算法设计与分析

Lecture 10: NP Complete Theory

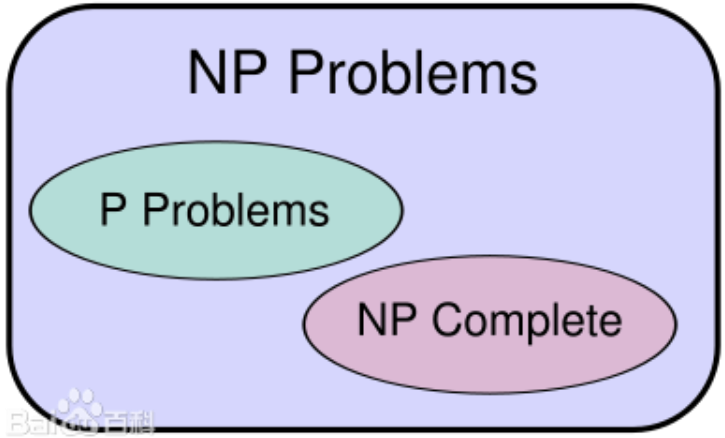
曹刘娟

厦门大学信息学院计算机系

caoliujuan@xmu.edu.cn

计算机理论核心问题

- P 等于NP?



- 千禧年七大数学难题之首

https://www.win.tue.nl/~gwoegi/P-versus-NP.htm
这个网页上有106个证明，有的说是，有的说不是

一个比较通俗易懂的博客
http://www.matrix67.com/blog/archives/105

A Story

- Suppose you work in industry, and your boss gives you the task of finding an efficient algorithm for some problem very important to the company.
- After laboring long hours on the problem for over a month, you have no idea at all toward an efficient algorithm.
- Giving up, you return to your boss and ashamedly announce that you simply can't find an efficient algorithm.
  - Your boss: "I'm going to fire you and hire some smarter guys to solve this problem!"
  - You: "It is not because I'm stupid, but it is not possible to find an efficient algorithm for this problem!"
- So, your boss gives you another month to prove that it is impossible.

A Story

- After a second month of hardworking, you fail again.
  - At this point you've failed to obtain an efficient algorithm and you've failed to prove that such an algorithm is not possible.
  - You are on the edge of being fired.
- Suddenly, you discover that the company's problem is similar to the 哈密顿回路 problem, and if the company's problem is solved, it will lead to a more efficient algorithm for the 哈密顿回路 problem.

A Story

- However, no one has ever found a more efficient algorithm for the 哈密顿回路 problem or proven that such an algorithm is not possible.
- You see one last hope. If you could prove that an efficient algorithm for the company's problem would automatically yield a more efficient algorithm for the 哈密顿回路 problem, it would mean that your boss is asking you to accomplish something that even the greatest computer scientists can't solve.
- You ask for a chance to prove this, and your boss agrees.

A Story

- After only a week of effort, you do indeed prove that an efficient algorithm for the company's problem would automatically yield a more efficient algorithm for the 哈密顿回路 problem.
- Rather than being fired, you're given a promotion because you have saved the company a lot of money.
- Your boss now realizes that it would not be worthy to continue to expend great effort looking for an exact algorithm for the company's problem and that other directions, such as looking for an approximate solution, should be explored.
- Happy ending~

Computational Complexity

- What we have just described is exactly what computer scientists have successfully done for the last 30 years.
- Given a problem, we can't solve it, and we can't prove it is impossible either, so we show that are equally hard as other similar problems.
  - If we had an efficient algorithm for any one of them, we would have efficient algorithms for all of them.
  - Such an algorithm has never been found, but it's never been proven that one is not possible.
- These interesting problems are called NP-complete (NP完全) and are the focus of this lecture.

DECISION PROBLEM

判定问题

Polynomial-Time Algorithm

Definition

A polynomial-time algorithm (多项式时间算法) is one whose worst-case time complexity is bounded above by a polynomial function of its input size. That is, if n is the input size, there exists a polynomial function $p(n)$ such that

$$W(n) = O\big(p(n)\big).$$

where $W(n)$ is the worst-case time complexity.

