Computationally Hard Problems

Randomized Algorithms, Complexity Classes and the Class $\mathcal{NP}$

Carsten Witt

Institut for Matematik og Computer Science
Danmarks Tekniske Universitet

Fall 2020

Computationally Hard Problems

Randomized Complexity Classes, Problems in the Class $\mathcal{NP}$

Carsten Witt

Institut for Matematik og Computer Science
Danmarks Tekniske Universitet

Fall 2020

Complexity Classes

- A *complexity class* is a collection of problems which are "equally difficult" to solve.
- "Difficulty" can be measured as use of resources.
- "Difficulty" can also be measured by how much additional information helps to solve the problem.
- Here, we measure how much (how often) randomization helps.

The Class $\mathcal{P}$

Definition

A yes-no–problem is in $\mathcal{P}$ if there is a polynomial p and a **deterministic** p-bounded algorithm A such that for every input $\boldsymbol{X}$ the following holds:
True answer for $\boldsymbol{X}$ is YES then $A(\boldsymbol{X}) = \mathrm{YES}$
True answer for $\boldsymbol{X}$ is NO then $A(\boldsymbol{X}) = \mathrm{NO}$

These are the "good old" efficiently deterministically solvable problems.

The Class $\mathcal{NP}$

Definition

A yes-no–problem is in $\mathcal{NP}$ ("nondeterministic polynomial") if there is a polynomial p and a randomized p-bounded algorithm A such that for every input $\boldsymbol{X}$ the following holds:
True answer for $\boldsymbol{X}$ is YES then $\exists R, \|R\| \leq p(\|\boldsymbol{X}\|) : A(\boldsymbol{X}, R) = \mathrm{YES}$
True answer for $\boldsymbol{X}$ is NO then $\forall R : A(\boldsymbol{X}, R) = \mathrm{NO}$
Here R is a sequence of random numbers of the type required by the algorithm.

An algorithm with these properties is called an $\mathcal{NP}$-*algorithm*.

The Class $\mathcal{NP}$

Alternative definition

Definition

A yes-no–problem is in $\mathcal{NP}$ if there is a polynomial p and a randomized p-bounded algorithm A such that for every input $\boldsymbol{X}$ the following holds:
True answer for $\boldsymbol{X}$ is YES then $\boldsymbol{P}_R\,[\,A(\boldsymbol{X},R)=\mathrm{YES}\,]>0$
True answer for $\boldsymbol{X}$ is NO then $\boldsymbol{P}_R\,[\,A(\boldsymbol{X},R)=\mathrm{NO}\,]=1$
where $\boldsymbol{P}_R\,[\,Z\,]$ denotes the probability of event Z over uniform distribution of R,
$\|R\|\leq p(\|\boldsymbol{X}\|)$.

The Class $\mathcal{RP}$

Definition

A yes-no–problem is in $\mathcal{RP}$ (*random polynomial time*) if there is a polynomial p and a randomized p-bounded algorithm A such that for every input $\boldsymbol{X}$ the following holds:
True answer for $\boldsymbol{X}$ is YES then $\boldsymbol{P}_R\,[\,A(\boldsymbol{X},R)=\mathrm{YES}\,] \geq \frac{1}{2}$
True answer for $\boldsymbol{X}$ is NO then $\boldsymbol{P}_R\,[\,A(\boldsymbol{X},R)=\mathrm{NO}\,] \;\; = 1$
An algorithm with these properties is called an $\mathcal{RP}$-*algorithm*.

$\mathcal{RP}$-algorithms are also called *Monte Carlo* algorithms. They have one-sided error. In contrast to $\mathcal{NP}$-algorithms, there is a good chance of getting the correct result for YES-inputs.

The Class $\mathcal{BPP}$

Definition

A yes-no–problem is in $\mathcal{BPP}$ (*bounded error probabilistic polynomial*) if there is a polynomial p and an $\varepsilon > 0$ and a randomized p-bounded algorithm A such that for every input $\boldsymbol{X}$ the following holds:

True answer for $\boldsymbol{X}$ is YES : $\boldsymbol{P}_R\,[\,A(\boldsymbol{X}, R) = \mathrm{YES}\,] \geq \frac{1}{2} + \varepsilon$

True answer for $\boldsymbol{X}$ is NO : $\boldsymbol{P}_R\,[\,A(\boldsymbol{X}, R) = \mathrm{NO}\,] \geq \frac{1}{2} + \varepsilon$

An algorithm with these properties is called a $\mathcal{BPP}$-*algorithm*.

