Title: proof of p=np

author: Hyeon-ho song

Contents

Abstract

This paper proves that p=np. This paper consists of a basic frame and examples. The pnp problem is related to the characteristics of the problem, and an attempt was made to use this to make the problem contradictory and solve it through the reduction method.

2020year MSC number: 68R99 None of the above, but in this section, keyword: p=np

한국어 초록

이 논문은 p=np-완전이라는 것을 불완전성 정리와 유사한 방식으로 증명한다.

content summary

The proof is as follows.

1. Establish the mathematical expression (∃x)Dem(x,sub(y,a,y)) or ~(∃x)Dem(x,sub(y,a,y)) using Dem and sub.

2. This equation is called the Gödel problem m.

3.Change y to m in sub.

4. This equation is called Gödel problem g.

5.sub(m,a,m) is g.

6.g is solved in polynomial time in a deterministic Turing machine.

7. Therefore, the precondition np-complete is not p is an incorrect precondition.

8. For this proof, use np-complete problems such as the independent set problem.

9. The proof when substituting the independent set problem also shows that p=np-complete is the correct prerequisite.

Summary of other contents

The academic significance of this problem is that it can increase the discussion on np-complete beyond now. The limitations in this proof process were successfully overcome through rigorous proof. At first, I calculated with prejudice, but eventually I broke through my prejudice and made calculations.

Introduction and Research Methods

You may already know this, but I will briefly explain the p np problem.

The p np problem is about the relationship between p and np problems and is a very important problem in mathematics and computer science.

The question is whether p=np-complete or not, and I will explain it in more detail. The p problem is a problem that can be solved in polynomial time on a deterministic Turing machine. On the other hand, the np problem is solved in polynomial time on a nondeterministic Turing machine. The problem is that in this case, the np problem takes too long to be solved. The np problem has a large time complexity, so it must ultimately be solved in a deterministic way with relatively low time complexity. In other words, is there an algorithm that can solve this np problem within a critical time? The additional things you need to know here are np-complete and np-hard. Here, whether np-complete is equal to p or not is the main issue of this problem. As described above, the pnp problem has a very important meaning in defining the characteristics of the problem. The PNP problem has not been solved since it was presented in 1971. However, it is a problem that does not have a long history compared to the Riemann hypothesis, which applies to both Hilbert's 23 problems and the Millennium Problem, and Fermat's Last Theorem solved by Andrew Wiles. Of course, the expectation that it will be solved in the 21st century is decreasing, but the figure is quite high compared to other problems. Many scholars think that p=np. This is because we have not yet found a case where the np problem is solved conclusively. I knew that it would be difficult to come up with an answer to the problem through such a formulaic solution, so I took a more direct approach. But I also had a problem. If solved in this way, the np-difficulty becomes completely the same as the p problem. Therefore, I decided to revisit the definition of np-difficulty. And when I completed the first solution, a problem arose. In order to make the expression true, all you have to do is modify it by removing the tilde, so you also had to look for contradictions. I thought about this

again and eventually found the contradiction in the revised formula and referenced it in the solution. The reason I solved the problem this way is because I was impressed by the proof of the incompleteness principle in the past. I looked at this proof, modified it, found the contradiction in the characteristics of the problem, and tried to solve it using the reductio method. At first, I naturally thought that np was not p. But when you solve the problem, p=np. At first, I was quite confused and reexamined the problem. However, since there were no parts that were inaccurate or contradictory, we decided to accept it as is. But still, considering possible mistakes, I decided to use the independent set problem as an example and solved the problem again. I solved the problem like this, but I wasn't immediately convinced, so I reviewed it several times. What I was particularly worried about was the definition of Dem and sub. I kept changing the definitions of Dem and sub. Also, the abbreviations used in the explanation were used incorrectly. I have fixed problems ranging from trivial and small to very critical problems. In addition, any explanation that may be problematic in the solution was provided with a reference and supplementary explanation.

Main text

Before we get into it, let's start with an explanation of the terms.

Dem and sub used in this paper are similar to those defined by Gödel, but are not mathematically the same.

Dem(x,y):= The problem of whether there is an algorithm x in which the Gödel problem of y is solved in polynomial time in a decisive way[1]

sub(m,a,n) is defined as the Gödel problem when the part corresponding to the Gödel problem a is changed to n in the Gödel problem m.

We define the Gödel problem. The Gödel problem consists of primes, like the Gödel number. The Gödel problem has a corresponding mathematical problem.

suppose np-complete is not p

~(∃x)Dem(x,sub(y,a,y))= Gödel problem m sub(y,a,y)=np-complete, G:~(∃x)Dem(x,sub(m,a,m)) problem ~(∃x)Dem(x,sub(m,a,m))is p problem. [cf. We provide a non-deterministic algorithm to check if a problem is np, and by comparing the problem to a 3-SAT problem , It is possible to know whether there is an algorithm that solves the problem decisively in polynomial time.]

definition of sub(m,a,m): Change the part corresponding to a in m to m. In ~(∃x)Dem(x,sub(y,a,y)), the part corresponding to a was changed to m. ∴sub(m,a,m)=~(∃x)Dem(x,sub(m,a,m))

~(∃x)Dem(x,sub(m,a,m)) is a p problem, but ~(∃x)Dem(x,sub(~(∃x)Dem(x,sub(m,a,m)))) and it is not a P problem. Modify the prerequisites by Proof by contradiction.