一个多项式时间算法他的最差情况复杂度是小于或者等于多项式时间的

Example

Algorithms with the following worst-case time complexities are all polynomial-time. 多项式时间

$$2n \quad 3n^3 + 4n \quad 5n + n^{10} \quad n \lg n$$

Algorithms with the following worst-case time complexities are not polynomial-time. 非多项式时间

$$2^n \quad 2^{0.01n} \quad 2^{\sqrt{n}} \quad n!$$

Polynomial-Time Algorithm

- In computer science, we divide problems into “easy group” and “difficult group” based on the boundary: can be solved in polynomial-time or not.
- A problem is called intractable (难解的) if it is impossible to solve it with a polynomial-time algorithm.
- A problem can be solved in $\Theta(n^{100})$ is also in “easy group”.
  - Once we have a $\Theta(n^{100})$ algorithm, we may improve it to a $\Theta(n^{90})$ algorithm a few moments later.

The Three General Problem Categories

- There are three general categories of problems we concern:
  1. Problems for which polynomial-time algorithms have been found. 找到多项式时间算法
  2. Problems that have been proven to be intractable. 被证明了算法是非多项式的
  3. Problems that have not been proven to be intractable, but for which polynomial-time algorithms have never been found. 不能证明是非多项式的，也找不到多项式时间算法
- It is a surprising phenomenon that most problems in computer science fall into either the first or third category. 大多数计算机的问题属于1或者3
  - It's extremely hard to prove the intractability of a problem (can't find a polynomial-time algorithm for it).

The Three General Problem Categories

1. Problems for which polynomial-time algorithms have been found.

- For example, we have found:
  - $\Theta(n \lg n)$ algorithms for sorting.
  - $\Theta(\lg n)$ algorithm for searching a sorted array.
  - $\Theta(n^{2.38})$ algorithm for matrix multiplication.
  - $\Theta(n^3)$ algorithm for chained matrix multiplication.
- There are some other problems for which we have developed polynomial-time algorithms, but for which the obvious brute-force algorithms are non-polynomial (usually exponential).
  - E.g. the shortest paths problem, the optimal BST problem, and the MST problem.

The Three General Problem Categories

3. Problems that have not been proven to be intractable, but for which polynomial-time algorithms have never been found.（尚未被证明为难以处理的问题，但从未找到多项式时间算法的问题。）

- For example, the traveling salesperson problem, the sum-of-subsets problem, the m-coloring problem for $m \geq 3$, and the Hamiltonian cycle problem all fall into this category.
- We have found branch-and-bound algorithms, backtracking algorithms, and other algorithms for these problems that are efficient for many large instances. However, they are still not polynomial-time.
  分治限界法，回溯法等能够有效的解答这些问题，但是这些方法不是多项式时间的

Decision Problem 判定问题

- It is more convenient to develop the theory if we restrict ourselves to decision problems (判定问题).
- 我们暂时不考虑最优化等比较难的问题，我们考虑简单点的判定问题，判定问题的输出是“是”或“否”
- The output of a decision problem is a simple “yes” or “no” answer.
- Yet when we introduced some of the problems mentioned previously, we presented them as optimization problems.
  - The output is an optimal solution.
- Each optimization problem has a corresponding decision problem.

Example 将优化问题转为判定问题

- The 0-1 knapsack optimization problem is to determine the maximum total profit of the items that can be placed in a knapsack with capacity W.
- The corresponding 0-1 knapsack decision problem is to determine, for a given profit P, whether it is possible to load the knapsack so as to keep the total weight no greater than W, while making the total profit **at least** equal to P.
  - This problem has the same parameters as the 0-1 knapsack optimization problem plus the additional parameter P. 判定问题和优化问题相比多了一个参数P

https://en.wikipedia.org/wiki/Knapsack_problem

Example

Optimization problem

i	v_i	w_i
1	\$10	1kg
2	\$12	1kg
3	\$15	2kg
4	\$20	3kg

W=5kg

What is the maximum total profit?
优化问题：最大利润是多少?

Decision problem

i	v_i	w_i
1	\$10	1kg
2	\$12	1kg
3	\$15	2kg
4	\$20	3kg

W=5kg

Can we make maximum total profit not less than 42?
我们能得到的最大利润能不少于42么?

