FSD301 - Introduction to computational complexity

Christophe Garion, Stéphanie Roussel, Pierre-Antoine Morin, and others...
ISAE-SUPAERO/DISC, ONERA

Credits

These slides have been written using previous lectures material from several authors. The following people should be accredited as co-authors of the present slides:

- Christophe Garion (ISAE/SUPAERO)
- Hélène Fargier (CNRS)
- Gérard Verfaillie (ONERA)
- Cédric Pralet (ONERA)
- Stéphanie Roussel (ONERA)

Many thanks to Pierre-Antoine Morin for his implication and the exercise on resource-constrained project scheduling.

License CC BY-NC-SA 3.0

Learning outcomes

At the end of this session you should

- have understood that some problems are (really) more difficult than others
- be able to show that a problem is in complexity class P, either directly or by reduction
- be able to show that a problem is in complexity class NP
- be able to show that a problem is in complexity class NPC by reduction
- know some classic problems: SAT, TSP, graph coloring etc.
- be able to explain the complexity class PSPACE

Outline

Outline

Outline

Connecting equipments

Input

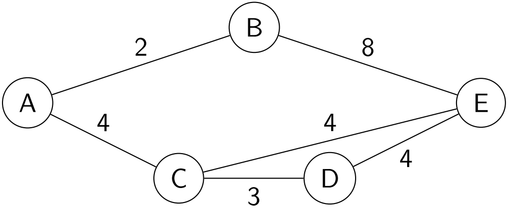
- some **equipments**
- possible **connections** between equipments
- **costs** associated to connections

Objective

Connect all equipments by **minimizing the total connections cost**.

Connecting equipments

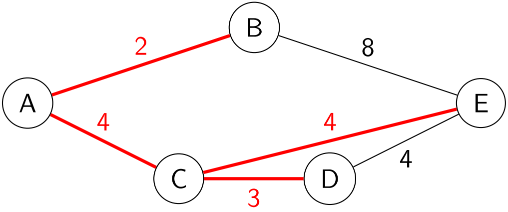
A possible model: **weighted graphs**.



Exercise

Is it easy or hard to compute?

A possible model: **weighted graphs**.



😄

Exercise

Is it easy or hard to compute?

Easy, this is a classical problem: computing the **Minimum Spanning Tree** of a graph.

Take-offs planning

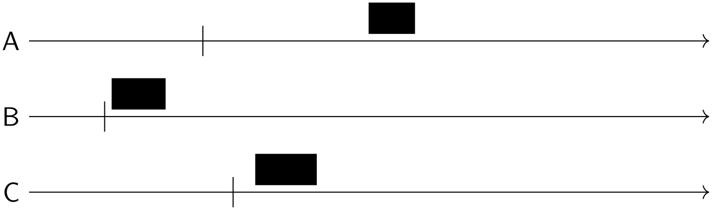
Input

- only one **runway**
- a set of **planes** that must take-off in a time horizon
- **minimal intervals** between take-offs (depending on the plane type)
- **earliest take-off times**

Objective

Plan take-offs in order to **minimize the maximal delay** on the set of planes (i.e. the difference between the expected take-off time and the real one).

Take-offs planning



Exercise

Is it easy or hard to compute?

Exercise

Is it easy or hard to compute?

Hard, **scheduling tasks** with length and earliest starting time with minimizing the **makespan** is a **difficult problem** (see the Travelling Salesman Problem).

Project management

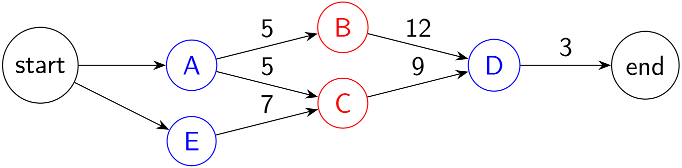
Input

- **tasks**
- tasks **durations**
- **precedence** constraints between tasks
- **resources**

Objective

Organize tasks in time in order to **minimize** the project total length.

A possible model: **weighted graphs** again!

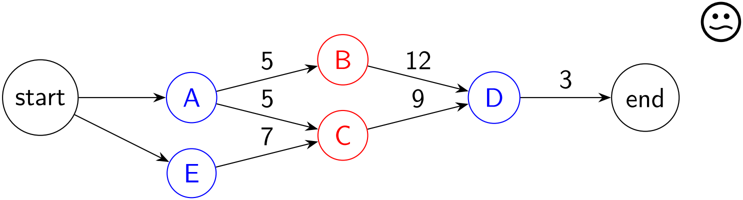


Exercise

Is it easy or hard to compute?

Project management

A possible model: **weighted graphs** again!



Exercise

Is it easy or hard to compute?

Easy, if you do not consider resources: longest path in a weighted acyclic graph.
Hard otherwise!

Outline

1. **Introduction to combinatorial optimization**
   - Some examples
   - Framework
2. **Complexity theory**
3. **The P complexity class**
4. **The NP and NPC classes**
5. **Beyond P and NP**

Some common features (and vocabulary)

- a **search space** $\mathcal{S}$: the possible alternatives
- the search space is represented by a set of **variables** $V = \{v_i \ : \ i \in \{1, \ldots, n\}\}$, each variable v_i having a domain D_i
  - an alternative is an instanciation of each v_i
  - the search space is thus $\prod_{i \in \{1, \ldots, n\}} D_i$
- a set Co of **constraints** to be satisfied
  - constraints are **assertions**
  - they may represent physical (real-world) limitations or user prerequisites
  - a **solution** is an alternative that satisfies all constraints in Co
- a set Cr of **criteria** to be satisfied as best
  - a criterion is a function from $\mathcal{S}$ to a totally ordered set ($\mathbb{R}^+$ for instance). This function has to be **minimized or maximized**
  - they represent **user preferences**

The knapsack problem

Input

- a set O of **objects** to put in the knapsack
- a set D of **dimensions** to be considered (weight, volume, etc.)
- for each dimension d, a maximal **capacity** Ca_d
- for each object o and each dimension d a **consumption** $Co_{o,d}$
- for each object o, its **value** V_o reflects its importance

Objective

Decide which objects to put in the knapsack in order to maximize the sum of values of the chosen objects while respecting the capacities.