[cf. Strictly speaking, if problem G is changed to (∃x)Dem(x,sub(m,a,m)) to make problem G true, It has contradictory answers in that it has algorithms to solve the wrong answer problem ~(∃ x)Dem(x,sub(m,a,m)).]

Therefore, the precondition that np is not p must be corrected.

assume np complete is p problem

(∃x)Dem(x,sub(y,a,y))=m sub(y,a,y)=np-complete

G:(∃x)Dem(x,sub(m,a,m))

problem ~(∃x)Dem(x,sub(m,a,m))is p problem [cf.We provide a non-deterministic algorithm to check if a problem is np, and by comparing the problem to a 3-SAT problem, we can check in polynomial time if the problem is np-complete.]

sub(m,a,m)=(∃x)Dem(x,sub(m,a,m))

When (∃x)Dem(x,sub((∃x)Dem(x,sub(m,a,m)))), the proposition is non-contradictory. ∴p=np-complete

In addition, np-hard problems that belong to the complement of the set of NP-hard∶=p problems are not solved non-deterministically, but the process of solving them in a deterministic way or proving that they are np-hard is np-hard. Alternatively, the np-hard used in the process of proving that it is np-hard is also np-hard, and if this process occurs infinitely continuously in one problem, the problem cannot be proven. [cf. When np-hard is entered in the previous proof process, if the above conditions are not satisfied, a contradiction occurs as in the case of np-complete. so p=np-compete□

Additional proof

I'll give an example in the proof above. An independent set problem (IS) is proved to be np by selecting any k vertices with a non-deterministic algorithm and then determining whether they form an independent set, and this method is p. We then restoration 3-SAT[2] to IS. The problem of determining whether an algorithm exists is the p problem. sub(y,a,y)≔ IS

suppose np-complete is not p

~(∃x)Dem(x,sub(y,a,y))=Gödel problem m

In other words, IS is a problem that is not solved in polynomial time in a decisive way.

sub(y,a,y) = np-completea = Gödel problem y

G:~(∃x)Dem(x,sub(m,a,m)) corresponds to a from m(x,sub(m,a,m)) sub(m,a,m) = G:~(∃x)Dem(x,sub(m,a,m)) ⇔ Goedel problem g

The problem g is p problem [cf. Because it is a question of whether the Gödel problem IS has X, we can decisively find the answer in polynomial time as described above.] ~(∃x)Dem(x,sub(~(∃ x)Dem(x,g)) This problem is that g does not have an algorithm x that can be solved in a deterministic way. However, problem g is a p problem itself, and for this problem, algorithm x exists because it is a p problem, that is, it is non-contradictory to go back to the regression method and build the premise that p=np-complete. [Cf. Strictly speaking, if problem G is changed to (∃x)Dem(x,sub(m,a,m)) to make problem G true, It has contradictory answers in that it has algorithms to solve the wrong answer problem ~(∃ x)Dem(x,sub(m,a,m))].

assume np-complete is p problem

(∃x)Dem(x,sub(y,a,y))=m sub(y,a,y)=np-complete

G:(∃x)Dem(x,sub(m,a,m))

sub(m,a,m)=(∃x)Dem(x,sub(m,a,m))

There is no problem with the above equation, so there is no need to modify the prerequisites.

In other words, np-complete, which requires precondition modification, does not require precondition modification, unlike the precondition that is not p, so p = np-complete.

Therefore, the precondition that np is not p must be corrected.

That is, IS can be solved in polynomial time in a deterministic way. □

academic significance

If p=np-complete, there is a big advantage that many problems can be solved in polynomial time rather than exponential time. However, polynomial time does not have much meaning if the order is large, so it is not the purpose of this paper. The significance of this paper is that it provides a more rigorous definition of np-hard along with definitions of various mathematical problems. A clear definition of np-hard has great mathematical significance. It will help define not only np-hard but also np-complete, which already exists, and will make great progress in the algorithm. In fact, even if this is proven right away, many problems will not be immediately solved. However, if it is proven, more mathematical research will be conducted than now, and this will directly lead to an algorithm that can be solved in polynomial time. In other words, this can break the pessimistic mood that the current np-complete is not p. In other words, it will become a driving force for more active research than now and will enable further academic progress. In other words, this proof can lay the foundation for solving problems through further academic progress and more rigorous definitions of the problem. And my proof method will definitely have an impact and be of great help in solving other problems.

And it will bring a big change to our perception. In fact, intuitively, np-complete does not seem to be p, and I thought so before calculating it. In other words, it could serve as a counterexample to this intuition.

Author's Note

There is information that may be helpful to this paper. First of all, it is helpful to understand Dem, sub, and Gödel problems, which are the basis of this proof, and then read this paper. Since the proof is a simple calculation, it would be a good idea to check the contents of this paper to see if it is correct. The first proof of this paper is the basis for the second proof, and it is recommended that you view both proofs together.

Annotation

[1] e.g. (∃x)Dem(x,y)⁝=y has an algorithm x that is solved in polynomial time in a deterministic manner. So the answer to the problem is that x exists. That is, y is a p problem.

[2] 3-satisfiability problem