Example 最短路径问题

- The shortest path optimization problem is to find the shortest path from vertex u to vertex v in an unweighted graph $G = (V, E)$. 最短路径优化问题
- The corresponding shortest path decision problem is to determine whether it is possible to find a path from vertex u to vertex v in an unweighted graph $G = (V, E)$, though at most k edges. 最短路径判定问题，到底存在一条长度的k的路径么?
  - This problem has the same parameters as the shortest path optimization problem plus the additional parameter k. 这里判定问题比优化问题多了一个参数k

Decision Problem

- A polynomial-time algorithm for an optimization problem automatically solves the corresponding decision problem. 一个多项式时间的优化问题算法可以用来求解对应的判定问题
  - E.g., find the shortest path and compare it with k.
- For many decision problems, it's been shown that a polynomial-time algorithm for the decision problem would also yield a polynomial-time algorithm for the corresponding optimization problem. 对于很多优化问题，他们对应的多项式时间判定算法可以转化为多项式时间的优化问题求解算法
  - Therefore, we can initially develop our theory considering only decision problems.
- They are equivalent!

Example 最小着色问题

- Considering the minimum vertex coloring problem.
- If we have a polynomial-time algorithm for the corresponding decision problem: ColorIsTrue(G, k). It returns true if and only if graph G can be k-colored. 能不能对一个图用k种颜色进行着色
- The algorithm for minimum vertex coloring problem can be constructed by:

```
MinColoring(G)
1 k ← 0
2 while ColorIsTrue(G, k) ≠ True do
3     k ← k + 1
4 return k
```

- It can be easily verified that MinColoring(G) is also polynomial-time.

通过这种方法将判定问题算法，转换为优化问题算法

P AND NP

Encoding 编码

- If a computer program is to solve an abstract problem (抽象问题), problem instances must be represented in a way that the program understands. 将一个抽象问题的实例编码成为计算机能够理解的
- An encoding is a mapping e from abstract objects to the set of binary strings. 编码将抽象的问题转换为二进制字符串
- Thus, a computer algorithm that "solves" some abstract decision problem actually takes an encoding of a problem instance as input. 一个计算机算法求解某个抽象判定问题是通过将问题实例的编码当做输入进行的。

Encoding 编码示例

- Abstract problem: 0/1 knapsack problem.
- Problem instance: 0/1 knapsack problem with $w_1 = 1, w_2 = 1, w_3 = 2, v_1 = 10, v_2 = 12, v_3 = 15, W = 3$.
- Encoding of this problem instance:

$$11001010101 \ldots.101010010101$$

Encoding

- We call a problem whose instance set is the set of binary strings a concrete problem (具体问题).
- Given a problem instance i of length $n = |i|$, we say that an algorithm solves a concrete problem in time $O\big(T(n)\big)$ if the algorithm can produce the solution in $O\big(T(n)\big)$ time.

一个算法在O(T(n))时间内得出解，就说该算法在O(T(n))时间内求解了一个具体问题

- A concrete problem is polynomial-time solvable (多项式时间可解的) if there exists an algorithm to solve it in time $O(n^k)$ for some constant k.

A Formal-Language Framework

- The formal-language framework allows us to express the relation between decision problems and algorithms that solve them concisely. 从形式语言的角度来分析具体问题和算法的关系
- An alphabet (字符集) Σ is a finite set of symbols.
- A language (语言) L over Σ is a set of strings made up of symbols from Σ.
  - For example, if $\Sigma = \{0,1\}$, the language $L = \{10, 11, 101, 111, 1011, 1101, 10001, \ldots\}$ is the language of binary representations of prime numbers.

A Formal-Language Framework

- Denote empty string by $\boldsymbol{\varepsilon}$, and the empty language by $\emptyset$.
- The language of all strings over Σ is denoted Σ^*.
  - For example, if $\Sigma = \{0,1\}$, then $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \dots\}$ is the set of all binary strings.
- Every language L over Σ is a subset of Σ^*.

