NP completeness

Course "Discrete Biological Models" (Modelli Biologici Discreti)

Zsuzsanna Lipták

Laurea Triennale in Bioinformatica
a.a. 2014/15, fall term

NP-completeness and the real world

Imagine you are working for a biotech company. One day your boss calls you and tells you that they have invented a new sequencing technology. It generates lots of fragments of the target molecule, which may overlap. Your job as **chief algorithm designer** is to write a program that reconstructs the target molecule.

You get to work ...

NP-completeness and the real world

Imagine you are working for a biotech company. One day your boss calls you and tells you that they have invented a new sequencing technology. It generates lots of fragments of the target molecule, which may overlap. Your job as **chief algorithm designer** is to write a program that reconstructs the target molecule.

You get to work ...
you find that ...

- you need a superstring (of all fragments)
- a shortest superstring
- you need to maximize the overlaps
- any superstring is just a permutation of the fragments
- you need the/a best permutation (which maximizes total overlap)

NP-completeness and the real world (2)

But after weeks and weeks and weeks ...
all you have come up with is:

Exhaustive algorithm

List all possible permutations and choose the best.

NP-completeness and the real world (3)

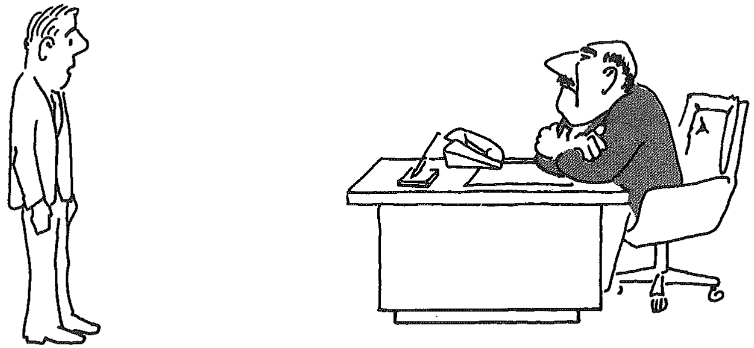
This is no good, since for 1,000 fragments (typical experiment) this would need far longer than the age of the universe. Why?

- $1000! \approx 4 \cdot 10^{2567}$ permutations — source: Wolfram Alpha
- say we need 1000 operations per permutation (summing overlaps)
- if we have a computer that does 1 billion ($= 10^9$) operations/sec
- it can handle 1 million (10^6) permutations per second
- So it will take $4 \cdot 10^{2567}/10^6 = 4 \cdot 10^{2561}$ seconds

Age of the universe:
about 14 billion years $\approx 4 \cdot 10^{17}$ seconds — source: Wikipedia

We can see that a million (10^6) or a billion (10^9) times faster computer wouldn't help much, either. Nor would faster handling of the permutations.

So you can go to your boss and say:

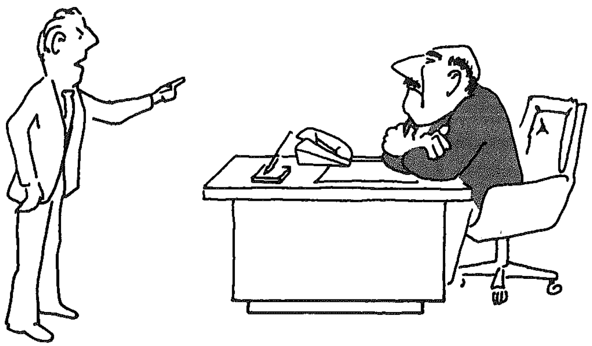


"I can't find an efficient algorithm, I guess I'm just too dumb."

Bad idea, you may get fired!

source: Garey & Johnson, A Guide to the Theory of NP-completeness, 1979

Or you could say:



"I can't find an efficient algorithm, because no such algorithm is possible!"

Unfortunately, it is very hard to do impossibility proofs...

source: Garey & Johnson, A Guide to the Theory of NP-completeness, 1979

You prove that the reconstruction problem is NP-complete, and you say:

"I can't find an efficient algorithm, but neither can all these famous people."

So it makes no sense to fire you and get another expert!

