Computational Complexity

Carlos Cotta

Departamento de Lenguajes y Ciencias de la Computación
Universidad de Málaga

http://www.lcc.uma.es/∼ccottap

Comput Eng, Softw Eng, Comput Sci & Math – 2023-2024

UNIVERSIDAD
DE MÁLAGA

Computational Complexity

1. Introduction
   - Complexity Classes
   - Reductions
2. Some important Complexity Classes
   - Polynomial Space and Exponential Time
   - P and NP
   - NP-Complete Problems
   - Optimization and Complexity

Recap

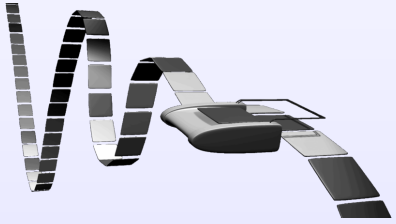
In the previous unit we presented the notion of computational problems and provided a taxonomy of these on the basis of what is being sought.

We also saw the notion of asymptotic complexity of an algorithm.

This latter notion can be used in turn to classify problems according to the complexity of the algorithms available for solving them.

Setting

Let us consider a computation model (this model can be Turing Machines or any other suitable formalism) and let us focus on decision problems (i.e., problems for which the answer is either `yes` or `no`).



Complexity Classes

We have to define a resource the TM consumes. As we saw, the most common choices are time and space.

Given $x \in I_P$, let $|x|$ be its size.

Complexity Class

Let REC be a resource the algorithm can consume, and let $f : \mathbb{N} \longrightarrow \mathbb{N}$ be a monotonically increasing function. The complexity class **REC**(f) is composed of all problems P such that for all $x \in I_P$ there exists an algorithm that solves x using at most $O(f(|x|))$ units of resource REC.

Examples of Complexity Classes

We typically use complexity classes based on space or on time:

- **TIME**(f): problems solvable in time bounded by f from above.
- **SPACE**(f): problems solvable using storage space bounded by f from above.

When we use a functional expression parameterized by k, we will be referring to all possible values of k, e.g., **TIME**(n^k) is the class of problems solvable in polynomial time (the degree of the polynomial being finite but unspecified).

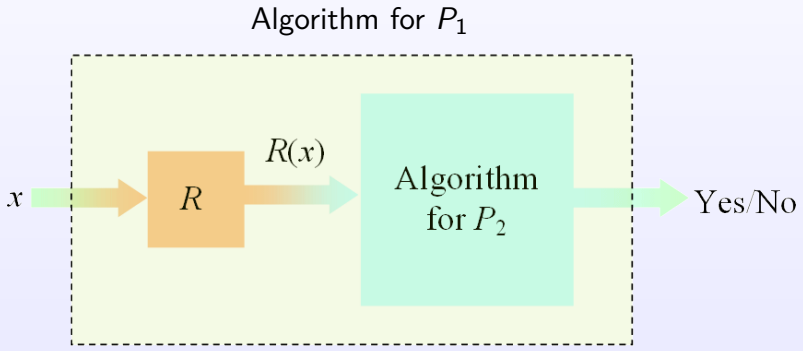
Problem Reduction

A reduction is a transformation from a problem to another.

Problem P_1 is reducible to problem P_2 if there exists an efficient procedure R that transforms any $x \in I_{P_1}$ into $R(x) \in I_{P_2}$.

If there exists a reduction from P_1 to P_2, then P_2 is at least as hard to solve as P_1.

Problem Reduction



Hard Problems

Hard Problem for a Class

Given class **C** and a problem P, P is said to be **C**-hard if any other $P' \in \mathbf{C}$ can be reduced to P.

Intuitively, a **C**-hard problem is at least as hard as any problem in **C**, and possibly harder.

Complete Problems

Complete Problem for a Class

Given class **C** and a problem P, P is said to be **C**-complete if it is **C**-hard and $P \in \mathbf{C}$.