A Formal-Language Framework

- As we encode every problem instance into binary representation, the set of instances for any decision problem Q is simply a subset of Σ^*, where $\Sigma = \{0,1\}$.
- Let x be an instance of the decision problem Q and $Q(x) = 1$ if x is true.
- Since Q is entirely characterized by those problem instances that produce a 1 (yes) answer, we can use language L over $\Sigma = \{0,1\}$ to represent problem Q, where （可以用语言来表达一个问题）

$$L = \begin{bmatrix} 10010010 \ldots 1010010 \\ 10111010 \ldots 1110100 \\ \vdots \\ 11001001 \ldots 1100011 \end{bmatrix} = \{x \in \Sigma^* : Q(x) = 1\}.$$

如果用另外一个字符集来表示语言，就变成了另外一个L，但是Q还是这个Q

Q和L其实是一回事，Q是抽象的，L是具体的
L包含所有判定为真的问题实例

A Formal-Language Framework

- Now, we can use language to represent problem.
- For example, the decision problem PATH has the corresponding language:

PATH = $\{\langle G, u, v, k\rangle$: $G = (V, E)$ is an undirected graph, $u, v \in V$, $k \geq 0$ is an integer, and there exists a path from u to v in G consisting of at most k edges$\}$.

A Formal-Language Framework

Definition 11.1

We say that an algorithm A **accepts (接受)** a string $x \in \{0,1\}^*$ if, given input x, the algorithm's output $A(x)$ is 1.

The language accepted by an algorithm A is the set of strings:

$$L = \{x \in \{0,1\}^* : A(x) = 1\}$$

that is, the set of strings that the algorithm accepts.

An algorithm A **rejects (拒绝)** a string x if $A(x) = 0$.

- However, we can't say that algorithm A solves L, because we're not sure if A will reject all $x \notin L$. 不能说算法A能够求解语言L
  - A may iterate forever for such x. 可能无限循环

A Formal-Language Framework

Definition 11.2

A language L is **decided (判定)** by an algorithm A if $\forall x \in L$ is accepted by A and $\forall x \notin L$ is rejected by A. 如果一个算法A接受L中的每个串，并且拒绝不在L中的串，就称A能够判定语言L

A language L is accepted 接受 in polynomial time by an algorithm A if there is a constant k such that for any length-n string $x \in L$, algorithm A accepts x in time $O(n^k)$.

A language L is decided 判定 in polynomial time by an algorithm A if there is a constant k such that for any length-n string $x \in \{0,1\}^*$, algorithm A correctly decides whether $x \in L$ in time $O(n^k)$.

判定的要求比接受更高，判定要求不仅仅能够在多项式时间接受L中的串，并且要在多项式时间拒绝不在L中的串

A Formal-Language Framework

- We can informally define a **complexity class (复杂类)** as a set of languages, membership in which is determined by a complexity measure, such as running time, of an algorithm that determines whether a given string x belongs to language L. 复杂类是语言的集合，比如按照某种时间复杂度将这些语言归类在一起
- Using this language-theoretic framework, we can provide an alternative definition of the complexity class P:

Definition 11.3

P = $\{L \in \{0,1\}^* :$ there exists an algorithm A that decides L in polynomial time$\}$. P是指有算法能在多项式时间内判定的语言的集合

Theorem 11.1

P = $\{L : L$ is accepted by a polynomial-time algorithm$\}$.

P and NP

- All decision problems for which we have found polynomial-time algorithms are certainly in P. 所有的我们能找到多项式时间复杂度算法的问题，肯定是P问题
  - Searching, sorting, matrix multiplication…
- However, could some decision problem like Hamiltonian Graph decision problem for which we have not found a polynomial-time algorithm also be in P?
  - We don't know! It could possibly be in P.
  - No one finds a polynomial-time algorithm and no one proves that it is not in P.

Hamiltonian Cycle Problem

- A Hamiltonian cycle (哈密顿回路) of an undirected graph $G = (V, E)$ is a simple cycle that contains each vertex in V.
- A graph that contains a Hamiltonian cycle is said to be Hamiltonian; otherwise, it is non-Hamiltonian.
- The Hamiltonian cycle problem "Does a graph G have a Hamiltonian cycle?" can be defined as a formal language:

$$\text{HamCycle} = \{G : G \text{ is a Hamiltonian graph}\}.$$

- Solving this problem can trace back to hundreds years ago. Now, we still can't find a polynomial-time algorithm that decides HamCycle.