Modelling the knapsack problem

Exercise

Model the knapsack problem!

Modelling the knapsack problem

Exercise

Model the knapsack problem!

- variables: $\forall o \in O \; p_o \in \{0, 1\}$
- constraints: $\forall d \in D \; \sum_{o \in O} p_o \times Co_{o,d} \leq Ca_d$
- criterion: maximize $\sum_{o \in O} p_o \times V_o$

Challenges

In the general case, there is **no analytical solution** to the problem.

Even if there is no analytical solution, **computational power** can be used to compute optimal or approached solutions.

Challenges

In the general case, there is **no analytical solution** to the problem.

Even if there is no analytical solution, **computational power** can be used to compute optimal or approached solutions.

But:

- even if it is finite, the search space can be **huge**
- alternatives **enumeration** (to verify them, compare them or find the best) can be impossible in practice
- sometimes, even **finding a solution** can be difficult!

Example: the Traveling Salesman Problem (TSP)

Input

- a set of towns to be visited by a salesman
- the distances between the towns

Objective

Find a **path** with **minimal length** starting from a town t, ending in v and visiting each town exactly one time (**hamiltonian path**).

Example: the Traveling Salesman Problem (TSP)

Input

- a set of towns to be visited by a salesman
- the distances between the towns

Objective

Find a **path** with **minimal length** starting from a town t, ending in v and visiting each town exactly one time (**hamiltonian path**).

If you have n towns, the size of the space search is $(n-2)!$.

Are TSP solution enumerable?

Optimistic hypothesis: 10^{-12} s to compute and evaluate a path.

# of towns	Computation time (s)
10	$3.63 \cdot 10^{-6}$
15	1.31
20	2432902
30	265252859812191058636

Some intuitions...

- 2432902 s $\approx$ 0.77 year
- 265252859812191058636 s $\approx$ 84111130077 millenia = more than **500 times the age of the Universe**!

➥ combinatorial explosion

How to deal with combinatorial explosion?

Challenge

Produce optimal or approached solution **without exploring the complete search space**.

- dedicated algorithms: efficient, but costly
- use/adapt existing algorithms and tools
- use a **generic modelling framework**, with **generic resolution algorithms**

Some modelling framework and resolution methods

	Continuous	Discrete
Linear	simplex Linear Programming (LP)	search tree + simplexe Integer Linear Programming (ILP)
Nonlinear	Nonlinear Programming (NLP) gradient descent	Constraint Programming (CP) tree search + constraints propagation local search

Graphs

Modelling framework : examples ✎

Exercise

What is the most appropriate modelling framework ?

1. The knapsack problem
   - variables: $\forall o \in O \;\; p_o \in \{0,1\}$
   - constraints: $\forall d \in D \;\; \sum_{o \in O} p_o \times Co_{o,d} \leq Ca_d$
   - criterion: maximize $\sum_{o \in O} p_o \times V_o$

Exercise

What is the most appropriate modelling framework ?

1. The knapsack problem
   - variables: $\forall o \in O \; p_o \in \{0, 1\}$
   - constraints: $\forall d \in D \; \sum_{o \in O} p_o \times Co_{o,d} \leq Ca_d$
   - criterion: maximize $\sum_{o \in O} p_o \times V_o$

$\rightarrow$ **Integer Linear Programming (ILP)**

Modelling framework : examples

2. Pigeon holes
   - description: consider n pigeons and m holes. Assign each pigeon to a hole such that there is not more than one pigeon per hole.

2. Pigeon holes
   - description: consider n pigeons and m holes. Assign each pigeon to a hole such that there is not more than one pigeon per hole.
   - first modelling:
     - variables: for $i \in [1, n], j \in [1, m],\ y_{i,j} \in [0, 1]$
     - constraints:

$$\forall i \in [1, n] \sum_{j=1}^{m} y_{i,j} = 1$$

$$\forall j \in [1, m] \sum_{i=1}^{n} y_{i,j} \leq 1$$

2. Pigeon holes
   - description: consider n pigeons and m holes. Assign each pigeon to a hole such that there is not more than one pigeon per hole.
   - first modelling:
     - variables: for $i \in [1, n], j \in [1, m],\ y_{i,j} \in [0, 1]$
     - constraints:
       $\forall i \in [1, n] \sum_{j=1}^{m} y_{i,j} = 1$
       $\forall j \in [1, m] \sum_{i=1}^{n} y_{i,j} \leq 1$

$\rightarrow$ **Integer Linear Programming (ILP)**

2. Pigeon holes
   - description: consider n pigeons and m holes. Assign each pigeon to a hole such that there is not more than one pigeon per hole.
   - first modelling:
     - variables: for $i \in [1, n], j \in [1, m],\ y_{i,j} \in [0, 1]$
     - constraints:
       $\forall i \in [1, n] \sum_{j=1}^{m} y_{i,j} = 1$
       $\forall j \in [1, m] \sum_{i=1}^{n} y_{i,j} \leq 1$

   $\rightarrow$ **Integer Linear Programming (ILP)**
   - second modelling:
     - variables: for $i \in [1, n],\ x_i \in [1, m]$
     - constraints: $\mathit{allDiff}\{x_i | i \in [1, m]\}$

2. Pigeon holes
   - description: consider n pigeons and m holes. Assign each pigeon to a hole such that there is not more than one pigeon per hole.
   - first modelling:
     - variables: for $i \in [1, n], j \in [1, m]$, $y_{i,j} \in [0, 1]$
     - constraints:
       $$\forall i \in [1, n] \sum_{j=1}^{m} y_{i,j} = 1$$
       $$\forall j \in [1, m] \sum_{i=1}^{n} y_{i,j} \leq 1$$

   $\rightarrow$ **Integer Linear Programming (ILP)**
   - second modelling:
     - variables: for $i \in [1, n]$, $x_i \in [1, m]$
     - constraints: $allDiff\{x_i | i \in [1, m]\}$

   $\rightarrow$ **Constraint Programming (CP)**

3. Transferring wheat
   - description: let A, B, C and D be 4 cities. There are 7 tons of wheat available in city A and 8 tons in city B. City C needs 6 tons of wheat and city D needs 4 tons. Given the following transferring costs for one ton, what is the minimum overall price to meet the demand of cities C and D ?