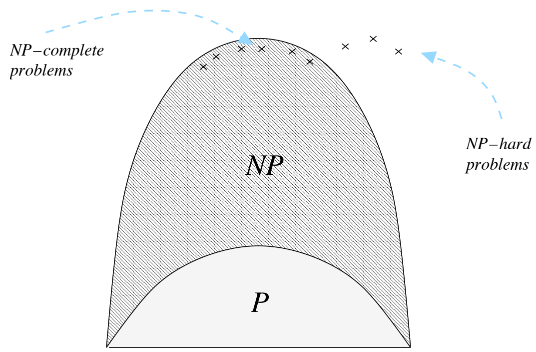
source: Garey & Johnson, A Guide to the Theory of NP-completeness, 1979

Overview

- P - class of problems that can be solved efficiently (i.e., in polynomial time)
- NP - class of problems that can solved in polynomial time by a non-deterministic Turing machine (nicer definition to follow)
- NP-complete problems - maximally difficult problems in NP: every other problem can be transformed into these in polynomial time (details later)
- NP-hard problems - like NP-complete problems, but not necessarily in NP

Overview

Usual way of drawing the classes P and NP:



The class P

Definition

A decision problem X is in P if there is a polynomial time algorithm A which solves X.

A decision problem is one that allows only YES or NO answers.

Examples

- Given a sorted array of n numbers, is number x present? $O(\log n)$ time
- Given an array of n numbers, is number x present? $O(n)$ time
- Given a graph G, is G Eulerian? $O(n+m)$ time, where $n = |V|, m = |E(G)|$.
  Details: Determine for each vertex whether it has even degree (or whether it is balanced if G is a digraph) in $O(n)$ time; determine with BFS whether G is connected in $O(n+m)$ time.

Polynomial time algorithms

Algorithm A is polynomial time if there is a polynomial p s.t. for every input I of size n, A terminates in at most $p(n)$ steps.

size is usually measured in bits, number of elements (for an integer array), number of vertices and edges (for graphs), number of characters (for strings)

The class NP

NP - class of problems that can solved in polynomial time by a non-deterministic Turing machine (non-deterministic polynomial time)

alternative definition:

NP = class of polynomial time checkable problems

Whenever the answer is YES, there must exist a certificate (proof) of this, and it must be checkable (verifiable) in polynomial time.

Example

Problem: Given a graph $G = (V, E)$, is G Hamiltonian (i.e. does it have a Hamiltonian cycle)?

Certificate: A Hamiltonian cycle: check whether it is a cycle, and whether it contains every vertex exactly once, in $O(n)$ time, where $n = |V|$.

The class NP

NP = class of polynomial time checkable problems

Whenever the answer is YES, there must exist a certificate (proof) of this, and it must be checkable (verifiable) in polynomial time.

Example

Problem: Given a fragment set $\mathcal{F}$ of m fragments, does a superstring of length $\leq k$ exist?

Certificate: a superstring S of length $\leq k$: check for every f whether f is substring of S, in $O(|f| + |S|)$ time; altogether $O(||\mathcal{F}|| + m|S|)$ time, so polynomial in input size.

input: $\mathcal{F}$, inputsize: $||\mathcal{F}|| = \sum_{i=1}^{m} |f_i|$, $|S| \leq \sum_i |f_i|$, so $m|S| \leq ||\mathcal{F}||^2$

The class NP

NP = class of polynomial time checkable problems

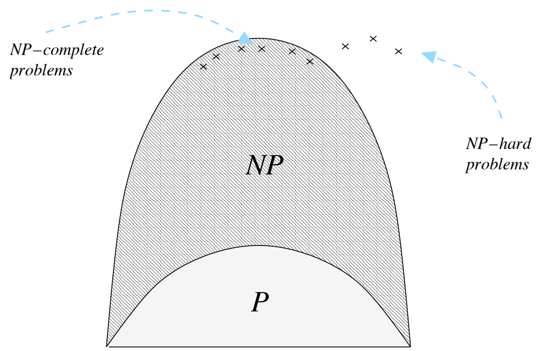
Whenever the answer is YES, there must exist a certificate (proof) of this, and it must be checkable (verifiable) in polynomial time.

Example

Problem: Given a complete digraph $G = (V, E)$ with non-negative weights on edges, does it have a Hamiltonian path of weight at least r?