P and NP

- It seems that the research on this kind of hard problem gets stuck.
- Instead, for this kind of decision problem like Hamiltonian Cycle decision problem, we'd like to if we can verify an answer in polynimial-time. 对于不确定能不能找到多项式时间判定算法的问题，我们希望知道能不能在多项式时间内验证其中某个答案的正确性

Hamiltonian Cycle Problem

- Now, if god tells you a cycle in G and let you verify if this cycle is a Hamiltonian cycle.
- Is this problem easy to solve?
- Of course! Simply check if it is a simple cycle and if every edge in this cycle is in E.
- The verification can be done in $O(n^2)$. It is obviously polynomial time.

Verification Algorithms 验证算法

- A verification algorithm (验证算法) can be defined as an algorithm A with two arguments: 一个验证算法要两个输入参数
  - An ordinary input string x, which is a problem instance; 具体问题实例
  - A string y called a certificate (证书), which is a given answer of x. 一个具体问题的答案
- Verification algorithm A verifies an problem instance string x if there exists a certificate y such that $A(x, y) = 1$.
- The language can be verified by a verification algorithm A is

  $L = \{x \in \{0,1\}^* : \text{there exists } y \in \{0,1\}^* \text{ such that } A(x, y) = 1\}$.
  - Intuitively, an algorithm A verifies a language L if for any string $x \in L$, there is a certificate y that A can use to prove that $x \in L$.
  - Moreover, for any string $x \notin L$, there must be no certificate proving that $x \in L$.

NP

Definition 11.3

P = $\{L \in \{0,1\}^* :$ there exists an algorithm A that decides L in polynomial time$\}$. P是指有算法能在多项式时间内判定的语言的集合

Definition 11.4

The complexity class NP is the class of languages that can be verified by a polynomial-time algorithm. 在多项式时间里面被验证

- Notice that NP stands for “non-deterministic polynomial”, rather than “non-polynomial”.
  - What is non-deterministic???

Nondeterministic Algorithm

- To state the notion of polynomial-time verifiability more concretely, we introduce the concept of a non-deterministic algorithm (非确定性算法).
- We can think of such an algorithm as being composed of the following two separate stages:
  1. **Guessing (Non-deterministic) Stage:** Guess a solution to the given instance. It is called non-deterministic because this stage is totally random.
  2. **Verification (Deterministic) Stage:** Verify the answer: (1) the answer for this instance is “yes”, (2) the answer for this instance is “no”, or (3) can’t verify at all (that is, going into an infinite loop). It is called deterministic because only “yes” or “no” can be produced in this stage.

Nondeterministic Algorithm

- For the Hamiltonian cycle problem, we can design a nondeterministic algorithm:

```
NondeterministicHamCycle(G)
1 S ← V
2 p ← ∅
3 for i ← 1 to |V| do
4     Randomly select one vertex v from S
5     p ← p ∪ {v}
6     S ← S − {v}
7 if verify(G, p)=1 then return True
8 else return False
```

Polynomial-Time Nondeterministic Algorithm

Definition

A polynomial-time non-deterministic algorithm is a non-deterministic algorithm whose verification stage is a polynomial-time algorithm. 多项式时间非确定性算法的验证阶段时多项式时间的

Definition

NP is the set of all decision problems that can be solved by polynomial-time non-deterministic algorithms. NP就是所有能够被多项式非确定算法所求解的判定问题集合

- We are not sure if one can be solved in polynomial-time, but we know that we can verify an answer of it in polynomial-time.
- This definition is equivalent to the previous definition.