Origin	Destination	Cost
A	C	5 k€
A	D	10 k€
B	C	15 k€
B	D	4 k€

Modelling framework : examples

3. Transferring wheat
   - description: let A, B, C and D be 4 cities. There are 7 tons of wheat available in city A and 8 tons in city B. City C needs 6 tons of wheat and city D needs 4 tons. Given the following transferring costs for one ton, what is the minimum overall price to meet the demand of cities C and D ?

Origin	Destination	Cost
A	C	5 k€
A	D	10 k€
B	C	15 k€
B	D	4 k€

   - variables: $q_{A\to C}$, $q_{A\to D}$, $q_{B\to C}$, and $q_{B\to D} \in \mathbf{R}^+$
   - constraints:
     - $q_{A\to C} + q_{A\to D} \leq 7$
     - $q_{B\to C} + q_{B\to D} \leq 8$
     - $q_{A\to C} + q_{B\to C} \geq 6$
     - $q_{A\to D} + q_{B\to D} \geq 4$
   - criterion: minimize $5q_{A\to C} + 10q_{A\to D} + 15q_{B\to C} + 4q_{B\to D}$

$\rightarrow$ **Linear Programming (LP)**

Modelling framework : examples ✎

4. Transferring wheat again
   - description: same problem when considering that only whole tons can be transferred.

Modelling framework : examples

4. Transferring wheat again
   - description: same problem when considering that only whole tons can be transferred.
   - variables: $q_{A\to C}$, $q_{A\to D}$, $q_{B\to C}$, and $q_{B\to D} \in \mathbf{N}^+$
   - constraints:
     - $q_{A\to C} + q_{A\to D} \leq 7$
     - $q_{B\to C} + q_{B\to D} \leq 8$
     - $q_{A\to C} + q_{B\to C} \geq 6$
     - $q_{A\to D} + q_{B\to D} \geq 4$
   - criterion: minimize $5q_{A\to C} + 10q_{A\to D} + 15q_{B\to C} + 4q_{B\to D}$

$\rightarrow$ **Integer Linear Programming (ILP)**

Outline

Motivation

Observation from previous examples:

- some **algorithms** seem to be **more efficient** than others
- for a given problem, some **instances** seem to be **more difficult to solve** than others
- some **problems** seem to be more difficult than others

Motivation

Observation from previous examples:

- some **algorithms** seem to be **more efficient** than others
- for a given problem, some **instances** seem to be **more difficult to solve** than others
- some **problems** seem to be more difficult than others

Objective

- build a **theory** that **explains** and **predicts** such phenomena
- the theory must be **independent** from the **languages**, **compilers** or **machines** that are used

This is **complexity theory**.

Algorithms and problems

Do not mix algorithms, instances and problems!

Problem

A set of **instances** with the **same structure** on which you ask a question.

Instance

A problem with data.

Algorithm

A **procedure** taking an instance as **input** and answering the question of the problem as output.

An example: shortest path

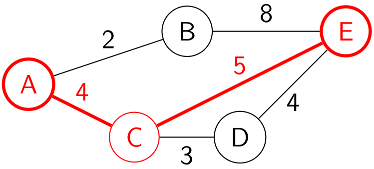
Exercise

Can you "instantiate" the previous notions on the shortest path problem?

Exercise

Can you "instantiate" the previous notions on the shortest path problem?

- **problem:** find the shortest path between 2 vertices in a graph
- **instance:** find the shortest path between A and E on the following graph



- **(several) algorithms:** paths enumeration, Dijkstra's algorithm, Bellman-Ford algorithm, Floyd-Warshall algorithm, Johnson's algorithm etc.

Another (important) example: SAT

The **SAT** (for **satisfiability**) problem is an important problem for complexity theory.

Definition (clause)

Let $\{x_1, \ldots, x_n\}$ be a set of boolean variables (i.e. $D_{x_i} = \{True, False\}$).

- **literal**: a variable x_i or the negation of a variable ($\neg x_i$)
- **clause**: a disjunction of literals, e.g. $x_1 \vee \neg x_4 \vee x_5$

- **problem:** given a **finite set of clauses**, is the set **satisfiable** (i.e. can you assign variables to make all the clauses true)?
- **instance:**
- **algorithms:**

Another (important) example: SAT

The **SAT** (for **satisfiability**) problem is an important problem for complexity theory.

Definition (clause)

Let $\{x_1, \ldots, x_n\}$ be a set of boolean variables (i.e. $D_{x_i} = \{\mathit{True}, \mathit{False}\}$).

- **literal**: a variable x_i or the negation of a variable ($\neg x_i$)
- **clause**: a disjunction of literals, e.g. $x_1 \vee \neg x_4 \vee x_5$

- **problem:** given a **finite set of clauses**, is the set **satisfiable** (i.e. can you assign variables to make all the clauses true)?
- **instance:** is $\{x_1 \vee \neg x_4 \vee x_5, \neg x_1 \vee \neg x_5\}$ satisfiable?
- **algorithms:** models enumeration, resolution algorithm, DPLL procedure etc.

Previously in TCS1-IN "Algorithms and programming"...