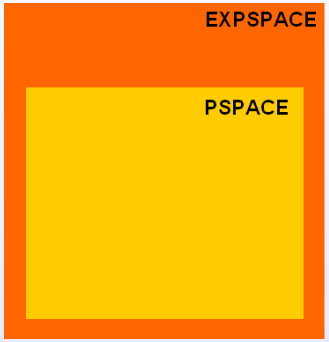
Complete problems for a class are usually the most interesting ones, since they characterize the class and its hardness.

Polynomial Space

PSPACE=**SPACE**(n^k) is the class of problems solvable using polynomial space.

EXPSPACE=**SPACE**(2^{n^k}) is the class of problems solvable using exponential space.

Obviously **PSPACE** $\subset$ **EXPSPACE**.



Characterizing **PSPACE**

QSAT (*quantified satisfiability*) is a **PSPACE**-complete problem.

QSAT

Given a totally quantified logical expression

$$Q_1 x_1 Q_2 x_2 \cdots Q_n x_n \phi(x_1, x_2, \cdots, x_n)$$

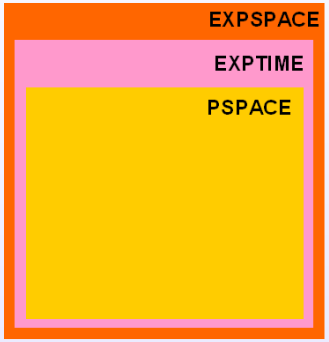
where Q_i is $\forall$ or $\exists$, is this expression true or false?

Exponential Time

EXPTIME=**TIME**(2^{n^k}) is the class of problems solvable in exponential time by a deterministic algorithm.

It is easy to see that **EXPTIME** $\subseteq$ **EXPSPACE**.

Also, **PSPACE** $\subseteq$ **EXPTIME**.



Game Complexity

PSPACE naturally characterizes the complexity of many 2-player games

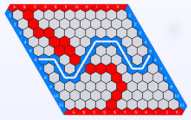
A game strategy is a winning strategy for player A if (and only if):

- when it is A's turn **there exists** at least a winning move.
- when it is B's turn, **for any move** he makes there is a subsequent winning move for A.

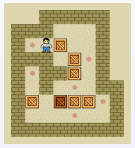
Note the relationship with existential ($\exists$) and universal ($\forall$) quantifiers.

Of course, we need the game duration be polynomial in the board size.

Game Complexity


Hex


Reversi


Sokoban

PSPACE-complete


Go


Chess

EXPTIME-complete

Non-Determinism

A non-deterministic algorithm can take random decisions during its execution.

For analysis purposes, the computation can be imagined to branch out, generating copies of the algorithm (each one corresponding to a possible random decision) that run in parallel.

As long as one of the branches finds the solution (i.e., determines that instance x is a yes-instance), the problem is solved.

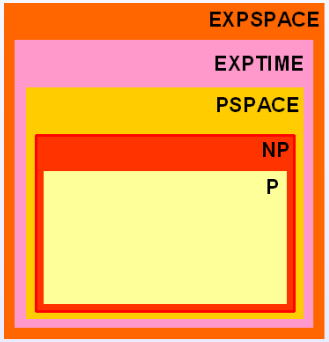
This non-deterministic computational model cannot compute anything a deterministic mode cannot do as well. However, it can do things more efficiently.

P vs NP

NP=**NTIME**(n^k) is the class of problems solvable in polynomial time by a non-deterministic algorithm.

P=**TIME**(n^k) is the class of problems solvable in polynomial time by a deterministic algorithm.

Clearly, **P** $\subseteq$ **NP**.



Class **NP**

SAT (*satisfiability*) is a paradigmatic **NP**-complete problem.

SAT

Given a logic formula in normal conjunctive form

$$C_1 \wedge C_2 \wedge \cdots C_m$$

where $C_i = (v_i^1 \vee v_i^2 \vee \cdots v_i^{r_i})$ and v_i^j is a logic variable or its negation, is there a truth assignment that makes the formula true?