P and NP

- There are thousands of problems that no one has been able to solve with polynomial-time algorithms but that have been proven to be in NP because polynomial-time nondeterministic algorithms have been developed for them.
  - The answers to them can be verified in polynomial-time.
- Every problem in P is also in NP. 所有的P问题都是NP问题
  - This is trivially true because any problem in P can be solved by a polynomial-time algorithm.
  - When it is solved, it is also verified.
- There are only few problems that have been proved not in NP.
  - The intractable problems, e.g. the Halting problem
  - 有一些问题不是NP问题，比如停机问题

P and NP

- Because P is in NP, it is easy to think that NP contains P as a proper subset.
- However, this may not be the case. That is, no one has ever proven that there is a problem in NP that is not in P. 目前没有人能够找到不是P的一个NP问题
- No one knows if P=NP or not yet.
  - If P=NP, it means that once we can verify answer of a problem in polynomial-time, we can solve it in polynomial-time! 如果 P=NP，意味着如果我们能够在多项式时间内验证一个问题的答案，那么我们也能在多项式时间内求解他

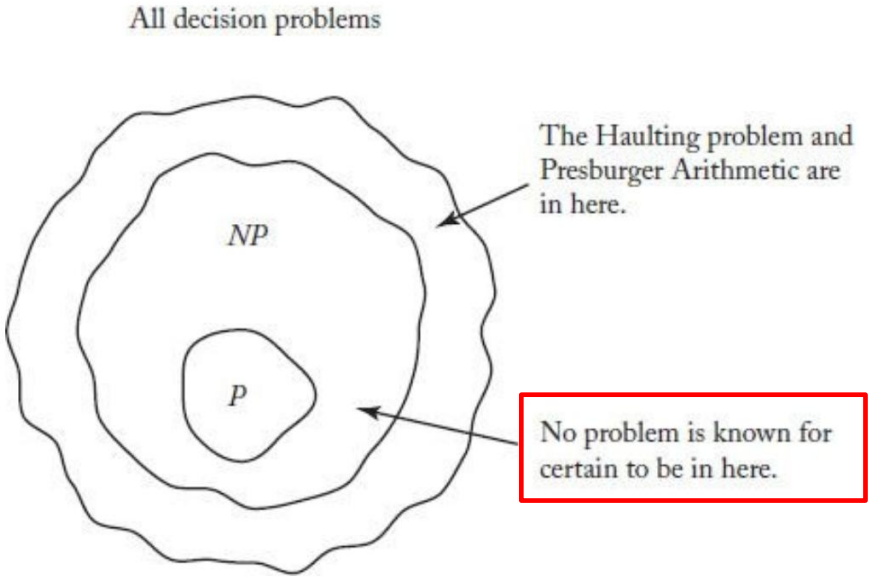


Image source: Figure 9.3, Richard E. Neapolitan, Foundations of Algorithms (5th Edition), Jones & Bartlett Learning, 2014

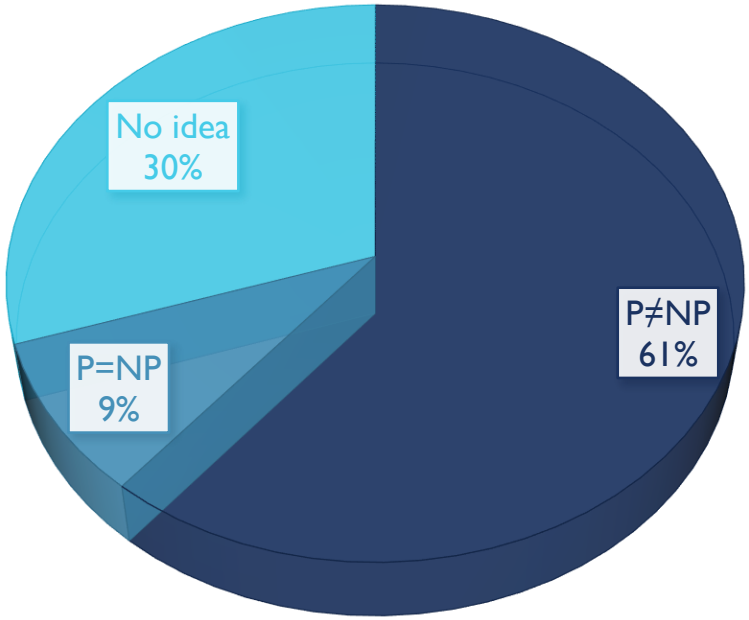
P=NP?