For a given algorithm $\mathcal{A}$:

- the **time complexity** of $\mathcal{A}$ is the time taken by $\mathcal{A}$ to answer
- the **space complexity** of $\mathcal{A}$ is the memory space taken by $\mathcal{A}$ to answer
- in practice, for time complexity:
  - the **worst-case** complexity is often used. The **maximal number of executed operations** for an input with **a given size** is computed
  - a **cost model** is chosen (the number of assignments, of array accesses etc.)
  - **asymptotic analysis** is used (mainly O notation)

Time complexity: some examples

Sorting an array with n elements

- selection sort:
- mergesort:

Shortest path in a graph with n vertices and m edges

- path enumerations:
- Dijkstra's algorithm (with simple implementation):

SAT with n variables with models enumeration

- satisfiable instance: sometimes immediate, sometimes 2^n
- unsatisfiable instance: always 2^n

➥ time complexity:

Time complexity: some examples

Sorting an array with n elements

- selection sort: $O(n^2)$
- mergesort: $O(n \cdot \log n)$

Shortest path in a graph with n vertices and m edges

- path enumerations: $O(n!)$
- Dijkstra's algorithm (with simple implementation): $O(m + n^2)$

SAT with n variables with models enumeration

- satisfiable instance: sometimes immediate, sometimes 2^n
- unsatisfiable instance: always 2^n

➥ time complexity: $O(2^n)$

Time complexity: some numbers

Complexity	Size of the biggest problem solvable with 10^6 instructions	Size of the biggest problem solvable with 10^{12} instructions
n	1 000 000	1 000 000 000 000
$n \cdot \log n$	64 000	32 000 000 000
n^2	1 000	1 000 000
n^3	100	10 000
2^n	20	40

Time complexity of a problem

Definition

The **time complexity** of a **problem** is the time complexity of the **best algorithm** solving the problem.

For instance, the time complexity of the array sorting problem is $O(n \cdot \log n)$ with n being the size of the array.

Time complexity of a problem

Definition

The **time complexity** of a **problem** is the time complexity of the **best algorithm** solving the problem.

For instance, the time complexity of the array sorting problem is $O(n \cdot \log n)$ with n being the size of the array.

- **polynomial problem**: easy, **tractable** problem for which there is a polynomial algorithm (**beware of degree and constants of the polynomial!**)
- **exponential problem**: hard, **intractable** problem

Some polynomial problems

- **sorting** problem
- **shortest path** in a weighted graph
- computing the **inverse** of an inversible matrix
- **linear programming**
- **primality** of an integer
- **minimum spanning tree** in a weighted graph
- **maximum flow** in a weighted graph
- **assignment problem**: maximum matching in a bipartite graph
- ...

Some exponential problems

Some problems for which the best known algorithm is exponential:

- **SAT**
- **Hamiltonian path**
- **Traveling Salesman** problem
- **graph coloring**
- **knapsack** problem
- **integer linear programming**
- ...

Some exponential problems

Some problems for which the best known algorithm is exponential:

- **SAT**
- **Hamiltonian path**
- **Traveling Salesman** problem
- **graph coloring**
- **knapsack** problem
- **integer linear programming**
- ...

But are these problems **really exponential**?

➥ **partial** response: **computational complexity theory** (do not confuse with algorithm complexity!)

➥ **idea**: **complexity classes** for problems

Which problems?

We will focus on **decision problems**, i.e. problems for which the answer is either **yes** or **no**.

For instance:

- existence of an hamiltonian path in a graph
- SAT

Which problems?

We will focus on **decision problems**, i.e. problems for which the answer is either **yes** or **no**.

For instance:

- existence of an hamiltonian path in a graph
- SAT

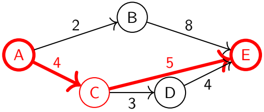
And **optimization problems**, i.e. problems for which you want to optimize (minimize) a criterion?

➥ you can always associate a decision problem to the optimization problem: is there a solution for which the **criteria is less than a value** k?

➥ **if the criterion are finite**, you can solve the optimization problem by **solving a finite number** of associated decision problems

Optimization and decision: example

- **problem:** shortest path between two vertices of a graph
- **associated decision problem:** is there a path with length less than k between the two vertices ($k \leq \sum_{e \in \text{edges}} d_e$)?



- **use the decision problem to solve the optimization one**:
  - let $k = 27$. Is there a path with length strictly less than 27? **Yes** (ABE)
  - let $k = 26\ldots$
  - $\ldots$
  - let $k = 9$. Is there a path with length strictly less than 9? **No:** the shortest path has a length of 9.

Outline

The P class

Definition

The P class (**Polynomial class**) is the class of decision problems that are **polynomial**.

Remember: this means that **there is at least one algorithm with polynomial-time complexity that solves the problem**.

Definition

The P class (**Polynomial class**) is the class of decision problems that are **polynomial**.

Remember: this means that **there is at least one algorithm with polynomial-time complexity that solves the problem**.

How to show that a problem is in P?

➥ an obvious approach: find a polynomial algorithm...

Exercise

Show that the following problem is in P: considering a graph with n vertices and m edges, is there a path between two vertices?

Path existence

Exercise

Show that the following problem is in P: considering a graph with n vertices and m edges, is there a path between two vertices?

Easy: you can use

- depth-first search
- breadth-first search
- even Dijkstra's algorithm!

A more complicated example

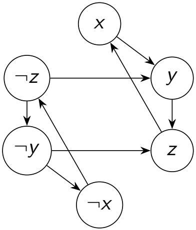
Theorem

*The **2-SAT** problem (the SAT problem in which all clauses have at most 2 literals) is in P.*

A more complicated example

Finding a polynomial algorithm is not so easy... You may represent the problem with a graph.

$\{\neg x \vee y, \neg y \vee z, x \vee \neg z, z \vee y\}$



- if there is a x s.t. there are a path from x to $\neg x$ and a path from $\neg x$ to x, φ is **UNSAT**
- otherwise, φ is **SAT**

You can of course reuse the results on the previous problem about vertex reachability!

Finding a polynomial algorithm for a problem may be difficult. There is also another approach: **polynomial reduction**.

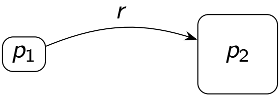
Definition

A **polynomial reduction** r of a problem p_1 to a problem p_2 is an **algorithm** with polynomial-time complexity that associates to each instance i_1 of p_1 an instance i_2 of p_2 such that the answers to i_1 and i_2 are **identical**.
If there is a polynomial reduction from p_1 to p_2, then $p_1 \leq_P p_2$ (p_2 is at least as difficult as p_1).

Theorem

Let p_1 and p_2 be problem s.t. $p_1 \leq_P p_2$. If $p_2 \in P$, then $p_1 \in P$.

Polynomial reduction



Remember:

- p_2 is **at least as difficult as** p_1
- therefore, if p_2 is "easy to solve", then is also p_1

Exercise

Let consider the 2-coloring problem: is it possible to color the vertices of a graph with only 2 colors s.t. two adjacent vertices do not have the same color?

