Computational Complexity

计算复杂性

张方国

中山大学计算机学院

isszhfg@mail.sysu.edu.cn

Lecture 4. The P versus NP Question

- The search version: finding versus checking($\mathcal{PF}$ vs $\mathcal{PC}$)
- The decision version: proving versus verifying ($\mathcal{P}$ vs $\mathcal{NP}$)
- Equivalence of the two formulations
- The traditional definition of NP

The Clay Mathematics Institute Millenium Prize Problems

- Birch and Swinnerton-Dyer Conjecture
- Hodge Conjecture
- Navier-Stokes Equations
- P vs NP
- Poincaré Conjecture
- Riemann Hypothesis
- Yang-Mills Theory

The P versus NP problem is perhaps one of the biggest open problems in computer science (and mathematics!) today.

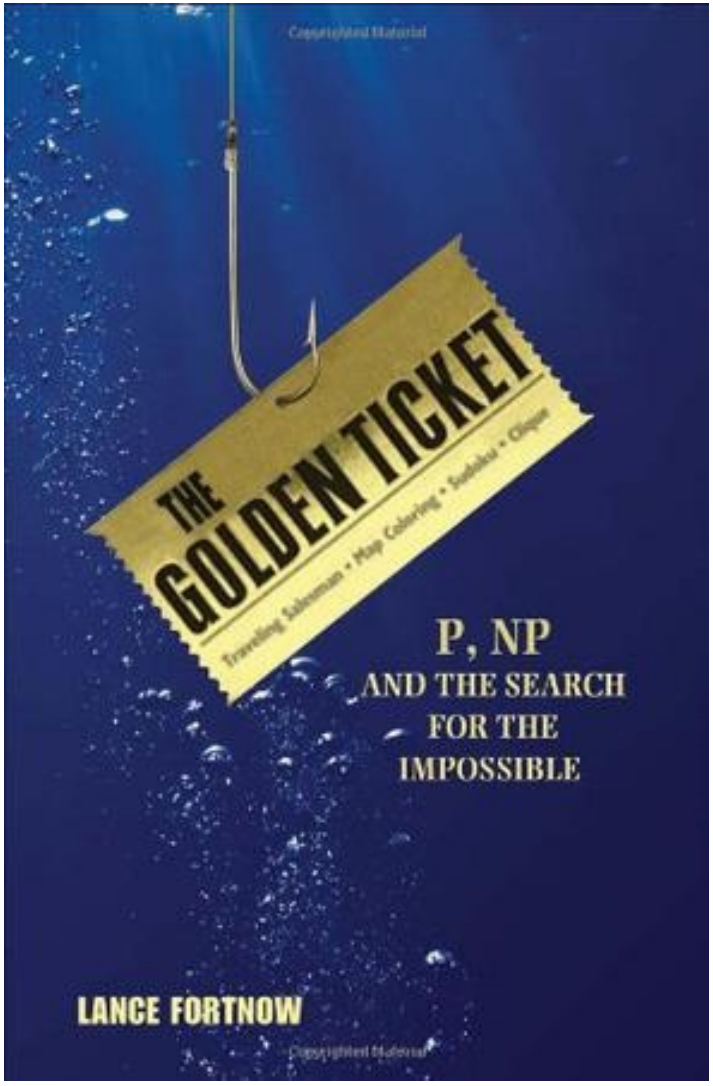
每个问题100万美金，如果能证明P=NP，其他问题也可以被解决，可得600万美金，这就是P和NP的奇妙!

Our experience: it is harder to solve a problem than it is to check the correctness of a solution (or proving a theorem is much more difficult than checking the proof of a theorem).

Is this experience merely a coincidence or does it represent a fundamental fact of life?

This is the essence of the P versus NP Question, where **P represents search problems that are efficiently solvable** and **NP represents search problems for which solutions can be efficiently checked**.

Another natural question captured by the P versus NP Question is whether proving theorems is harder that verifying the validity of these proofs. In this case, **P represents decision problems that are efficiently solvable**, whereas **NP represents sets that have efficiently checkable proofs of membership**.


