Syntactic solution to P v.s. NP problem

Hideto Ishihara

November 18, 2016

Department of Mathematics, Graduate School of Science, Kyoto University
e-mail: h.ishihara26@gmail.com

Abstract
Let the class P be the set of problems that are solved in polynomial
time and the class NP the set of problems that are checked in polynomial
time. We show that the class NP is not equal to the class P. The problem
is well-known as P v.s. NP problem. Our proof is a new approach. The
thesis that understanding is the same as classification is used.

Keywords: P v.s. NP problem

1 Introduction

For Turing Machine (see [8]) the class P is defined as the set of problems that are
solved in polynomial time and the class NP is defined as the set of problems that
are checked in polynomial time (see [2]). The class P or NP is also defined with
the use of deterministic or nondeterministic Turing Machines. The following is
well-known as P v.s. NP problem (see e.g. [1]).

Theorem 1. The class NP is not equal to the class P.

A P-(resp. NP-)problem is a computational problem in the class P (resp.
NP). We construct a sequence of configurations of a nondeterministic Turing
Machine solving an NP-problem that is not a sequence of configurations of a
deterministic Turing Machine solving a P-problem and then prove the class NP
is not equal to the class P.

The proof proceeds as follows. First as mentioned above we shall construct
a sequence of configurations of a nondeterministic Turing Machines solving an
NP-problem.

Let \( \$ \) be not in the input alphabet. First take a deterministic algorithm A
solving a P-problem with all inputs are solutions. Assume \( \$ \) does not appear in
the algorithm A. We construct an algorithm \( P_A \) from A in which we obtain \( x\$x \)
from the input \( x \) and then perform A to the latter \( x \) to obtain \( x\$1 \). We construct
a nondeterministic algorithm $NP_A$ in which we perform $P_A$ to the input $x$ and then obtain $x$ or 1 (or possibly both) from $x1$. We say that two sets are not separated by deterministic Turing Machines solving P-problems if they are not separated by any deterministic Turing Machines solving P-problems. If a nondeterministic transition function solving an NP-problem has sets of outputs not separated by deterministic Turing Machines solving P-problems there exists an NP-problem which is not solved by a deterministic transition function and we are done. Select at least in two ways $x$ or 1 (or possibly both) at the branches so that the resulting sets are not separated by deterministic Turing Machines solving P-problems. This is possible because $\mathbb{R}$ is uncountable. There thus exists a nondeterministic algorithm solving an NP-problem not separated by deterministic Turing Machines solving P-problems. The NP-problem of the corresponding nondeterministic transition function is a desired one.

From above there exists an NP-problem that is not solved by a deterministic Turing Machine in polynomial time. Hence we conclude that the class NP is not equal to the class P. The assertion follows.

2 Proof of Theorem

We prove the main theorem. Turing Machine is defined in [8], Chapter 3, Section 3.1, definition 3.3. We cite the definitions of the class P and NP (see [2], Chapter 2, Section 2.3.1, definition 2.4 and Section 2.6, definition 2.7).

Definition 2. We denote by P the class of decision problems that are efficiently solvable and by NP the class of sets that are each accepted by some nondeterministic polynomial time Turing Machine.

A P-(resp. NP-)problem is a computational problem in the class P (resp. NP).

Plan. First we shall construct a sequence of configurations of a nondeterministic Turing Machine solving an NP-problem that is not a sequence of configurations of a deterministic Turing Machine solving a P-problem.

Let $\sharp$ be not in the input alphabet. Take a deterministic algorithm $A$ solving a P-problem with all inputs being solutions. Assume $\sharp$ does not appear in any configurations in the algorithm $A$.

Lemma 3. There exists an algorithm $P_A$ from $A$ in which we obtain $x\sharp x$ from the input $x$ and then perform $A$ to the latter $x$ to obtain $x1$.

Proof. Copy the input $x$, move the head to the right of $x$ and write $\sharp$ and moving right write $x$. Move the head to the right of $\sharp$ and perform $A$ to the latter $x$. Observe that such an algorithm exists for each one solving a P-problem. The assertion follows. \qed
**Lemma 4.** There exists a nondeterministic algorithm $NP_A$ in which we perform $P_A$ to the input $x$ and then obtain $x$ or 1 (or possibly both) from $x\#1$.

**Proof.** For the input $x$ perform the algorithm $P_A$ to obtain $x\#1$. Delete $\#1$ or $x\#$ to obtain $x$ or 1 (or possibly both). The resulting algorithm is a nondeterministic algorithm (distinct difference (see Remark 7)). The assertion follows. □

**Plan.** We shall prove that there exists an NP-problem that is not solved by a deterministic algorithm of finite description in polynomial time.

Two sets are not separated by deterministic Turing Machines solving P-problems if they are not separated by any deterministic Turing Machines solving P-problems.

If a nondeterministic transition function solving an NP-problem has sets of outputs not separated by deterministic Turing Machines solving P-problems there exists an NP-problem solved by a branch of the nondeterministic algorithm which is not solved by any deterministic transition function and we are done. We note that a deterministic algorithm solving one P-problem constructed from a branch of the NP-problem, if it exists, solves the others and that the conclusions of the P-problems corresponding to any branches of the nondeterministic algorithm are the same because they are not separated by deterministic Turing Machines solving P-problems.

**Lemma 5.** There exist at least two sets of outputs of $NP_A$ not separated by deterministic Turing Machines solving P-problems.

**Proof.** Select at least in two ways $x$ or 1 (or possibly both) at the branches so that the resulting sets are not separated by deterministic Turing Machines solving P-problems. This is possible because the set of P-problems is countable and $\mathbb{R}$ is uncountable. The assertion follows. □

Thus there exists a nondeterministic algorithm solving an NP-problem not separated by deterministic Turing Machines solving P-problems with cardinality $> 1$. The NP-problem solved by the corresponding nondeterministic transition function is a desired one.

**Remark.** The above argument does not work if we replace P and NP with EXP and NP. (It is shown that trying to choose a deterministic algorithm which solves such a branch results in a nondeterministic algorithm of the same computational complexity.)

**Remark 6.** Here we used the distinct difference (see Remark 7).

**Proof of Theorem 1.** From above there exists an NP-problem that is not solved by a deterministic Turing Machine in polynomial time. Hence we conclude that the class NP is not equal to the class P. This shows the assertion. □
Remark 7. In the proof we used the distinct difference: the difference between the class P and the class NP is that it is defined by deterministic algorithms or by nondeterministic algorithms.

Remark 8. The above solution and proof of P v.s. NP problem implies that adding some necessary conditions may drastically ease the proof of a proposition. Hence a hard problem may become an undergraduate level problem as time goes by, especially just after it has been solved.

References

claymath.org/sites/default/files/pvnp.pdf [Accessed: 19th July 2016]


https://sites.google.com/site/cabstruction [Accessed: 18th November 2016]


[6] Ochanomizu University, Heisei 25 nendo zenkoku gakuryoku chousa (kimokomakai chousa) no kedka wo katsuyou shita gakuryoku ni eikyou wo ataeru youm bunseki ni kansuru chousa kenkyuu (A study of analysis of factors for academic performance with the use of National academic ability survey 2013 (fine survey)), 2014