It has two-sided error.

The Class $\mathcal{ZPP}$

Definition

A yes-no–problem is in $\mathcal{ZPP}$ (*zero error probabilistic polynomial*) if there is a polynomial p and a randomized p-bounded algorithm A such that for every input $\boldsymbol{X}$ the following holds:

True answer for $\boldsymbol{X}$ is YES : $\boldsymbol{P}_R\,[\,A(\boldsymbol{X},R)=\mathrm{YES}\,] \geq \frac{1}{2}$

True answer for $\boldsymbol{X}$ is YES : $\boldsymbol{P}_R\,[\,A(\boldsymbol{X},R)=\mathrm{NO}\,] = 0$

True answer for $\boldsymbol{X}$ is NO : $\boldsymbol{P}_R\,[\,A(\boldsymbol{X},R)=\mathrm{NO}\,] \geq \frac{1}{2}$

True answer for $\boldsymbol{X}$ is NO : $\boldsymbol{P}_R\,[\,A(\boldsymbol{X},R)=\mathrm{YES}\,] = 0$

An algorithm with these properties is called a $\mathcal{ZPP}$-*algorithm*. It never answers wrong, but might refuse to answer (DON'T KNOW).

$\mathcal{ZPP}$-algorithms are also called *Las Vegas* algorithms.

Putting a Problem into a Complexity Class

The problem

We present a case study consisting of a problem, a randomized algorithm and a classification of the algorithm and problem.

The problem:

- Given are two arrays A and B of length n.
- Both contain the same kind of objects.
- Some object x_0 is contained at least $k > 0$ times in both A and B; or the arrays do not share a common object.
- We want to find some object which is in both A and B: answer YES iff there is such an object.

If the objects come from an ordered set, then one could order A and B and traverse both. The time for this $O(n \log(n))$.

Example

A randomized algorithm

Consider randomized algorithm $\textsc{FindInBoth}(A, B, t)$:

1. Set count to 0.
2. Randomly pick $a \in A$ and $b \in B$.
3. If $a = b$ answer YES and stop.
4. Increment count.
5. If $\text{count} < t$ goto 2.
6. Answer NO.

Example

A randomized algorithm

Pseudo code for FindInBoth(A, B, t):

proc FindInBoth(A, B, t)
$c \leftarrow 0$
while $c < t$ **do**
  $i \leftarrow \text{rand}(1, n); j \leftarrow \text{rand}(1, n)$
  if $A[i] = B[j]$ **then**
    return(YES)
  end if
  $c \leftarrow c + 1$
end while
return(NO)
end proc

Example

Facts

What do we want to know about the algorithm?

- The running time is (non-)deterministic?
- The result is always correct?
- Is the error one-sided?
- What is the success probability?
- Can one have 50% success probability?
- Is the algorithm $\mathcal{NP}$, $\mathcal{RP}$, ...?

Running Time

proc FINDINBOTH(A, B, t)
$c \leftarrow 0$
while $c < t$ **do**
  $i \leftarrow \text{rand}(1, n); j \leftarrow \text{rand}(1, n)$
  if $A[i] = B[j]$ **then**
    return(YES)
  end if
  $c \leftarrow c + 1$
end while
return(NO)
end proc

The worst-case time is deterministically bounded by $O(t)$.

Is the Answer always Correct?

```
proc FindInBoth(A, B, t)
c ← 0
while c < t do
   i ← rand(1, n); j ← rand(1, n)
   if A[i] = B[j] then
      return(YES)
   end if
   c ← c + 1
end while
return(NO)
end proc
```

No; even if there is a common element in both arrays the algorithm might always guess some that are different (assuming $n > 1$).

One-sided Error

YES.

