CS201 Discrete mathematics project
11912039 郑鑫颖

Topic: NP complete problem and related topic

OUTLINE

1. Classification of decision problems

1.1 P problem

P is the set of decision problems that can be decided by a computer in a polynomial time. For example, we use the algorithm with $O(n^3)$ complexity to solve the problem of matrix multiplication, so the problem of matrix multiplication belongs to P class.

1.2 NP problem

NP (Non-Deterministic Polynomial Problems) is the set of decision problems that we can decide them in polynomial time, if we are given the right certificate. Some problems, such as finding Hamiltonian loops in undirected graphs, are difficult to find (or may not exist at all) polynomial time algorithms. But if we are given an arbitrary loop, it is easy to determine whether it is a Hamiltonian loop by looking at whether all the vertices are in the loop.

1.3 NPC problem

To have a better understanding, we discuss it in part 4.

1.4 NP hard problem

We discuss it in part 5.

2. Reduction

Let's start with an old joke: A physicist and a mathematician are sitting together. Suddenly, the coffee machine caught fire. The physicist took a garbage can, emptied it, ran to the sink, filled it with water, and put out the fire. He then filled the trash can with water and put it next to the coffee machine. The next day, the same two people were sitting in the same place and the coffee machine caught fire again. This time, the mathematician stood up, picked up the garbage can filled with water, emptied the water, put some garbage in it, and handed it to the physicist. This reduces the problem to a problem that has been solved before. What mathematician does is the simple procedure of reduction.

A reduction is an algorithm for transforming one problem into another problem. An efficient reduction from one problem to another can be used to show that the second problem is at least as difficult as the first. (If problem A is more difficult than problem B, it means that the algorithm solving problem A requires higher complexity, greater memory, etc.) The existence of a reduction from A to B can be written in the shorthand notation $A \leqslant_p B$, usually with a subscript on the $\leqslant$ to indicate the type of reduction being used (p: polynomial reduction).

2.1 polynomial reduction

if an algorithm solving the second problem exists, then the first problem can be solved by transforming or reducing it to inputs for the second problem and calling the algorithm as a subroutine one or more times. If both the time required to transform the first problem to the second, and the number of times the subroutine is called is polynomial, then the first problem is polynomial-time reducible to the second. There are three most common types of polynomial-time reduction: Many-one reductions, Truth-table reduction and Turing reduction. Since reduction is not our focus, we will not

discuss it here in details.

2.2 steps of reduction

1. Transform an instance of problem A to an instance of problem B.
2. Solve the B instance
3. Transform the B result to a A result.

3. NP-Complete problem

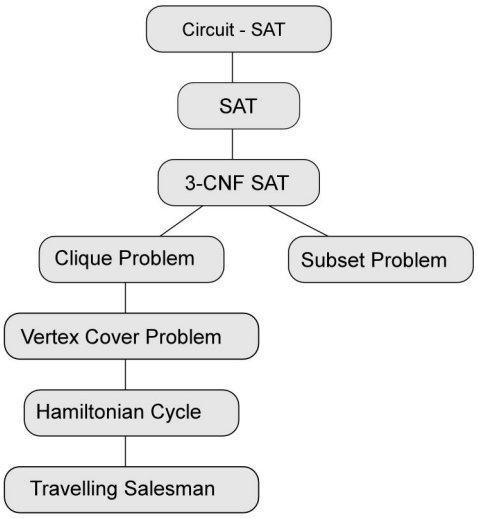
3.1 Definition

A problem X is NP-Complete if:

(1) X is in NP

(2) Every problem in NP reduces to X in polynomial time.

However, if we want to proof that a problem is NP-complete, a more commonly used way is to show some NP-complete problems known is reducible to this problem. The picture indicates the reductions used to prove their NP-completeness. If you know a problem is NPC, then you know that you will probably not be able to find an efficient algorithm for it.



3.2 SAT problem

3.2.1 Conjunctive Normal Form (CNF)

CNF is an essential concept when we discuss SAT problem. A boolean expression is said to be in CNF form if it is a conjunction of disjunctions (logical OR) of literals (clause): $F=C_1 \cap C_2 \cap C_3 \cap C_4 \cdots\cdots \cap C_n$, where $C_i=a_1^i \cup a_2^i \cup a_3^i \cdots\cdots \cup a_n^i$.

Every boolean expression can be expressed as a CNF by repeatedly using De Morgan's Laws, the law of double negation and distributive laws.

3.2.2 SAT problem

A given boolean expression, determining if there exists a truth assignment to its variables for which the expression is true is defined as the satisfiability problem.

3.2.2.1 2-SAT

Clause is defined as a disjunction (logical OR) of literals. In the 2-SAT problem, every clause has two literals and is solvable in a polynomial-time problem. Now let's look at its solution:

Step1: Pick a variable a, and then assign true.

Step2: Assign true to the other variable of all the clauses with negation a.

Step3: If those clauses are satisfied, then we pick the remaining clauses which don't have a or negation a and go to step2.

Step4: if the clauses with negation a are not all satisfied at this point, then we assign false to a and go to step2.

Step5: If all the clauses containing a and negation a are all not satisfied, then the 2-SAT is unsatisfiable.

Above all, we solve the 2-SAT problem in polynomial time.