Alternative Definition of **NP**

NP is composed of problems for which there exists a succinct certificate for any yes-instance.

In other words, given a tentative solution $y \in sol_P(x)$, it is possible to determine whether y is feasible in **TIME**(n^k).

Consider for example SAT: if we are given a truth assignment, we can easily check whether it makes the formula true.

A non-deterministic algorithm can "guess" the solution in linear-time and verify it in poly-time.

Class **P**

Problems in **P** can be solved by a TM in **TIME**(n^k).

HORNSAT (*Horn satisfiability*) is **P**-complete.

HORNSAT

A Horn clause is a disjunction of literals in which at most one is not negated. Given a Horn formula

$$C_1 \wedge C_2 \wedge \cdots C_m$$

where each C_i is a Horn clause, is there a satisfying truth assignment?

Note that HORNSAT is particular case of SAT, and therefore **P** $\subseteq$ **NP**.

Some Problems in **P**

Some problems in **P**:

- Linear programming: given a matrix $\mathbf{A}_{m\times n} \in \mathbb{Q}^{m\times n}$ and two vectors $\mathbf{b}_{m\times 1} \in \mathbb{Q}^m$ and $\mathbf{c}_{1\times n} \in \mathbb{Q}^n$, find a vector $\mathbf{x}_{n\times 1} \in \mathbb{Q}^n$ such that $\mathbf{c}\cdot\mathbf{x}$ is maximal, subject to $\mathbf{A}\cdot\mathbf{x} \leqslant \mathbf{b}$.
- Primality check: given a natural number $n \in \mathbb{N}$ check whether it is prime or not.
- Maximum matching: given a graph $G(V, E)$, find the largest edge subset $E' \subseteq E$ such that no two edges in E' are adjacent.

P vs **NP**

P represents a largely accepted notion of tractability, much like membership to **NP** is often considered a proof of intractability.

This distinction is useful, but there is a lot of pathology involved: why should $O(n^{15})$ be better in practice than $O(2^{\frac{n}{100}})$?

$\mathbf{P} \subseteq \mathbf{NP}$, but $\mathbf{P} = \mathbf{NP}$? It is an open problem. The answer is worth 10^6 US$!

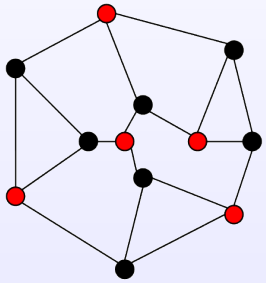
We do know that either $\mathbf{P} = \mathbf{NP}$, or there exist problems in **NP** that neither are in **P** nor are **NP**-complete.

Some **NP**-complete Problems

Independent Set

Independent Set

Given a graph $G(V, E)$ and a natural number K, is there a subset $V' \subseteq V$, such that $|V| = K$ and for all $(u, v) \in E$ either $u \notin V'$, $v \notin V'$ or both?



Some **NP**-complete Problems

Independent Set

It is easy to see that Independent Set $\in$ **NP**.

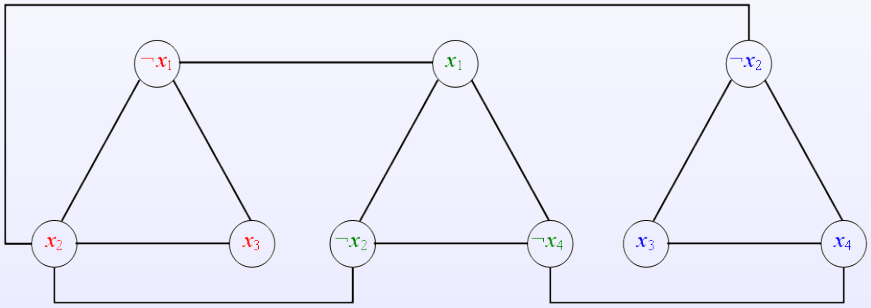
A reduction from SAT to Independent Set can be done as follows: given a formula $C_1 \wedge C_2 \wedge \cdots C_m$, build a graph $G(V, E)$ with a node for each literal in a clause, and

- an edge between any two literals in the same clause.
- an edge between any literal and its negation if the latter appears in any other clause.