Polynomial reduction: example

Exercise

Let consider the 2-coloring problem: is it possible to color the vertices of a graph with only 2 colors s.t. two adjacent vertices do not have the same color?

We can exhibit a polynomial reduction from 2-coloring to **2-SAT**:

- transform 2-coloring into 2-SAT:
  - for each vertex v_i of the graph, create one boolean variable x_i
  - for each edge (v_i, v_j) of the graph, add two clauses $x_i \vee x_j$ and $\neg x_i \vee \neg x_j$
- how to go from a solution of 2-SAT to a solution of 2-coloring:
  - if x_i is true, then v_i has the first color
  - if x_i is false, then v_i has the second color
- we can show that answering "is the obtained 2-SAT problem satisfiable?" is equivalent to answering "is the graph 2-colorable?"
- this reduction is **polynomial**, therefore **2-coloring** $\leq_P$ **2-SAT**

Therefore **2-coloring** $\in$ **P**.

Outline

Quick poll

What does “NP” stand for?

Quick poll

What does "NP" stand for?

How many among you think that NP means "Non Polynomial"?
Be honest ☺.

The NP class: Turing again!

Definition

A problem p is in the NP class if it can be solved by a **nondeterministic Turing machine** in **polynomial time** (hence the name NP).

Hu? We have not talked about Turing machines!

➥ they are in fact foundations for complexity theory

Definition (more intuitive)

A problem p is in the NP class if **verifying** if an alternative is a solution can be done in **polynomial time** (existence of a **polynomial verifier**).

If p is a problem in NP, there is a verifier V, such that for every instance i_p of p:

- if the answer to i_p is YES, there is a solution (also called a **certificate** or a **witness**) w such that V returns YES in polynomial time for the input (i_p, w)
  - ➥ V can verify in polynomial time that w is a correct solution to the problem
- if the answer to i_p is NO, then for all possible certificates w, V returns NO in polynomial time for the input $(i_p, w$

P and NP: intuition

An intuitive view:

- P: the class of problems for which **finding a solution is easy**
- NP: the class of problems for which **verifying a solution is easy**

An intuitive view:

- P: the class of problems for which **finding a solution is easy**
- NP: the class of problems for which **verifying a solution is easy**

Exercise

Is P $\subseteq$ NP?

P and NP: intuition

An intuitive view:

- P: the class of problems for which **finding a solution is easy**
- NP: the class of problems for which **verifying a solution is easy**

Exercise

Is P $\subseteq$ NP?

Yes, because if a problem p is in P, there is a polynomial algorithm $\mathcal{A}$ that solves p.
The verifier for p uses simply $\mathcal{A}$ (ignore the certificate and answer the question on p).

Some problems in NP

- **SAT**: the verifier is provided by boolean computation
- **Traveling Salesman Problem**: is there a cycle going through all vertices of a weighted graph with a total cost less than k?
  Verifier:
- **k-coloring**: same as 2-coloring but with k colors
  Verifier:

Some problems in NP

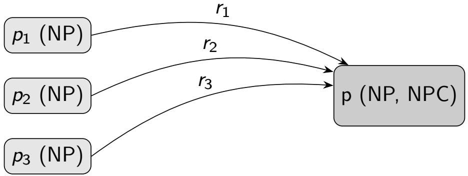
- **SAT**: the verifier is provided by boolean computation
- **Traveling Salesman Problem**: is there a cycle going through all vertices of a weighted graph with a total cost less than k?
  Verifier: simply verify first that all towns are in the certificate and then compute the total cost of the certificate
- **k-coloring**: same as 2-coloring but with k colors
  Verifier: simply use depth-first search for instance to verify for each vertex of the certificate if one of its neighbors has the same color

Are there in NP problems that are more difficult than the others?

Definition

A problem p is NP-complete iff:

- $p \in \mathrm{NP}$
- for **every** problem p' in NP, there is a polynomial reduction from p' to p.



p is at least as difficult as any problem in NP.

How to prove that your problem is in NPC?

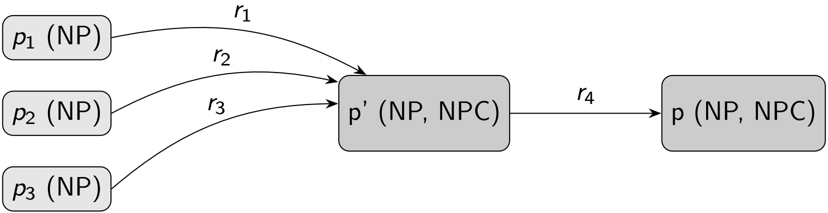
The hard way: direct proof of the hypothesis to verify to be in NPC.

Theorem (Cook, 1971)

SAT is NP-complete.

How to prove that your problem is in NPC (easier)?

To prove that p is in NPC, use a reduction!



It is sufficient to prove that:

- $p \in$ NP
- there is a problem p' in NPC s.t. $p' \leq_P p$

Beware: you have to reduce p' to p, not p to p'!

Reductions: examples

3-SAT

SAT with clauses with at most 3 literals.

- 3-SAT $\in$ NP
- reduction from SAT to 3-SAT:
  - replace each clause $C = x_1 \vee x_2 \vee \ldots \vee x_k$ $(k > 3)$ with

    $$C' = (x_1 \vee x_2 \vee y_1) \wedge (\neg y_1 \vee x_3 \vee y_2) \wedge \ldots \wedge (\neg y_{k-3} \vee x_{k-1} \vee x_k)$$

  - show that C and C' are equivalent
- 3-SAT $\in$ NPC

$$\boxed{\text{SAT (NPC)}} \xrightarrow{r_1} \boxed{\text{3-SAT (NPC)}}$$

Reductions: examples

k-coloring

- k-coloring $\in$ NP (easy to verify the given coloring)
- reduction from 3-SAT to k-coloring
- k-coloring $\in$ NPC

$$\boxed{\text{SAT (NPC)}} \xrightarrow{r_1} \boxed{\text{3-SAT (NPC)}} \xrightarrow{r_2} \boxed{\text{k-coloring (NPC)}}$$

A night at the museum

Exercise

Suppose that you are a museum administrator and you want to find the shortest path going through all the museum.
What is the associated decision problem? What is its complexity class?

