Nondeterministic Turing Machines

Remarks.

- The complexity class P consists of all decision problems that can be solved by a deterministic Turing machine in polynomial time.
- The complexity class NP consists of all decision problems that can be solved by a non-deterministic Turing machine in polynomial time.
  - Guess a solution and check in polynomial time.

Nondeterministic Turing Machines

Remarks.

- The complexity class P consists of all decision problems that can be solved by a deterministic Turing machine in polynomial time.
- The complexity class NP consists of all decision problems that can be solved by a non-deterministic Turing machine in polynomial time.
  - Guess a solution and check in polynomial time.
- NP stands for Non-deterministic Polynomial.

Nondeterministic Turing Machines

Remarks.

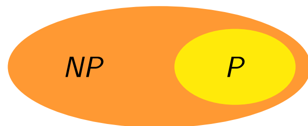
- The complexity class P consists of all decision problems that can be solved by a deterministic Turing machine in polynomial time.
- The complexity class NP consists of all decision problems that can be solved by a non-deterministic Turing machine in polynomial time.
  - Guess a solution and check in polynomial time.
- NP stands for Non-deterministic Polynomial.

Nondeterministic Turing Machines

Remarks.

- The complexity class P consists of all decision problems that can be solved by a deterministic Turing machine in polynomial time.
- The complexity class NP consists of all decision problems that can be solved by a non-deterministic Turing machine in polynomial time.
  - Guess a solution and check in polynomial time.
- NP stands for Non-deterministic Polynomial.

Lemma 8.2.
$P \subseteq NP$.



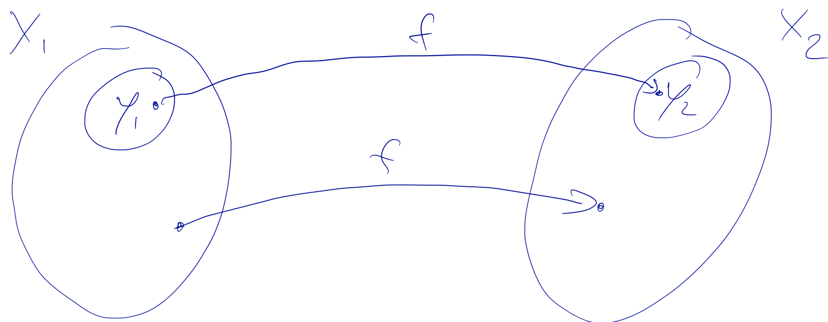
Proof: Deterministic Turing machines are a special case of non-deterministic Turing machines. □

Polynomial Transformations/Reductions

Definition 8.3.

Let $\mathcal{P}_1 = (X_1, Y_1)$ and $\mathcal{P}_2 = (X_2, Y_2)$ be decision problems. We say that $\mathcal{P}_1$ polynomially transforms to $\mathcal{P}_2$ if there exists a function $f : X_1 \to X_2$ computable in polynomial time such that for all $x \in X_1$

$$x \in Y_1 \iff f(x) \in Y_2 \ .$$



Polynomial Transformations/Reductions

Definition 8.3.

Let $\mathcal{P}_1 = (X_1, Y_1)$ and $\mathcal{P}_2 = (X_2, Y_2)$ be decision problems. We say that $\mathcal{P}_1$ polynomially transforms to $\mathcal{P}_2$ if there exists a function $f : X_1 \to X_2$ computable in polynomial time such that for all $x \in X_1$

$$x \in Y_1 \iff f(x) \in Y_2 \ .$$

Remarks.

- A polynomial transformation is also called Karp reduction.
- Polynomial transformations are transitive.

Polynomial Transformations/Reductions

Definition 8.3.

Let $\mathcal{P}_1 = (X_1, Y_1)$ and $\mathcal{P}_2 = (X_2, Y_2)$ be decision problems. We say that $\mathcal{P}_1$ polynomially transforms to $\mathcal{P}_2$ if there exists a function $f : X_1 \to X_2$ computable in polynomial time such that for all $x \in X_1$

$$x \in Y_1 \quad \Longleftrightarrow \quad f(x) \in Y_2 \ .$$

Remarks.