THE
GOLDEN TICKET
P, NP
AND THE SEARCH
FOR THE
IMPOSSIBLE
LANCE FORTNOW

The search version: finding versus checking

Definition

(Solving a **search problem**): Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ and $R(x) \stackrel{\text{def}}{=} \{y : (x,y) \in R\}$ denote the set of solutions for the instance x. A function $f : \{0,1\}^* \to \{0,1\}^* \cup \{\perp\}$ *solves the search problem of* R if for every x the following holds: if $R(x) \neq \emptyset$ then $f(x) \in R(x)$ and otherwise $f(x) = \perp$.

We consider only search problems in which the length of the solution is bounded by a polynomial in the length of the instance. Recalling that search problems are associated with binary relations, we focus our attention on polynomially bounded relations:

Definition

(polynomially bounded relations): We say that $R \subseteq \{0,1\}^* \times \{0,1\}^*$ is polynomially bounded if there exists a polynomial p such that for every $(x,y) \in R$ it holds that $|y| \leq p(|x|)$.

Note: For a polynomially bounded relation R it makes sense to ask whether or not, given a problem instance x, one can efficiently find an adequate solution y. The polynomial bound on the length of the solution(i.e., y) guarantees that a negative answer is not merely due to the length of the required solution.

在polynomially bounded relations限定下，我们定义search problems的两类新问题：$\mathcal{PF}$和$\mathcal{PC}$ 和decision problems的两类新问题：$\mathcal{P}$和$\mathcal{NP}$

The class P as a natural class of search problems

Restricting our attention to polynomially bounded relations, we identify the corresponding fundamental class of search problem (or binary relation) denoted $\mathcal{PF}$ ("Polynomial-time Find"):

efficiently solvable search problems

The search problem of a polynomially bounded relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ is efficiently solvable if there exists a polynomial time algorithm A such that for every x it holds that $A(x) \in R(x) \stackrel{\text{def}}{=} \{y : (x,y) \in R\}$ if and only if $R(x)$ is not empty. Furthermore, if $R(x) = \emptyset$ then $A(x) = \perp$, indicating that x has no solution.

We denote by $\mathcal{PF}$ the class of search problems that are efficiently solvable (and correspond to polynomially bounded relations). That is, $R \in \mathcal{PF}$ if R is polynomially bounded and there exists a polynomial time algorithm that given x finds y such that $(x, y) \in R$ (or asserts that no such y exists).

Note: The algorithm A always outputs a correct answer, which is a valid solution in the case that such a solution exists and otherwise provides an indication that no solution exists.

The class NP as another natural class of search problems

Natural search problems have the property that valid solutions can be efficiently recognized. That is, given an instance x of the problem R and a candidate solution y, one can efficiently determine whether or not y is a valid solution for x with respect to the problem R.

search problems with efficiently checkable solutions

The search problem of a polynomially bounded relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ has efficiently checkable solutions if there exists a polynomial time algorithm A such that for every x and y, it holds that $A(x,y) = 1$ if and only if $(x,y) \in R$.

We denote by $\mathcal{PC}$ (standing for "Polynomial-time Check") the class of search problems that correspond to polynomially-bounded binary relations that have efficiently checkable solutions. That is, $R \in \mathcal{PC}$ if the following two conditions hold:

1. For some polynomial p, if $(x, y) \in R$ then $|y| \leq p(|x|)$.
2. There exists a polynomial-time algorithm that given (x, y) determines whether or not $(x, y) \in R$.

The P versus NP question in terms of search problems

The class $\mathcal{PC}$ is the natural domain for the study of which problems are in $\mathcal{PF}$, because the ability to efficiently recognize a valid solution is a natural prerequisite for a discussion regarding the complexity of finding such solutions. We warn, however, that $\mathcal{PF}$ contains (unnatural) problems that are not in $\mathcal{PC}$.

Is it the case that every search problem in $\mathcal{PC}$ is in $\mathcal{PF}$?