Exercise

Suppose that you are a museum administrator and you want to find the shortest path going through all the museum.
What is the associated decision problem? What is its complexity class?

The associated decision problem is the following: "given a graph G and an integer k, is there a complete path (going through all vertices, maybe several times) with length less than or equal to k".
The problem is in **NPC**:

- you can verify that an alternative is a solution in polynomial time
- you know that the decision problem for Hamiltonian path is NP-complete. But finding an Hamiltonian path in a graph with n vertices is equivalent to finding a complete path of length $\leq n$. Therefore, there is a polynomial reduction from Hamiltonian path to our problem.

Input: project data

- $\mathcal{A}$: set of **activities**; p_i = processing time of activity i
- $\mathcal{R}$: set of renewable **resources**; b_k = capacity of resource k
- $a_{i,k}$: amount of resource k required by activity i
- $G = (\mathcal{A}, E)$: precedence graph (acyclic); there is an edge from i to j iff activity i must be finished before activity j starts

Alternative - schedule

- S_i: start date of activity i
- $C_{\max}$: makespan (greatest completion date) $= \max_{i \in \mathcal{A}} (S_i + p_i)$

Resource-Constrained Project Scheduling Problem (RCPSP)

Constraints

- Precedence constraints
- Resource constraints

Objective

- Optimization variant: minimize C_{max}
- Decision variant: does a schedule such that $C_{\mathrm{max}} \leq h$ exist?

RCPSP – Modelling (abstract) ✎

Exercise

Give a formulation of precedence constraints and resource constraints

Exercise

Give a formulation of precedence constraints and resource constraints

1. Precedence constraints

$$\forall (i_1, i_2) \in E \qquad S_{i_1} + p_{i_1} \leq S_{i_2}$$

2. Resource constraints

$$\forall k \in \mathcal{R}, \forall t \in \mathbb{N} \qquad \sum_{i \in \mathcal{A}_t(S)} a_{i,k} \leq b_k$$

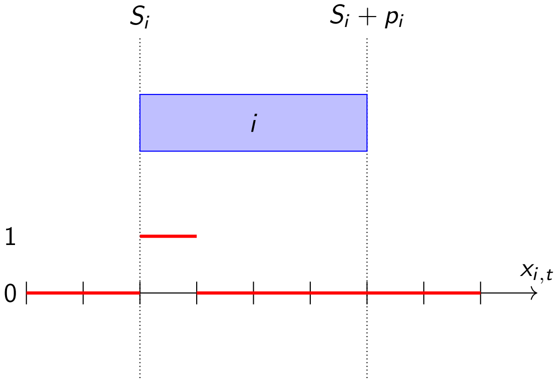
$$\mathcal{A}_t(S) = \left\{ i \in \mathcal{A} \;\middle|\; S_i \leq t < S_i + p_i \right\}$$

Exercise

Give a mixed integer linear programming model for the RCPSP

Exercise

Give a mixed integer linear programming model for the RCPSP



Minimize C_{max} such that:

- $\forall i \in \mathcal{A}, \;\; C_{\max} \geq S_i + p_i$
- $\forall i \in \mathcal{A}, \;\; \sum_{t=0}^{h-1} x_{i,t} = 1$
- $\forall i \in \mathcal{A}, \;\; S_i = \sum_{t=0}^{h-1} t\, x_{i,t}$
- $\forall (i_1, i_2) \in E, \;\; S_{i_1} + p_{i_1} \leq S_{i_2}$
- $\forall k \in \mathcal{R}, \;\; \forall t \in [0, h-1], \;\; \sum_{i \in \mathcal{A}} \sum_{u=\max(0,t-p_i+1)}^{t} a_{i,k}\, x_{i,u} \leq b_k$
- $\forall i \in \mathcal{A}, \;\; \forall t \in [0, h-1], \;\; x_{i,t} \in \{0, 1\}$

Exercise

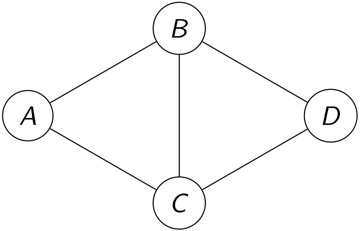
What is the complexity class of the RCPSP?

Exercise

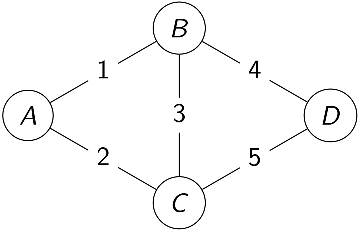
What is the complexity class of the RCPSP?

- RCPSP $\in$ **NP**
  - Test makespan: $\mathcal{O}(1)$
  - Test precedence constraints: $\mathcal{O}(|E|)$
  - Test resource constraints: $\mathcal{O}(|\mathcal{R}||\mathcal{A}|^2)$ ⚠
- h-coloring $\leq_{\mathbf{P}}$ RCPSP
  - Vertex $v \longrightarrow$ Activity i_v such that $p_{i_v} = 1$
  - Edge $e = (v_1^e, v_2^e) \longrightarrow$ Resource k_e such that $b_{k_e} = 1$
  - $q_{i_v, k_e} = 1$ if $v \in \{v_1^e, v_2^e\}$, 0 otherwise
  - $E = \emptyset$ (no precedence relations)
  - The graph admits a h-coloring **iff** there exists a schedule such that $C_{\max} \leq h$.
- RCPSP $\in$ **NPC**