Certificate: a Ham. path of weight $\geq r$: check weight of path, check whether it contains every vertex exactly once, in $O(n)$ time, where $n = |V|$

P and NP

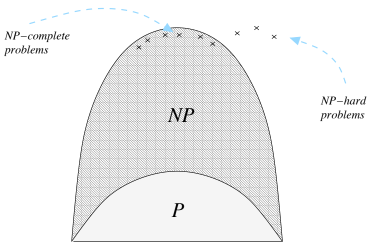


The $P = NP$ question

One of the big open question of computer science is:

Is $P = NP$?

- Since $P \subseteq NP$ is clear, the question is: $NP \subseteq P$?
- Can every problem in NP be solved efficiently?
- I.e. is the shaded area empty?



N.B.: There is a US \$ 1,000,000 prize for solving this question.

The $P = NP$ question

Most people believe: $P \neq NP$.

NP-complete and NP-hard problems

NP-complete problems

A problem X is NP-complete if

- it is in NP
- every problem in NP can be transformed/reduced to it in polynomial time (details soon)

NP-hard problems

A problem X is NP-hard if

- every problem in NP can be transformed/reduced to it in polynomial time (details soon)

i.e., the only difference is that it is not necessarily itself in NP.

NP-complete and NP-hard problems

Theorem

Let X be an NP-complete or NP-hard problem. If we find a polynomial time algorithm for X, then $P = NP$.

Corollary

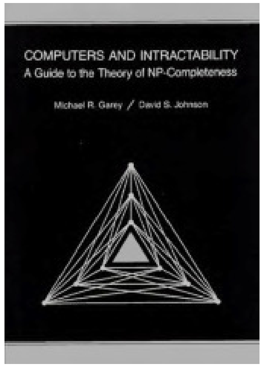
Let X be an NP-complete or NP-hard problem. Then there is no polynomial time algorithm for X, unless $P = NP$.

Since we all believe that $P \neq NP$, this means in practical terms:

In practice

Let X be an NP-complete or NP-hard problem. Then there is no polynomial time algorithm for X, fullstop.

NP-complete problems

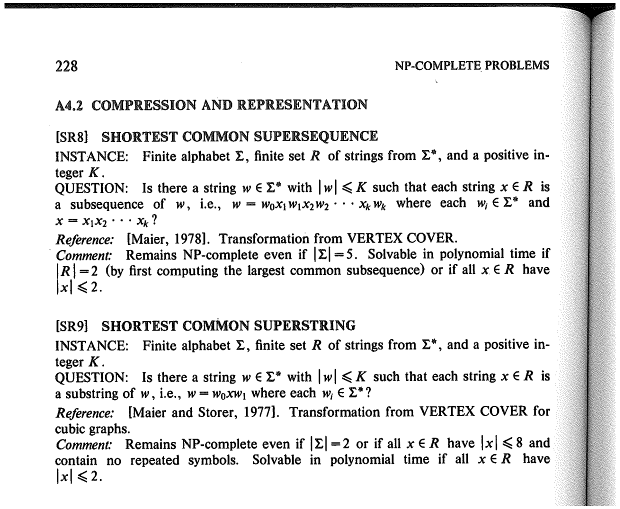


Michael R. Garey, David S. Johnson, *Computers and Intractability - A Guide to the Theory of NP-completeness*, 1979

one of the best known and most cited books ever in computer science

NP-complete problems

Contains a list of known NP-complete problems:



228 NP-COMPLETE PROBLEMS

A4.2 COMPRESSION AND REPRESENTATION

[SR8] SHORTEST COMMON SUPERSEQUENCE

INSTANCE: Finite alphabet Σ, finite set R of strings from Σ^*, and a positive integer K.
QUESTION: Is there a string $w \in \Sigma^*$ with $|w| \leqslant K$ such that each string $x \in R$ is a subsequence of w, i.e., $w = w_0 x_1 w_1 x_2 w_2 \cdots x_k w_k$ where each $w_i \in \Sigma^*$ and $x = x_1 x_2 \cdots x_k$?
Reference: [Maier, 1978]. Transformation from VERTEX COVER.
Comment: Remains NP-complete even if $|\Sigma| = 5$. Solvable in polynomial time if $|R| = 2$ (by first computing the largest common subsequence) or if all $x \in R$ have $|x| \leqslant 2$.