Probability of Success

- Element x_0 is present $k > 0$ times in both arrays.
- To pick x_0 in both arrays happens with prob. $\left(\frac{k}{n}\right)^2$.
- To miss x_0 in at least one array happens with prob. $1 - \left(\frac{k}{n}\right)^2$.
- To miss t times happens with prob. $\left(1 - \left(\frac{k}{n}\right)^2\right)^t$.
- $\left(1 - \left(\frac{k}{n}\right)^2\right)^t$ is at most $1/2$ for $t := \lceil n^2/k^2 \rceil$ (using $(1 - 1/x)^x \leq 1/2$)
- More precise solution: $t := \lceil -1/\log_2(1 - (k/n)^2) \rceil$ (Appendix B.2 in notes).
  Ex: $n = 1000$, $k = 10$ gives $t \sim 6931.1$.
- In any case, $t = O(n^2)$.

Which Class Does the Algorithm Belong to?

To those required for $\mathcal{RP}$-problems!

Why?

See the definition of $\mathcal{RP}$ and $t = O(n^2)$.

Computationally Hard Problems

The Class $\mathcal{NP}$

Carsten Witt

Institut for Matematik og Computer Science
Danmarks Tekniske Universitet

Fall 2020

Overview

- Repeating and interpreting the definition
- Some examples
- Showing that a problem is in $\mathcal{NP}$
- Identifying the hard problems in $\mathcal{NP}$
  – the $\mathcal{NP}$-complete problems
- Methods for proving that a problem is hard

Definition

Definition

A yes-no–problem is in $\mathcal{NP}$ if there is a polynomial p and a randomized p-bounded algorithm A such that for every input $\boldsymbol{X}$ the following holds:
True answer for $\boldsymbol{X}$ is YES then $\boldsymbol{P}_R[\,A(\boldsymbol{X},R)=\mathrm{YES}\,]>0$
True answer for $\boldsymbol{X}$ is NO then $\boldsymbol{P}_R[\,A(\boldsymbol{X},R)=\mathrm{NO}\,]=1$
where $\boldsymbol{P}_R[\,Z\,]$ denotes the probability of event Z over uniform distribution of R,
$\|R\|\leq p(\|\boldsymbol{X}\|)$.

Toy Example

Problem [FINDPAIR]
Input: An array $X : [x_1, x_2, \ldots, x_n]$ of numbers.
Output: YES if there are at least two identical numbers, NO otherwise.

Algorithm:

do 3 times:
$i \leftarrow \text{rand}(1, n);\ j \leftarrow \text{rand}(1, n)$
if $(i \neq j) \wedge (X[i] = X[j])$ **then**
  return YES
end if
end do
return NO

Next: the same algorithm with random numbers as additional input

Toy Example

Input $X = [x_1, \ldots, x_n]$, $R = [r_1, \ldots, r_6]$

proc FINDPAIR(X, R)
$c \leftarrow 1$
do 3 times:
$i \leftarrow R[c]$; $c \leftarrow c + 1$
$j \leftarrow R[c]$; $c \leftarrow c + 1$
if $(1 \leq i \neq j \leq n) \wedge (X[i] = X[j])$ **then**
return YES
end if
end do
return NO

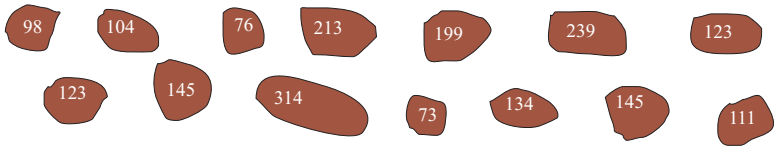
Input $X = [5, 3, 6, 3]$, $R = [1, 2, 3, 1, 2, 1]$ results in NO.

Motivation

- The intention is to show that the class $\mathcal{NP}$ contains problems that are hard to solve.
- This is justified by interpreting the definition as follows:
- "Even if we allow guessing, the problem can rarely be solved in polynomial time."
- Or "Even guessing does not help much to solve the problem."
- BUT: not all problems in $\mathcal{NP}$ are hard.
- This is because $\mathcal{NP}$ contains $\mathcal{P}$, the efficiently solvable problems.