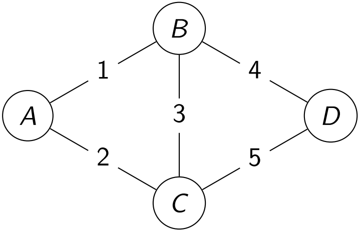
RCPSP – Reduction

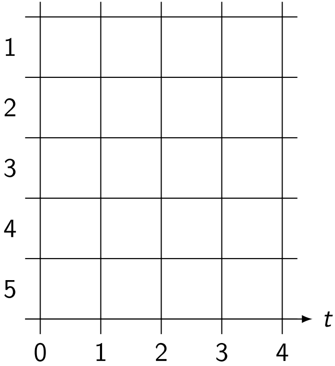


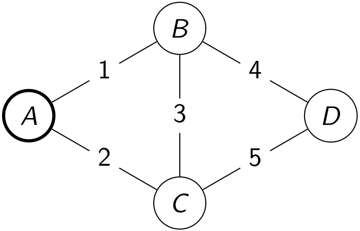
RCPSP – Reduction

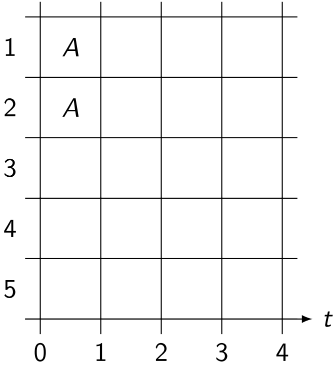


RCPSP – Reduction

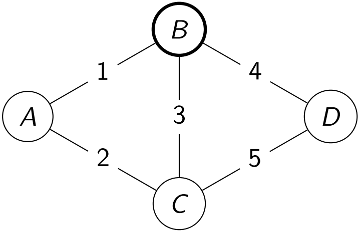


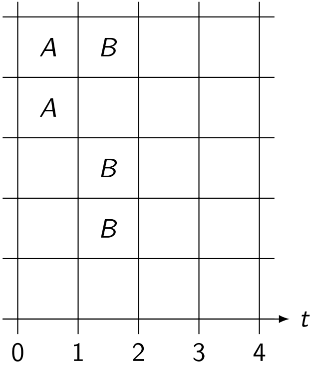



B
1
4
A
3
D
2
5
C

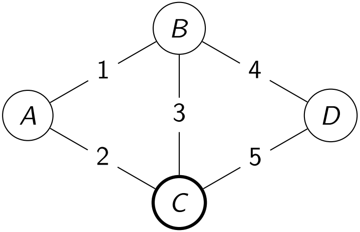

1
A
2
A
3
4
5
t
0
1
2
3
4

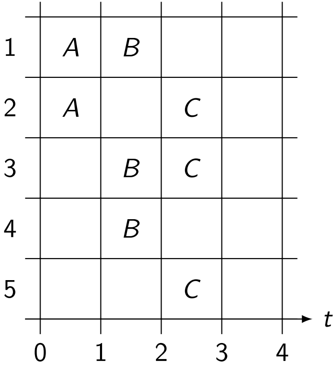
RCPSP – Reduction

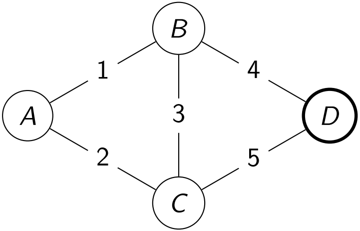


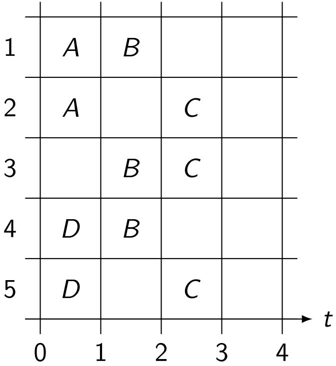


RCPSP – Reduction

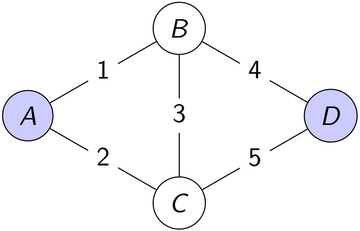


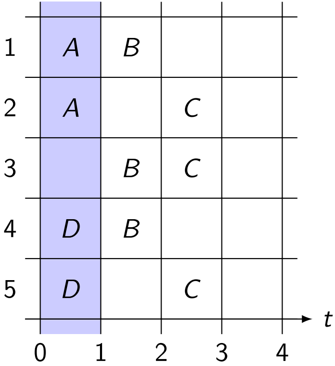


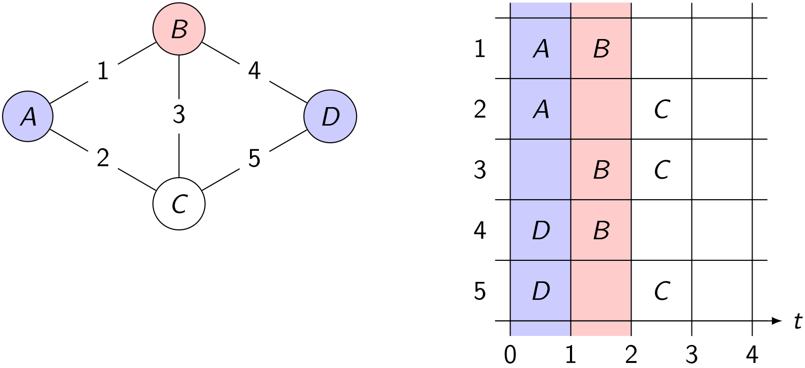

B
1
4
A
3
D
2
5
C


1 A B
2 A C
3 B C
4 D B
5 D C
t
0 1 2 3 4

RCPSP – Reduction

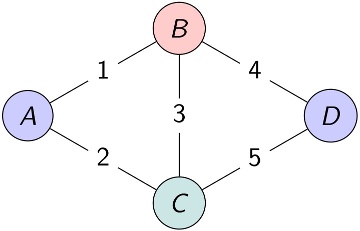


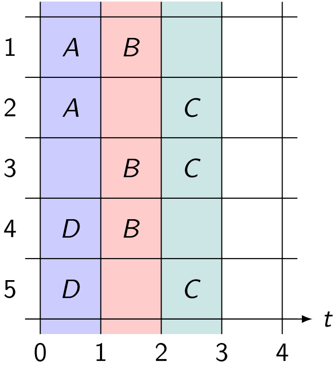




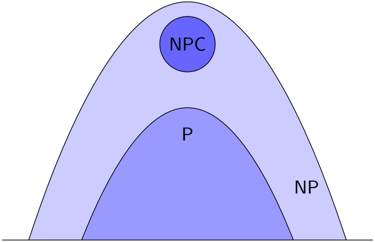
A
B
C
D
1
2
3
4
5
1
2
3
4
5
A
B
A
C
B
C
D
B
D
C
t
0
1
2
3
4

RCPSP – Reduction





Complexity hierarchy:



- for some problems in NP, we do not know if they are in P
- if **one** problem in P is also in NPC, then $\mathbf{P} = \mathbf{NP}$
- there is a \$1 000 000 prize for this question...
- **conjecture**: $\mathsf{P} \neq \mathsf{NP}$

Outline

PSPACE - Definition

A problem is in PSPACE iff it can be solved using a quantity of memory polynomial in the size of the problem.