- A polynomial transformation is also called Karp reduction.
- Polynomial transformations are transitive.

Lemma 8.4.

Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be decision problems. If $\mathcal{P}_2 \in P$ and $\mathcal{P}_1$ polynomially transforms to $\mathcal{P}_2$, then $\mathcal{P}_1 \in P$.

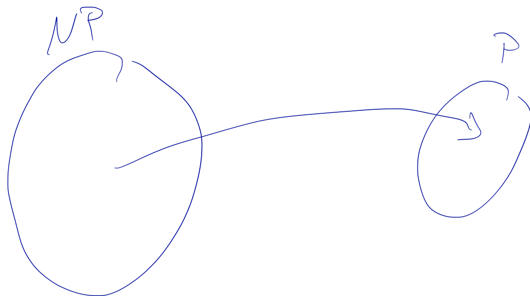
NP-Hardness and NP-Completeness

Definition 8.5.

Let $\mathcal{P}$ be an optimization or decision problem.

i $\mathcal{P}$ is NP-hard if all problems in NP polynomially transform to $\mathcal{P}$.

ii $\mathcal{P}$ is NP-complete if in addition $\mathcal{P} \in NP$.



NP-Hardness and NP-Completeness

Definition 8.5.

Let $\mathcal{P}$ be an optimization or decision problem.

i. $\mathcal{P}$ is NP-hard if all problems in NP polynomially transform to $\mathcal{P}$.
ii. $\mathcal{P}$ is NP-complete if in addition $\mathcal{P} \in NP$.

Satisfiability Problem (SAT)

Given: Boolean variables $x_1, \ldots, x_n$ and a family of clauses where each clause is a disjunction of Boolean variables or their negations.

Task: decide whether there is a truth assignment to $x_1, \ldots, x_n$ such that all clauses are satisfied.

NP-Hardness and *NP*-Completeness

Definition 8.5.

Let $\mathcal{P}$ be an optimization or decision problem.

i $\mathcal{P}$ is *NP*-hard if all problems in *NP* polynomially transform to $\mathcal{P}$.

ii $\mathcal{P}$ is *NP*-complete if in addition $\mathcal{P} \in NP$.

Satisfiability Problem (SAT)

Given: Boolean variables $x_1, \ldots, x_n$ and a family of clauses where each clause is a disjunction of Boolean variables or their negations.

Task: decide whether there is a truth assignment to $x_1, \ldots, x_n$ such that all clauses are satisfied.

Example: $(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2)$

Cook's Theorem (1971)

Theorem 8.6.

The Satisfiability problem is *NP*-complete.

Stephen Cook (1939–)

Cook's Theorem (1971)

Stephen Cook (1939–)

Theorem 8.6.

The Satisfiability problem is *NP*-complete.

Proof idea: SAT is obviously in *NP*. One can show that any non-deterministic Turing machine can be encoded as an instance of SAT. □

Proving NP-Completeness

Lemma 8.7.

Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be decision problems. If $\mathcal{P}_1$ is *NP*-complete, $\mathcal{P}_2 \in NP$, and $\mathcal{P}_1$ polynomially transforms to $\mathcal{P}_2$, then $\mathcal{P}_2$ is *NP*-complete.

Proof: As mentioned above, polynomial transformations are transitive. □

Proving NP-Completeness

Lemma 8.7.

Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be decision problems. If $\mathcal{P}_1$ is NP-complete, $\mathcal{P}_2 \in NP$, and $\mathcal{P}_1$ polynomially transforms to $\mathcal{P}_2$, then $\mathcal{P}_2$ is NP-complete.

Proof: As mentioned above, polynomial transformations are transitive. □

Integer Linear Programming Problem (ILP)

Given: matrix $A \in \mathbb{Z}^{m \times n}$, vector $b \in \mathbb{Z}^m$.

Task: decide whether there is $x \in \mathbb{Z}^n$ with $A \cdot x \geq b$.

Proving NP-Completeness

Lemma 8.7.

Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be decision problems. If $\mathcal{P}_1$ is NP-complete, $\mathcal{P}_2 \in NP$, and $\mathcal{P}_1$ polynomially transforms to $\mathcal{P}_2$, then $\mathcal{P}_2$ is NP-complete.

Proof: As mentioned above, polynomial transformations are transitive. □

Integer Linear Programming Problem (ILP)

Given: matrix $A \in \mathbb{Z}^{m \times n}$, vector $b \in \mathbb{Z}^m$.

Task: decide whether there is $x \in \mathbb{Z}^n$ with $A \cdot x \geq b$.

Theorem 8.8.

ILP is NP-complete.

Proof of Thm 8.8:

By reduction from SAT.

Given an instance of SAT with variables $x_1, \ldots, x_n$ construct an instance of ILP with variables $x_1, \ldots, x_n$

- Induce Bounds

$$0 \leq x_i \leq 1 \quad \forall i$$

- For each clause of SAT, say $(x_j \vee \neg x_k \vee x_l)$ introduce a constraint

$$x_j + (1 - x_k) + x_l \geq 1$$

Satisfying truth assignment
$\iff$ feasible integer solution

Transformations for Karp's 21 *NP*-Complete Problems

96

SATISFIABILITY

CLIQUE

0-1 INTEGER PROGRAMMING

SATISFIABILITY WITH AT MOST 3 LITERALS PER CLAUSE

NODE COVER

SET PACKING

CHROMATIC NUMBER

FEEDBACK NODE SET

FEEDBACK ARC SET

DIRECTED HAMILTON CIRCUIT

SET COVERING

EXACT COVER

CLIQUE COVER

UNDIRECTED HAMILTON CIRCUIT

3-DIMENSIONAL MATCHING

KNAPSACK

HITTING SET

STEINER TREE

SEQUENCING

PARTITION

MAX CUT

FIGURE 1 - Complete Problems

RICHARD M. KARP

Richard M. Karp (1972). *Reducibility Among Combinatorial Problems*

P vs. NP

Theorem 8.9.

If a decision problem $\mathcal{P}$ is NP-complete and $\mathcal{P} \in P$, then $P = NP$.

Proof: See definition of NP-completeness and Lemma 8.4. $\square$

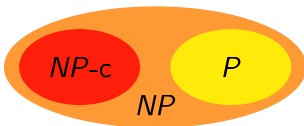
P vs. NP

Theorem 8.9.

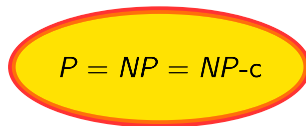
If a decision problem $\mathcal{P}$ is NP-complete and $\mathcal{P} \in P$, then $P = NP$.

Proof: See definition of NP-completeness and Lemma 8.4. □

There are two possible szenarios for the shape of the complexity world:



scenario A



scenario B

- It is widely believed that $P \neq NP$, i. e., scenario A holds.

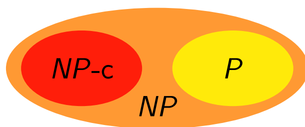
P vs. NP

Theorem 8.9.

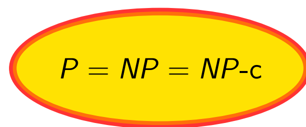
If a decision problem $\mathcal{P}$ is NP-complete and $\mathcal{P} \in P$, then $P = NP$.

Proof: See definition of NP-completeness and Lemma 8.4. □

There are two possible szenarios for the shape of the complexity world:



scenario A



scenario B

- It is widely believed that $P \neq NP$, i. e., scenario A holds.
- Deciding whether $P = NP$ or $P \neq NP$ is one of the seven millenium prize problems established by the Clay Mathematics Institute in 2000.