The formula is satisfiable if, and only if, there exists an independent set of size m.

Reduction from 3-SAT to Independent Set

$$(\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_4) \wedge (\neg x_2 \vee x_3 \vee x_4)$$



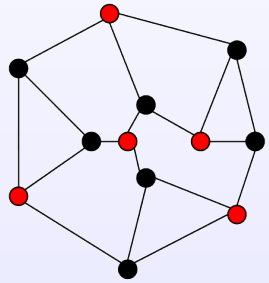
Subset $\{\neg x_1, \neg x_2, x_4\}$ is a size-3 independent set, and a valid solution for the 3-SAT instance shown.

Some **NP**-complete Problems

Vertex Cover

Vertex Cover

Given a graph $G(V, E)$ and a natural number K, is there a subset $V' \subseteq V$, such that $|V| = K$ and for all $(u, v) \in E$ it holds that $u \in V'$ or $v \in V'$?



Some **NP**-complete Problems

Vertex Cover

It is easy to see that Vertex Cover $\in$ **NP**.

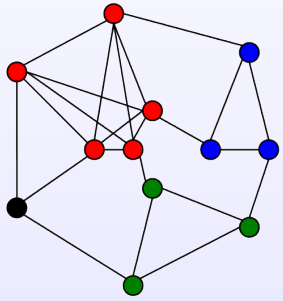
A reduction from Independent Set to Vertex Cover can be done as follows: given a graph $G(V, E)$, the existence of an independent set S of size K implies that $V \setminus S$ is a vertex cover of size $n - K$.

Some **NP**-complete Problems

Clique

Clique

Given a graph $G(V, E)$ and a natural number K, is there a subset $V' \subseteq V$, such that $|V| = K$ and for all $u, v \in V'$ it holds that $(u, v) \in E$?



Some **NP**-complete Problems

Clique

It is easy to see that Clique $\in$ **NP**.

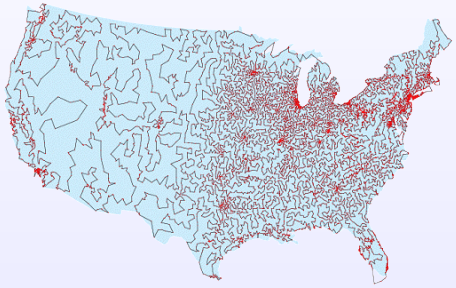
A reduction from Independent Set to Clique can be done as follows: given a graph $G(V, E)$, the existence of an independent set S of size K implies that S is a clique in the complementary graph $\bar{G}(V, \bar{E})$, in which $(u, v) \in \bar{E} \Leftrightarrow (u, v) \notin E$.

Some **NP**-complete Problems

Travelling Salesman Problem (TSP)

Travelling Salesman Problem

Given a complete weighted graph $G(V, E, W)$, is there a tour that visits once each node in the graph, such that its length less or equal to K?



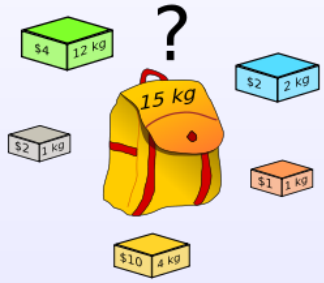
Optimal tour traversing 13,509 cities in the USA.

Some **NP**-complete Problems

0-1 Knapsack

0-1 Knapsack

Given a collection O of n objects with weights $p_1, \cdots, p_n$ and values $v_1, \cdots, v_n$, is there a subset $S \subseteq O$ whose total value is greater or equal than K, such that its total weight is less or equal than W?