[SR9] SHORTEST COMMON SUPERSTRING

INSTANCE: Finite alphabet Σ, finite set R of strings from Σ^*, and a positive integer K.
QUESTION: Is there a string $w \in \Sigma^*$ with $|w| \leqslant K$ such that each string $x \in R$ is a substring of w, i.e., $w = w_0 x w_1$ where each $w_i \in \Sigma^*$?
Reference: [Maier and Storer, 1977]. Transformation from VERTEX COVER for cubic graphs.
Comment: Remains NP-complete even if $|\Sigma| = 2$ or if all $x \in R$ have $|x| \leqslant 8$ and contain no repeated symbols. Solvable in polynomial time if all $x \in R$ have $|x| \leqslant 2$.

NP-complete problems

You have found your problem (the Shortest Common Superstring Problem) in the Garey-Johnson:

"I can't find an efficient algorithm, but neither can all these famous people."

Decision problems vs. optimization problems

Optimization problem

Given $\mathcal{F}$, find a shortest common superstring S.

Decision problem

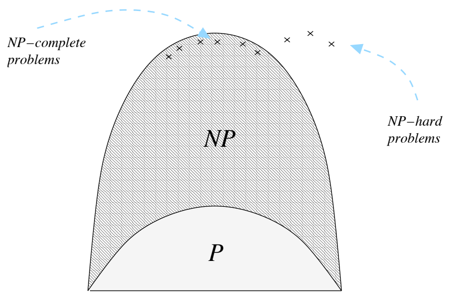
Given $\mathcal{F}$, does a common superstring S exist with $|S| \leq k$?

N.B.

If I can solve the decision problem, then I can also solve a reduced version of the optimization problem: determining the length of an SCS.
Use $\log(||\mathcal{F}||)$ many calls to the decision problem.

(Details: Determine the length of an SCS using binary search: Let $N = ||\mathcal{F}||$. We know that a common superstring exists with length N, namely the concatenation of all $f \in \mathcal{F}$. Now set $k = N/2$; if the answer is YES, continue with $k = N/4$, else with $k = 3N/4$, etc.)

Decision problems vs. optimization problems



- Many *NP*-hard problems are optimization versions of *NP*-complete problems.
- If the decision version is *NP*-complete, then the optimization version is automatically *NP*-hard. (Why?)
- So it's enough to prove that the decision version is *NP*-complete. (Or to find it in the Garey-Johnson.)

Polynomial time reductions/transformations

Now we want to talk about *NP*-complete problems. These are the ones to which all others in *NP* can be reduced in polynomial time. What does this mean?

"Translate problem X to problem Y in polynomial time"

Definition

We say that problem X is polynomial time reducible to problem Y, $X \leq_p Y$, if there is an algorithm A and a polynomial p, s.t. for every instance I of X, A constructs an instance J of Y in time $p(|I|)$ such that:

$$I \text{ is a YES-instance of } X \quad \Leftrightarrow \quad J \text{ is a YES-instance of } Y.$$

An example reduction

SCS-decision

Given: Fragment set $\mathcal{F}$

Question:
Does a superstring of length $\leq k$ exist?

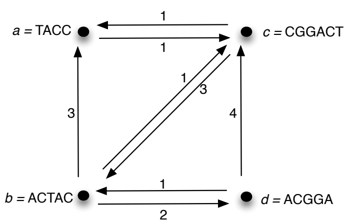
Weighted Hamiltonian path-decision

Given: A weighted complete digraph G.

Question: Does G have a Ham. path of weight $\geq ||\mathcal{F}|| - k$?

Example

$\{\texttt{TACC}, \texttt{CGGACT}, \texttt{ACTAC}, \texttt{ACGGA}\}$



An example reduction

Let $\mathcal{F}$ contain m fragments of length r.

- Time for transformation: $O(m^2r^2)$, (or even faster): the time of the transformation is dominated by the time for computing the overlaps between the fragments, i.e. the weights on the edges
- This is polynomial in the input size $||\mathcal{F}|| = mr$ (namely, with polynomial $p(n) = n^2$).
- YES on the left $\Leftrightarrow$ YES on the right
- Therefore we have shown:
  SCS-decision $\leq_p$ Weighted Hamiltonian path-decision

Polynomial time reductions/transformations

Definition (again)

We say that problem X is polynomial time reducible to problem Y, $X \leq_p Y$, if there is an algorithm A and a polynomial p, s.t. for every instance I of X, A constructs an instance J of Y in time $p(|I|)$ such that:

$$I \text{ is a YES-instance of } X \quad \Leftrightarrow \quad J \text{ is a YES-instance of } Y.$$

N.B.