PSPACE and PSPACE-completeness ✎

PSPACE - Definition

A problem is in PSPACE iff it can be solved using a quantity of memory polynomial in the size of the problem.

PSPACE-completeness - Definition

A problem p is PSPACE-complete iff:

- $p \in$ PSPACE
- for every problem p' in PSPACE, there is a polynomial reduction from p' to p

PSPACE - Definition

A problem is in PSPACE iff it can be solved using a quantity of memory polynomial in the size of the problem.

PSPACE-completeness - Definition

A problem p is PSPACE-complete iff:

- $p \in$ PSPACE
- for every problem p' in PSPACE, there is a polynomial reduction from p' to p

Exercise

Show that P $\subseteq$ PSPACE

PSPACE - Definition

A problem is in PSPACE iff it can be solved using a quantity of memory polynomial in the size of the problem.

PSPACE-completeness - Definition

A problem p is PSPACE-complete iff:

- $p \in$ PSPACE
- for every problem p' in PSPACE, there is a polynomial reduction from p' to p

Exercise

Show that P $\subseteq$ PSPACE

A time polynomial algorithm may require at most a polynomial quantity of memory.

3-SAT $\in$ PSPACE

Example of algorithm:

- enumerate all assignments with a binary counter (requires n bits)
- check if each assignment satisfies the CNF

3-SAT $\in$ PSPACE

Example of algorithm:

- enumerate all assignments with a binary counter (requires n bits)
- check if each assignment satisfies the CNF

Exercise

Show that NP $\subseteq$ PSPACE

3-SAT $\in$ PSPACE

Example of algorithm:

- enumerate all assignments with a binary counter (requires n bits)
- check if each assignment satisfies the CNF

Exercise

Show that NP $\subseteq$ PSPACE

Let p be a problem of NP

- $p \leq_P$ **3-SAT**
- there exists a polynomial reduction from p to **3-SAT** (polynomial quantity of memory)
- **3-SAT** can be solved using a polynomial quantity of memory
- p can be solved using a polynomial quantity of memory
- $p \in$ PSPACE

Example of a PSPACE-complete problem

QBF-SAT (Quantified Boolean Formula - SAT)

Let $\phi(x_1, \dots x_n)$ be a CNF. Is the following formula satisfiable ?
$\exists x_1 \forall x_2 \exists x_3 \dots \forall x_{n-1} \exists x_n \ \ \phi(x_1, \dots x_n)$

- Intuition (game): Alice picks a value for x_1, then Bernard picks a value for x_2, then Alice again for x_3, etc. Can Alice satisfy ϕ whatever the choices of Bernard ?
- **QBF-SAT** $\in$ PSPACE
  - test all assignments recursively
  - only one bit is stored at each step

  $\rightarrow$ memory proportional to the tree depth
- **QBF-SAT** is PSPACE-complete

Beyond P, NP and PSPACE...

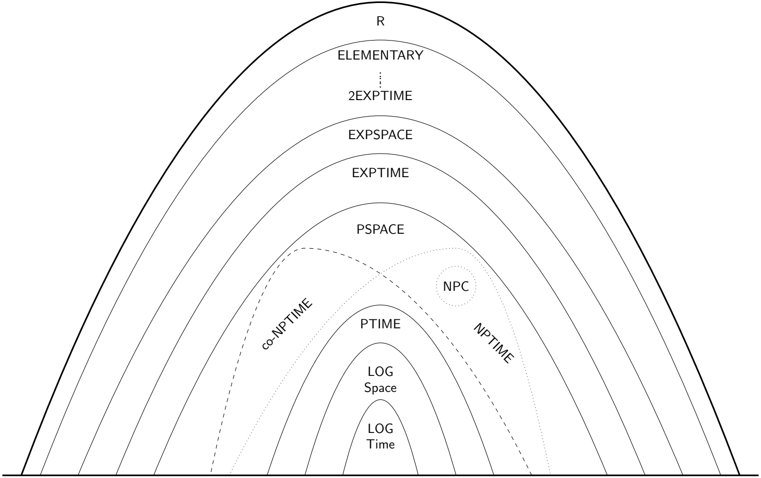


Image by S. Sardina based on Papadimitriou 1994.

What you should remember:

- algorithms complexity $\neq$ problems complexity
- if your problem is in P:
  - it means that your problem can be **solved in polynomial time**
  - it is a rather **good news**, but **beware to coefficients and degree**
- if your problem is in NPC, it is **hard** to solve, do not expect a polynomial algorithm, but:
  - the complexity is a **worst case** complexity
  - the **size** of the instances you deal with may be **small**
  - a **subproblem** of your problem may be in P
  - you may use **incomplete algorithms** to solve your problem
- if your problem is in NP, but not in P nor NPC, this is an uncertain situation...

Papadimitriou, C.H. (1994).
Computational Complexity.
Theoretical computer science.
Addison-Wesley.
ISBN: 9780201530827.

Perifel, Sylvain (2014).
Complexité algorithmique.
https://www.irif.fr/~sperifel/livre_complexite.html.

Garey, Michael R. and David S. Johnson (1990).
Computers and Intractability; A Guide to the Theory of NP-Completeness.
New York, NY, USA: W. H. Freeman & Co.
ISBN: 0716710455.

Aaronson, Scott et al. (2016).
The Complexity Zoo.
https://complexityzoo.uwaterloo.ca/Complexity_Zoo.