3.2.2.2 Weighted 2-SAT

Given a CNF with 2 literals per clause and an integer k, the problem asked if there is a truth assignment with **at most** k variables is true for which the CNF is satisfiable. This problem is NP-Complete.

3.2.2.3 3-SAT

In the 3-SAT problem, each clause has at most 3 literals. This problem is NP-Complete. And it is always used to prove other problems to be NP-Complete.

4. Cook's Theorem

4.1 The proof of the Cook's Theorem

The statement of Cook-Levin theorem is the boolean satisfiability problem is NP-complete. A Boolean function is satisfiable if there exist X_1 to X_n so that φ (x_1...x_n) is true. To test whether there exists a truth assignment, we need to tested all possible assignments. If there are n variables in the boolean expression, then it would take $O(2^n)$ time to determine.

SAT problem belongs to NP class since if we are given a truth assignment for x_1,...,x_n, we can check if the Boolean function is True or not in polynomial time. So, what we need to prove is that there is a polynomial reduction A $\leqslant_p$ SAT for every A in NP.

Now let's proof the correctness of Cook's Theorem.

Because A∈NP, there is a verifier V which can verify the decision problem A in polynomial time. Let w be an input on which verifier V checks if the output is a yes or a no or. Because V works in polynomial time, at every time step, the algorithm makes the hardware affect a constant number of memory units. For example: assure that at every time step, the machine affects only one memory step.

The first time: following is the input.

1	1	0	1	0	1	1	0	1	0

The second time: the algorithm makes the first cell into 0.

0	1	0	1	0	1	1	0	1	0

The third time: the algorithm keeps the second cell as 1.

0	1	0	1	0	1	1	0	1	0

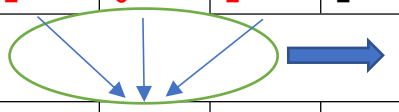
Then what we want to do is to convert each time step into a row of circuits and the initial input acts as an input into our first row. We want each output form the row of circuit to correspond exactly with what the algorithm does in each time step.

Look at an individual row for cell I.

0	1	0	1	0	1	1	0	1	0

Actually, cell I is only based on three cells: cell I itself, the cell before or after it (the Turing machine may move from left or right to cell I). So, we can build the circuit with this three as input and generate a specific result stored in cell I. And then, we can fill in the overall circuit with a bunch of that unit circuit to convert from row to row.

0	1	0	1	0	1	1	0	1	0



0	1	0	1	0	1	1	0	1	0

For the size of the overall circuit, assure that input has the size of n^k, so every row has the circuit of size n^k, and there are totally n^k steps. So the total size is $n^k * n^k$, which is still polynomial.

This circuit is a direct emulation of how the verifier algorithm works. So to solve a decision problem in NP, we need to be able to figure out if there's an easily verifiable proof for that problem which maps directly whether or not there is some inputs makes the boolean function, which is represented by the circuit, true. Which is the problem SAT. In this way, we have proofed the Cook's Theorem.

5. Extended topic

5.1 P versus NP problem

P versus NP problem asked whether NP set is the same as P set. If any NP-complete problem can be solved quickly, then every problem in NP can, because the definition of an NP-complete problem states that every problem in NP must be reducible to every NP-complete problem in polynomial time. This means solving any NP-Complete problem would solve all NP problems, furthermore, will prove P=NP.

5.2 NP-hard

A problem A is NP-hard when every problem L in NP can be reduced to it in polynomial time, that is, any problem $L \in NP$, $L \leqslant_p A$. So, finding a polynomial time algorithm to solve any NP-hard problem would give polynomial time algorithms for all the problems in NP, and will also lead to the result that P=NP.

6. References

Datta, Subham. "SAT and 3-SAT - Cook-Levin Theorem." Baeldung on Computer Science, 19 Oct. 2020, www.baeldung.com/cs/cook-levin-theorem-3sat#sat-variants.

"Many-one reduction." Wikipedia, Wikimedia Foundation, 6 November 2019, en.wikipedia.org/wiki/Many-one reduction.

"NP-Hardness." Wikipedia, Wikimedia Foundation, 17 Dec. 2020, en.wikipedia.org/wiki/NP-hardness.

"NP-completeness." Wikipedia, Wikimedia Foundation, 23 December 2020, en.wikipedia.org/wiki/NP-completeness.

"Polynomial-time reduction." Wikipedia, Wikimedia Foundation, 12 April 2020, en.wikipedia.org/wiki/Polynomial-time reduction.

"Reduction(complexity)." Wikipedia, Wikimedia Foundation, 18 May 2020, en.wikipedia.org/wiki/Reduction(complexity).

Undefined Behavior. (2018, August 14). NP-Complete Explained (Cook-Levin Theorem) [YouTube video]. Retrieved from https://www.bing.com/videos/search?q=cook+theorem+&&view=detail&mid=9E32B8C98AE893983FC69E32B8C98AE893983FC6&&FORM=VRDGAR&ru=%2Fvideos%2Fsearch%3Fq%3Dcook%2Btheorem%2B%26FORM%3DHDRSC3#

Carl Kingsford. (n.d.). The classes P and NP. Retrieved from: https://www.cs.cmu.edu/~ckingsf/bioinfo-lectures/np.pdf

The Cook-Levin Theorem(2020, February 28) Retrieved from: http://www.cs.toronto.edu/~ashe/cook-levin-handout.pdf