- Think of $X \leq_p Y$ as meaning: X is "not harder" than Y
- Note that we are still paying for the reduction/transformation: it takes polynomial time!

There are important technical differences between Karp-reductions/transformations (R. Karp, 1972), and Turing-reductions (S. Cook, 1971), which we are ignoring here.

Polynomial time reductions/transformations

NP-complete problems

A problem X is NP-complete if

- it is in NP
- for every problem $Y \in NP$: $Y \leq_p X$.

Theorem

Let X be an NP-complete problem. If we find a polynomial time algorithm for X, then $P = NP$.

Polynomial time reductions/transformations

Theorem

Let X be an NP-complete problem. If we find a polynomial time algorithm for X, then $P = NP$.

Proof

Let A be a polytime algorithm for X. We have to show: $NP \subseteq P$. Let Y be any problem in NP. We will now give a polytime algorithm for Y. Let I be an instance of Y. Since X is NP-complete, there is an algorithm B which transforms any instance I of Y into an instance J of X in polynomial time, say in $p_B(|I|)$. In particular this implies $|J| \leq p_B(|I|)$. The algorithm A for X solves J in time $p_A(|J|) \leq p_A(p_B(|I|))$ (since all parameters are positive). Thus, we have an algorithm $C = A \circ B$ (B followed by A) for Y, which gives an answer to instance I in time at most $p_C(|I|) := p_B(|I|) + p_A(p_B(|I|))$, which, being a sum and composition of polynomials, is a polynomial in $|I|$.

Is your problem NP-complete?

So if you have a computational problem, and you think it might be NP-complete, then

- Find it in the Garey-Johnson, or some other compendium of NP-complete/NP-hard problems, or
- prove that it is NP-complete.
  This is far beyond the scope of this course ...
  You will learn how to do this in an advanced course on algorithms/complexity.

Recap

- NP: problems which are polynomial-time checkable
  (for YES-instances, a certificate exists which can be verified in polynomial time)
- polynomial time reduction/transformation, $X \leq_p Y$, means: instances of X can be transformed to instances of Y in polynomial time
- NP-complete problems: problems in NP to which all problems in NP can be polytime reduced
- NP-hard problems: like NP-complete but not necessarily in NP
- An efficient (=polytime) algorithm for an NP-complete or NP-hard problem would imply: P=NP, which nobody believes is true
- Therefore, no efficient algorithm can exist for these problems (we all believe)

Why do we believe that $P \neq NP$?

- Many, many important real-life problems are NP-complete or NP-hard.
- Finding an efficient algorithm for just one of these would prove $P = NP$.
- Much much work has gone into finding efficient algorithms for these.
- By some of the best mathematicians and computer scientists on earth.
- No one has been able to find an efficient algorithm for any one of these problems so far.

Why do we believe that $P \neq NP$?

...by some of the best mathematicians and computer scientists on earth ...

"I can't find an efficient algorithm, but neither can all these famous people."

What to do next?

If we find that our computational problem is NP-complete or NP-hard, then what can we do?

- despair/give up
- Run an exhaustive search algorithm – This is often possible for small instances, often combined with specific tricks.
- Verify that your instances are really general. Often, the general case is NP-complete, but many special cases are not!
  E.g. the SBH-sequencing problem is not NP-complete: otherwise we could not have found a better formulation (Euler cycles in de Bruijn graphs) which is polynomially solvable!

(continued on next page)

What to do next?

(continued from previous page)

- Devise heuristics: algorithms that work well in practice but without guaranteeing the quality of the solution.
- Polynomial time approximation algorithms: Algorithms that do not guarantee the optimal solution, but a solution which approximates the optimum. E.g. the Greedy Algorithm for SCS guarantees a solution which is at most 4 times longer than the optimum.
- and, and, and ...

Summary

- There are certain problems for which no efficient algorithms exist (very, very, very, very probably)
- These are called NP-complete (decision problems) or NP-hard (optimization problems)
- Many real-life problems, also in computational biology/bioinformatics, are NP-complete/NP-hard
- There are ways of dealing with this (see previous list)

Summary

Merry Christmas!