Example: Potato Soup

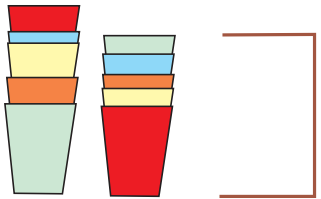
Problem [Potato Soup] A recipe calls for B grams of potatoes. You have a bag with n potatoes. Can one select some of them such that their weight is exactly B grams?



This problem is "hard to solve".

Example: Stacking Glasses

Problem [GLASSES IN A CUPBOARD] You have n glasses of equal height. If glass g_j is put into glass g_i let d_{ij} be the amount of g_j above the rim of g_i. You want to stack them into a single stack, so they fit into a cupboard of height h; is that possible?



This problem is hard to solve.

Example: TSP

Problem [Traveling Salesman (Traveling Salesperson)] A salesperson has customers in n different cities. He/she lives in city number 1. There is direct road between every pair of different cities and no possibility to change roads outside the cities. The distance between city i and city j is $d(i,j) \in \mathbb{N}$. The salesperson wants to visit all cities (in any order) and then return home without visiting another city twice. Is there such a tour whose total length is at most some given value B?

This problem is hard to solve.

Wikipedia's example:
the shortest roundtrip through Germany's 15 largest cities: there are 14! tours.

A good example?

Example: Road Maintenance

Problem [Road Maintenance] A person is responsible for maintaining roads in some area. There are n cities and roads between some of the cities. There are no possibilities to change roads outside the cities. The person wants to check all roads by starting and ending the tour in city 1. The person does not want to use a road twice but might visit a city more than once. Is such a tour possible?

This problem is easy to solve.

Example: Satisfiability

Problem [SATISFIABILITY]
Input: A set of clauses $C = \{c_1, \ldots, c_k\}$ over n boolean variables $x_1, \ldots, x_n$.
Output: YES if there is a satisfying assignment, i. e., if there is an assignment

$$a\colon \{x_1, \ldots, x_n\} \to \{0, 1\}$$

such that every clause c_j is satisfied, and NO otherwise.

This problem is hard to solve.

Proving a Problem in $\mathcal{NP}$

In order to prove that a problem P is in $\mathcal{NP}$, we have to:

1) Design a **deterministic** algorithm A which takes as input a problem instance $\boldsymbol{X}$ and a random sequence R. Especially:
   - 1a) Specify what the random sequence R consists of: bits, integers in some **finite** range, etc.
   - 1b) Specify how A interprets R as a guess.
   - 1c) Specify how A verifies the guess.
2) Show that the two conditions are met:
   - 2a) If the answer to $\boldsymbol{X}$ is YES, then there is a string R^* (randomly created with positive probability) such that $A(\boldsymbol{X}, R^*) = \text{YES}$.
   - 2b) If the answer to $\boldsymbol{X}$ is NO, then $A(\boldsymbol{X}, R) = \text{NO}$ for all R.
3) Show that A is p-bounded for some polynomial p.

Example: Potato Soup

Theorem

POTATO SOUP *is in* $\mathcal{NP}$.

Step 1) Describe the algorithm. First in words.

1a) The algorithm receives the real input $\boldsymbol{X}$, i. e., the potatoes with their weights and the target B, and the random string R which consist of bits: $R = r_1, r_2, \ldots, r_m$.

1b) If there are fewer bits than potatoes, answer NO.

1b) Consider the first n bits. If the i-th bit is 1, mark potato i.

1c) If the marked potatoes weigh B grams, answer YES, otherwise NO.

Example: Potato Soup

For those who prefer a formal description:

$$A\left(((w_1,\ldots,w_n),B),R\right) = \begin{cases} \text{YES}, & \text{if} \quad |R| \geq n \ \wedge \ \sum_{i=1}^{n} r_i w_i = B \\ \text{NO} & \text{otherwise} \end{cases}$$

Note: a description in words is enough and – when precisely formulated – the best one.

Example: Potato Soup

Let us do Step 3) next (running time).

- There are n potatoes.
- It is checked that there are at least n bits in R. Time: $O(n)$.
- Every potato is marked or not marked. Time: $O(n)$.
- The weights of the marked potatoes are added. Time: $O(n)$.
- The accumulated weight is compared with B and the answer is returned. Time $O(1)$.