If $\mathcal{PC} \subseteq \mathcal{PF}$, then this would mean that whenever solutions to given instances can be efficiently checked (for correctness) it is also the case that such solutions can be efficiently found (when given only the instance). This would mean that all reasonable search problems (i.e., all problems in $\mathcal{PC}$) are easy to solve. Such a situation would contradict the intuitive feeling that some reasonable search problems are hard to solve. Furthermore, in such a case, the notion of "solving a problem" will lose its meaning (because finding a solution will not be significantly more difficult than checking its validity).

On the other hand, if $\mathcal{PC} \setminus \mathcal{PF}$ then there exist reasonable search problems (i.e., some problems in $\mathcal{PC}$) that are hard to solve. This conforms with our basic intuition by which some reasonable problems are easy to solve whereas others are hard to solve. Furthermore, it reconfirms the intuitive gap between the notions of solving and checking (asserting that in some cases "solving" is significantly harder than "checking").

The decision version: proving versus verifying

NP问题的定义大都是考虑decision问题的。

The study of search problems can be "reduced" to the study of decision problems(一个对搜索型问题的算法自然的也是对判定型问题的算法). or the decision problems are almost the same (asymptotically) as their search counterparts.

The decision problems are interesting and natural per se. some people do care about the truth.

Specifically, determining whether a given object has some predetermined property constitutes an appealing computational problem.

Decision vs Search:

- Clearly, solving "search problem" efficiently means "decision problem" can be solved efficiently. – Just run the algorithm for the search problem and modify the output suitably.
- So if "decision" version is difficult then "search" version is definitely difficult!
- In some cases, we can use an algorithm that solves the decision problem to solve the search problem.

Sub-sum problem

The class P as a natural class of decision problems

Definition

(Solving a decision problem): Let $S \subseteq \{0,1\}^*$. A function $f : \{0,1\}^* \to \{0,1\}$ *solves the decision problem of S (or decides membership in S)* if for every x it holds that $f(x) = 1$ if and only if $x \in S$.

Efficiently solvable decision problems:

- A decision problem $S \subseteq \{0,1\}^*$ is efficiently solvable if there exists a polynomial time algorithm A such that, for every x it holds that $A(x) = 1$ if and only if $x \in S$.
- We denote by $\mathcal{P}$ the class of decision problems that are efficiently solvable.

The class NP and NP-proof Systems

We view NP as the class of decision problems that have efficiently verifiable proof systems. Loosely speaking, we say that a set S has a proof system if instances in S have valid proofs of membership (i.e., proofs accepted as valid by the system), whereas instances not in S have no valid proofs. Indeed, proofs are defined as strings that (when accompanying the instance) are accepted by the (efficient) verification procedure. We say that V is a verification procedure for membership in S if it satisfies the following two conditions:

1. Completeness: True assertions have valid proofs;
2. Soundness: False assertions have no valid proofs.

Our focus here is on **efficient verification procedures** that utilize relatively short proofs.

efficiently verifiable proof systems

A decision problem $S \subseteq \{0,1\}^*$ has an efficiently verifiable proof system if there exists a polynomial p and a polynomial-time (verification) algorithm V such that the following two conditions hold:

1. Completeness: For every $x \in S$, there exists y of length at most $p(|x|)$ such that $V(x,y) = 1$.
   (Such a string y is called an **NP-witness** for $x \in S$.)
2. Soundness: For every $x \notin S$ and every y, it holds that $V(x,y) = 0$.

Thus, $x \in S$ if and only if there exists y of length at most $p(|x|)$ such that $V(x, y) = 1$.

In such a case, we say that S has an **NP-proof system**, and refer to V as its **verification procedure** (or as the proof system itself).

We denote by $\mathcal{NP}$ the class of decision problems that have efficiently verifiable proof systems.

In some cases, V(or the set of pairs accepted by V) is called a witness relation of S.

Using this definition, it is typically easy to show that natural decision problems are in $\mathcal{NP}$.

Observe that $\mathcal{P} \subseteq \mathcal{NP}$ holds: A verification procedure for claims of membership in a set $S \in \mathcal{P}$ may just ignore the alleged NP-witness and run the decision procedure that is guaranteed by the hypothesis $S \in \mathcal{P}$; that is, $V(x, y) = A(x)$, where A is the aforementioned decision procedure.