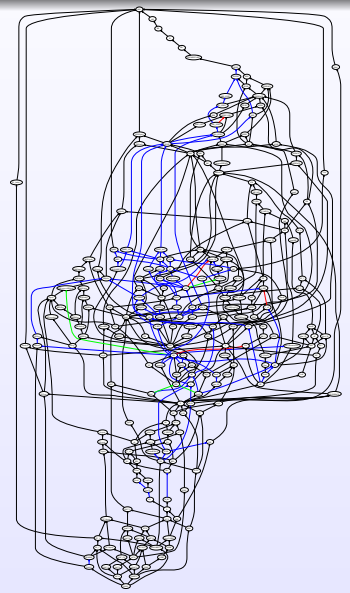


And many more...

A plethora of problems have been shown to be **NP**-complete.

A Compendium of NP Optimization Problems
Pierluigi Crescenzi, Viggo Kann (editors)
`http://www.nada.kth.se/~viggo/wwwcompendium/`

The Complexity Class Zoo



There are many other complexity classes.

There are also richer approaches to quantify complexity from a multidimensional perspective (parameterized complexity).

What about optimization problems?

The complexity classes defined before are based on decision problems.

Solving an optimization problem will be at least as hard –and in general harder– than solving the decisional version.

If $P^{(D)}$ is the decisional version of problem P and $P^{(D)}$ is **C**-complete, its optimization version $P^{(O)}$ is **C**-hard.

If **P** was equal to **NP**...

... we could solve **NP**-hard optimization problems in polynomial time.

Graph Coloring

Given a graph $G(V, E)$, assign a color $c(v) \in \mathbb{N}$ to each vertex $v \in V$, such that for any edge $(v, w) \in E$ it holds that $c(v) \neq c(w)$.

The decisional version asks whether a certain graph can be colored with k colors or fewer.

The optimization version tries to find the minimal k such that the graph is k-colorable. Such k is called the chromatic number χ_G of G.

Imagine **P**=**NP**

Graph coloring is an **NP**-complete problem. If we had a polynomial-time algorithm IsColorable(G, k) telling whether G is k-colorable, then we could define the following algorithm:

Chromatic Number

$i \leftarrow 1$;
while ¬IsColorable(G,i) **do** $i \leftarrow i + 1$ **endwhile**
$\chi_G \leftarrow i$

Any graph is n-colorable, $n = |V|$, so we need to call IsColorable at most n times. This is polynomial in the input size, and since IsColorable was assumed to be polynomial as well, so would be the computation of the Chromatic Number.

How to tackle problems not in **P**?

If we want to solve a decision/optimization problem not in **P**, in the worst case we will have a super-polynomial (often exponential or even super-exponential) runtime.

Available options:

- Use an exact algorithm: sometimes the worst case is rare in practice and in general the resolution can be efficient.
- Use an approximate algorithm: not practical in general.
- Use an heuristic algorithm: we will obtain probably good solution, although we cannot prove this a priori.

Complementary Bibliography


C.H. Papadimitriou
Computational complexity,
Addison-Wesley, 1994.


P. Crescenzi, V. Kann
A Compendium of NP Optimization Problems,
https://www.csc.kth.se/tcs/compendium/


S. Aaronson *et al.*
Complexity Zoo,
https://complexityzoo.net/Complexity_Zoo

Image Credits

- Turing Machine: By Schade - Own work, free use
- Hex board: By Jean-Luc W - Own work, CC BY-SA 3.0
- Reversi: By Zen ar218 - Own work, Public Domain
- Sokoban: By Carloseow at English Wikipedia, CC BY 3.0
- Go: By Donarreiskoffer - Self-photographed, CC BY-SA 3.0
- Chess: By Alan Light - Own work by the original uploader, CC BY-SA 3.0
- Vertex Cover: Weisstein, Eric W. "Vertex Cover." From MathWorld – A Wolfram Web Resource. https://mathworld.wolfram.com/VertexCover.html
- TSP: David Applegate, Robert Bixby, Vasek Chvatal and William Cook
- Knapsack: By Dake, CC BY-SA 3.0
- Complexity class inclusion diagram: Greg Kuperberg, https://www.math.ucdavis.edu/~greg/zoology/diagram.pdf