Altogether the time is $O(n)$. Choosing a linear polynomial $p(n) = cn$ for some sufficiently large constant c suffices.

Example: Potato Soup

Step 2) the conditions are met.

- For 2a) assume that the answer is YES.
- Then there is a subset of the potatoes weighing exactly B g.
- Let $L \subseteq \{1, \ldots, n\}$ be the set of their indices.
- Construct a bit string $R^* = r_1 \cdots r_n$ by $r_i = 1$ iff $i \in L$.
- When A receives R^*, it will select exactly the potatoes in L, check their weight and answer YES.
- Altogether there is a string of length at most $p(n)$ that will give YES. The probability of randomly creating it is positive since it is drawn uniformly at random from a finite set. (The probability is at least 2^{-n}.)

Example: Potato Soup

- For 2b) assume that the answer is NO.
- Then no set of potatoes weighs B g.
- If R is too short, the algorithm will correctly answer NO.
- Otherwise it marks some potatoes and computes their weight.
- This is compared to B.
- As no set of potatoes weighs B, the answer is NO.

This completes the proof.

Example: TSP

Theorem

TRAVELING SALESMAN *is in* $\mathcal{NP}$.

(Step 1) Describe the algorithm.

1a) Let the string $R = r_1, r_2, \ldots, r_m$ consist of random integers in the range $1, \ldots, n$.

1b) If there are fewer than n integers, answer NO.
If the first n integers are not pairwise different, answer NO. Otherwise, consider the first n integers and interpret them to describe the order in which the cities are visited.

Example: TSP

1c) Compute the cost of that tour:

$$C = d(r_1, r_2) + \cdots + d(r_{n-1}, r_n) + d(r_n, r_1)$$

If $C \leq B$ then answer YES and NO otherwise.

Example: TSP

Step 2) the conditions are met.

- For 2a) assume that the answer is YES.
- Then there is a tour T_0 of length at most B.
- Let $z_1, \ldots, z_n$ be the order of the cities on that tour.
- Construct the sequence of integers (each from $\{1, \ldots, n\}$) $R^* = z_1 \cdots z_n$.
- When A receives R^*, it constructs exactly the tour T_0. As the length of it is at most B, the answer is YES.
- Altogether there is a string of length at most n that will give YES. The probability of randomly creating it is positive since it is drawn uniformly at random from a finite set. More precisely: the probability is at least $(1/n)^n$.

Example: TSP

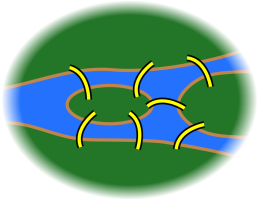
- For 2b) assume that the answer is NO.
- Then no tour has length B or less.
- If R is too short or contains a city twice, the algorithm will correctly answer NO.
- Otherwise it creates the tour specified by R.
- Its length is compared to B.
- As no tour of length $\leq B$ exists, the answer is NO.

The running time is $O(n)$. It is $O\big(n + n^2\big)$ if the reading of the input graph is also considered.

Example: Road Maintenance

This problem is solvable in polynomial time by a deterministic algorithm.

It is also know as the Euler cycle problem.



(http://en.wikipedia.org/wiki/Image:7_bridges.png)

RoadMaintenance is in class $\mathcal{P}$.

As $\mathcal{P} \subseteq \mathcal{NP}$, it is also in $\mathcal{NP}$.

On Problems Outside $\mathcal{NP}$

Is every decision problem in $\mathcal{NP}$? Most likely, NO!

Problem [UnSatisfiable]
Input: A set of clauses $C = \{c_1, \ldots, c_k\}$ over n boolean variables $x_1, \ldots, x_n$.
Output: YES if there is NO satisfying assignment, i. e., if for all assignments

$$a\colon \{x_1, \ldots, x_n\} \to \{0, 1\}$$

at least one clause c_j is NOT satisfied, and NO otherwise.

A single guess (random string) does not seem enough to verify a statement for all assignments.

Remember: $\mathcal{NP}$-problems typically demand that there is some object/property etc.