Note that for any search problem R in $\mathcal{PC}$, the set of instances that have a solution with respect to R(i.e., the set $S_R \stackrel{\text{def}}{=} \{x : R(x) \neq \emptyset\}$) is in $\mathcal{NP}$. Specifically, for any $R \in \mathcal{PC}$, consider the verification procedure V such that $V(x, y) \stackrel{\text{def}}{=} 1$ if and only if $(x, y) \in R$, and note that the latter condition can be decided in poly($|x|$)-time.

Thus, any search problem in $\mathcal{PC}$ can be viewed as a problem of searching for (efficiently verifiable) proofs (i.e., NP-witnesses for membership in the set of instances having solutions).

On the other hand, any NP-proof system gives rise to a natural search problem in $\mathcal{PC}$, that is, the problem of searching for a valid proof (i.e., an NP-witness) for the given instance(i.e., the verification procedure V yields the search problem that corresponds to $R = \{(x, y) : V(x, y) = 1\}$). Thus, $S \in \mathcal{NP}$ if and only if there exists $R \in \mathcal{PC}$ such that $S = \{x : R(x) \neq \emptyset\}$.

Equivalence of the two formulations

The two formulations of the P-vs-NP Questions are equivalent. That is, every search problem having efficiently checkable solutions is solvable in polynomial time (i.e., $\mathcal{PC} \subseteq \mathcal{PF}$) if and only if membership in any set that has an NP-proof system can be decided in polynomial time (i.e., $\mathcal{NP} \subseteq \mathcal{P}$).

Theorem

$\mathcal{PC} \subseteq \mathcal{PF}$ if and only if $\mathcal{P} = \mathcal{NP}$.

Proof: Suppose $\mathcal{PC} \subseteq \mathcal{PF}$.We will show that this implies the existence of an efficient algorithm for finding NP-witnesses for any set in $\mathcal{NP}$, which in turn implies that this set is in $\mathcal{P}$.

Specifically, let S be an arbitrary set in $\mathcal{NP}$, and V be the corresponding verification procedure. Then $R \stackrel{\text{def}}{=} \{(x,y) : V(x,y) = 1\})$ is a polynomially bounded relation in $\mathcal{PC}$, and by the hypothesis its search problem is solvable in polynomial time (i.e., $R \in \mathcal{PC} \subseteq \mathcal{PF}$).

Denoting by A the polynomial-time algorithm solving the search problem of R, we decide membership in S in the obvious way. That is, on input x, we output 1 if and only if $A(x) \neq \perp$, where the latter event holds if and only if $A(x) \in R(x)$, which in turn occurs if and only if $R(x) \neq \emptyset$. Thus,$\mathcal{NP} \subseteq \mathcal{P}$(and $\mathcal{P} = \mathcal{NP}$) follow.

Suppose, on the other hand, that $\mathcal{NP} = \mathcal{P}$. We will show that this implies an efficient algorithm for determining whether a given string y' is a prefix of some solution to a given instance x of a search problem in $\mathcal{PC}$, which in turn yields an efficient algorithm for finding solutions. Specifically, let R be an arbitrary search problem in $\mathcal{PC}$. Then the set $S'_R \stackrel{\text{def}}{=} \{\langle x, y' \rangle : \exists y'' \text{ s.t. } (x, y'y'') \in R\}$ is in $\mathcal{NP}$ (because $R \in \mathcal{PC}$), and hence S'_R is in $\mathcal{P}$ (by the hypothesis $\mathcal{NP} = \mathcal{P}$). This yields a polynomial-time algorithm for solving the search problem of R, by extending a prefix of a potential solution bit-by-bit (while using the decision procedure to determine whether or not the current prefix is valid). That is, on input x, we first check whether or not $(x, \lambda) \in S'_R$ and output $\perp$ (indicating $R(x) = \emptyset$) in case $(x, \lambda) \notin S'_R$. Next, we proceed in iterations, maintaining the invariant that $(x, y') \in S'_R$. In each iteration, we set $y' \leftarrow y'0$ if $(x, y'0) \in S'_R$ and $y' \leftarrow y'1$ if $(x, y'1) \in S'_R$. If none of these conditions hold (which happens after at most polynomially many iterations) then the current y' satisfies $(x, y') \in R$. Thus, for an arbitrary $R \in \mathcal{PC}$ we obtain that $R \in \mathcal{PF}$, and $\mathcal{PC} \subseteq \mathcal{PF}$ follows. ■

