = \{(d, 2), (2, 3), (3, 6), (6, d), (d, 4), (4, 5), (5, 8), (8, 7), (7, d)\} \in X$$

- node partitions

$$x = \{(2, C1), (3, C1), (6, C1), (4, C2), (5, C2), (7, C2), (8, C2)\} \in X$$

$$f(x) = 137$$

## Interlude 6: combining alternative representations

The *CMSTP* and the *VRP* share an interesting complication:  
different definitions of the ground set  $B$  are possible and natural

- the description as a set of edges/arcs  
looks preferable to manage the objective
- the description as a set of pairs (vertex,tree)/(node/circuit) looks  
better to generate optimal solutions and to deal with feasibility

Which description should be adopted?

- the one that makes easier the most frequent operations
- both, if they are used much more frequently than updated, so that  
the burden of keeping them up-to-date and consistent is acceptable

Answer all the fundamental questions on all the considered problems

① Objective function:

- a) What is the cost of computing  $f(x)$  given  $x$ ?
- b) Is  $f(x)$  additive, quadratic, etc...?
- a) What is the cost of computing  $f(x')$  given  $f(x)$  and a "small" transformation  $x \rightarrow x'$ ?
- c) Is  $f(x)$  "flat"?

② Feasibility:

- a) What is the cost of testing whether subset  $x$  is a feasible solution?
- b) What is the cost of testing whether subset  $x'$  is a feasible solution given a feasible solution  $x$  and a "small" transformation  $x \rightarrow x'$ ?
- c) Are some transformations intrinsically feasible (or unfeasible)?
- d) Is it easy to find a feasible solution?  
Is there a subset that is always feasible?

③ Relations between problems:

- a) Are there transformations from/to the problem to/from other ones?

④ Ground sets:

- a) Are there alternative definitions of the ground set?
- b) What are their relative advantages and disadvantages?