P vs NP Problem

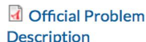
Suppose that you are organizing housing accommodations for a group of four hundred university students. Space is limited and only one hundred of the students will receive places in the dormitory. To complicate matters, the Dean has provided you with a list of pairs of incompatible students, and requested that no pair from this list appear in your final choice. This is an example of what computer scientists call an NP-problem, since it is easy to check if a given choice of one hundred students proposed by a coworker is satisfactory (i.e., no pair taken from your coworker's list also appears on the list from the Dean's office), however the task of generating such a list from scratch seems to be so hard as to be completely impractical. Indeed, the total number of ways of choosing one hundred students from the four hundred applicants is greater than the number of atoms in the known universe! Thus no future civilization could ever hope to build a supercomputer capable of solving the problem by brute force; that is, by checking every possible combination of 100 students. However, this apparent difficulty may only reflect the lack of ingenuity of your programmer. In fact, one of the outstanding problems in computer science is determining whether questions exist whose answer can be quickly checked, but which require an impossibly long time to solve by any direct procedure. Problems like the one listed above certainly seem to be of this kind, but so far no one has managed to prove that any of them really are so hard as they appear, i.e., that there really is no feasible way to generate an answer with the help of a computer. Stephen Cook and Leonid Levin formulated the P (i.e., easy to find) versus NP (i.e., easy to check) problem independently in 1971.

Image credit: on the left, Stephen Cook by Jiří Janíček (cropped). CC BY-SA 3.0

Rules:

Rules for the Millennium Prizes

Related Documents:



Official Problem Description

Minesweeper

Related Links:

Lecture by Vijaya Ramachandran

Complexity of Linear Programming

- As discussed in Chapter 4, so far no variant of the simplex method has been shown to have a polynomial running time.
- Therefore, the complexity of Linear Programming remained unresolved for a long time.

Complexity of Linear Programming

- As discussed in Chapter 4, so far no variant of the simplex method has been shown to have a polynomial running time.
- Therefore, the complexity of Linear Programming remained unresolved for a long time.
- Only in 1979, the Soviet mathematician Leonid Khachiyan proved that the so-called ellipsoid method earlier developed for nonlinear optimization can be modified in order to solve LPs in polynomial time.
- In November 1979, the New York Times featured Khachiyan and his algorithm in a front-page story.

Complexity of Linear Programming

- As discussed in Chapter 4, so far no variant of the simplex method has been shown to have a polynomial running time.
- Therefore, the complexity of Linear Programming remained unresolved for a long time.
- Only in 1979, the Soviet mathematician Leonid Khachiyan proved that the so-called ellipsoid method earlier developed for nonlinear optimization can be modified in order to solve LPs in polynomial time.
- In November 1979, the New York Times featured Khachiyan and his algorithm in a front-page story.
- Details can, e. g., be found in the book of Bertsimas & Tsitsiklis (Chapter 8) or in the book *Geometric Algorithms and Combinatorial Optimization* by Grötschel, Lovász & Schrijver (Springer, 1988).

An Approach to Difficult Problems

Mathematicians disagree as to the ultimate practical value of Leonid Khachiyan's new technique, but concur that in any case it is an important theoretical accomplishment.

Mr. Khachiyan's method is believed to offer an approach for the linear programming of computers to solve so-called "traveling salesman" problems. Such problems are among the most intractable in mathematics. They involve, for instance, finding the shortest route by which a salesman could visit a number of cities without his path touching the same city twice.

Each time a new city is added to the route, the problem becomes very much more complex. Very large numbers of variables must be calculated from large numbers of equations using a system of linear programming. At a certain point, the compexity becomes so great that a computer would require billions of years to find a solution.

In the past, "traveling salesmen" problems, including the efficient scheduling of airline crews or hospital nursing staffs, have been solved on computers using the "simplex method" invented by George B. Dantzig of Stanford University.

As a rule, the simplex method works well, but it offers no guarantee that after a certain number of computer steps it will always find an answer. Mr. Khachiyan's approach offers a way of telling right from the start whether or not a problem will be soluble in a given number of steps.

Two mathematicians conducting research at Stanford already have applied the Khachiyan method to develop a program for a pocket calculator, which has solved problems that would not have been possible with a pocket calculator using the simplex method.

Mathematically, the Khachiyan approach uses equations to create imaginary ellipsoids that encapsulate the answer, unlike the simplex method, in which the answer is represented by the intersections of the sides of polyhedrons. As the ellipsoids are made smaller and smaller, the answer is known with greater precision. MALCOLM W. BROWNE

New York Times, Nov. 27, 1979

Thank You!