$\mathcal{PC} \subseteq \mathcal{PF}$ if and only if $\mathcal{P} = \mathcal{NP}$

The traditional definition of NP

non-deterministic polynomial-time Turing machines:

A non-deterministic Turing machine is defined as TM except that the transition function maps symbol-state pairs to subsets of triples (rather than to a single triple) in $\Sigma \times Q \times \{-1, 0, +1\}$. The transition function of NTM is a mapping:

$$\Sigma \times Q \to 2^{\Sigma \times Q \times \{-1,0,+1\}}.$$

Accordingly, the configuration following a specific instantaneous configuration may be one of several possibilities, each determine by a different possible triple. Thus, the computations of a non-deterministic machine on a fixed input may result in different outputs.

In the context of decision problems one typically considers the question of whether or not there exists a computation that starting with a fixed input halts with output 1. We say that the **non-deterministic machine** M **accept** x if there exists a computation of M, on input x that halts with output 1. The set accepted by a non-deterministic machine is the set of inputs that are accepted by the machine.

A non-deterministic polynomial-time Turing machine is defined as one that makes a number of steps that is polynomial in the length of the input. Traditionally, $\mathcal{NP}$ is defined as the class of sets that are accepted by some non-deterministic polynomial-time Turing machine.

Theorem

A set S has an NP-proof system if and only if there exists a non-deterministic polynomial-time machine that accepts S.

The two definitions of **NP** as the class of problems solvable by a nondeterministic Turing machine (TM) in polynomial time and the class of problems verifiable by a deterministic Turing machine in polynomial time are equivalent.

To show this, first suppose we have a deterministic verifier. A nondeterministic machine can simply nondeterministically run the verifier on all possible proof strings (this requires only polynomially-many steps because it can nondeterministically choose the next character in the proof string in each step, and the length of the proof string must be polynomially bounded).

If any proof is valid, some path will accept; if no proof is valid, the string is not in the language and it will reject.

Conversely, suppose we have a nondeterministic TM called A accepting a given language L. At each of its polynomially-many steps, the machine's computation tree branches in at most a constant number of directions. There must be at least one accepting path, and the string describing this path is the proof supplied to the verifier. The verifier can then deterministically simulate A, following only the accepting path, and verifying that it accepts at the end. If A rejects the input, there is no accepting path, and the verifier will never accept.

Proof Sketch: Suppose, on one hand, that the set S has an NP-proof system, and let us denote the corresponding verification procedure by V. Consider the following non-deterministic polynomial-time machine, denoted M. On input x, machine M makes an adequate $m = \mathrm{poly}(|x|)$ number of non-deterministic steps, producing (non-deterministically) a string $y \in \{0,1\}^m$, and then emulates $V(x,y)$. We stress that these non-deterministic steps may result in producing any m-bit string y. Recall that $x \in S$ if and only if there exists y of length at most $\mathrm{poly}(|x|)$ such that $V(x,y) = 1$. This implies that the set accepted by M equals S.

Suppose, on the other hand, that there exists a non-deterministic polynomial-time machine M that accepts the set S. Consider a deterministic machine M' that on input (x,y), where y has adequate length, emulates a computation of M on input x while using y to determine the non-deterministic steps of M. That is, the i^{th} step of M on input x is determined by the i^{th} bit of y (which indicates which of the two possible moves to make at the current step). Note that $x \in S$ if and only if there exists y of length at most $\mathrm{poly}(|x|)$ such that $M'(x,y) = 1$. Thus, M' gives rise to an NP-proof system for S. □

Thank You Very Much!