- To prove P=NP or P≠NP is one of the Millennium Prize Problems (千禧年大奖难题), which are seven problems in mathematics that were stated by the Clay Mathematics Institute on May 24, 2000.
  - A correct solution to any of the problems results in a US$1 million prize being awarded by the institute to the discoverer(s).
- They are:
  - ? P versus NP (P/NP问题)
  - ? Birch and Swinnerton-Dyer conjecture (贝赫和斯维讷通-戴尔猜想)
  - ? Hodge conjecture (霍奇猜想)
  - ? Navier-Stokes existence and smoothness (纳维-斯托克斯存在性与光滑性)
  - ☑ Poincaré conjecture (庞加莱猜想)
  - ? Riemann hypothesis (黎曼猜想)
  - ? Yang-Mills existence and mass gap (杨-米尔斯存在性与质量间隙)

Statistic source: http://www.cs.umd.edu/~gasarch/papers/poll.pdf

P=NP?

A POLL OVER 100 COMPUTER SCIENTISTS IN 2002



Statistic source: http://www.cs.umd.edu/~gasarch/papers/poll.pdf

Consequences of Solution

- Either P=NP or P≠NP is proved would advance theory enormously, and perhaps have huge practical consequences as well.
- If P=NP is proved:
  - Some cryptography encryption methods in NP will not be treated safe any more.
  - Researchers will be more confident to propose polynomial-time algorithms for problems in NP.
- If P≠NP is proved:
  - Some cryptography encryption methods in NP will be treated safe.
  - It would allow one to show in a formal way that many common problems cannot be solved efficiently, so that the attention of researchers can be focused on partial solutions or solutions to other problems.

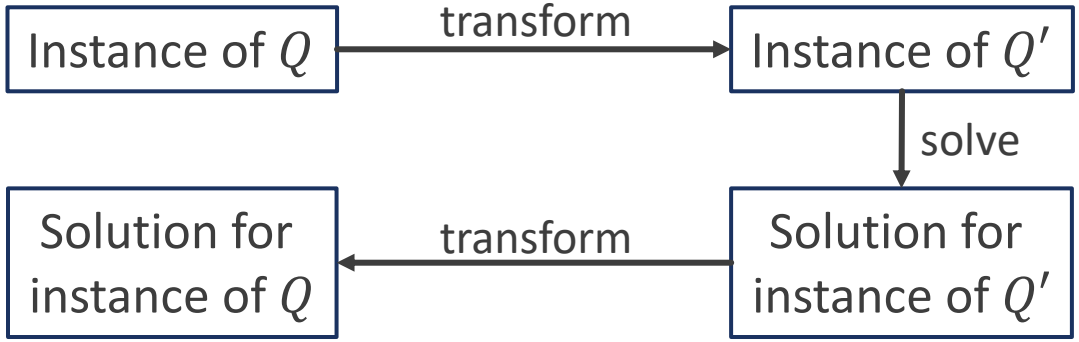
NP COMPLETE

NP Complete

- There are a lot of problems:
  - No polynomial-time algorithm is found; 没有找到多项式时间算法
  - Can't prove it is impossible to find a polynomial-time algorithm. 不能证明找不到多项式时间算法
- There is a special group among these problems. If a polynomial-time algorithm is developed for one of it, every problem in this group can be solved in polynomial time. 一些问题，如果能找到其中一个问题的多项式时间算法，那么所有的这些问题都能在多项式时间内解决
- This group is called NP complete.
  - There are more than 3000 known NP-complete problems. See here for some typical ones.

Reducibility

- Before formally define NP complete, we first introduce the concept of reducibility (约简).
- If any instance of problem Q can be easily transformed into an instance of problem Q', we say that problem Q reduces to problem Q', and the solution for the transformed instance of Q' is also the solution for the instance of Q.



- If we find such a transformation, Q will not be harder than Q'.

Example

- Problem Q: solve linear equation $ax + b = 0$.
- Problem Q': solve quadratic equation $ax^2 + bx + c = 0$.
- Any instance of Q can be transformed into an instance of Q':

$$ax + b = 0 \quad \Rightarrow \quad 0x^2 + ax + b = 0$$

Definition 11.6

A language L_1 is polynomial-time reducible to a language L_2, written $L_1 \leq_P L_2$, if there exists a polynomial-time computable function f: $\{0,1\}^* \to \{0,1\}^*$ such that for all $x \in \{0,1\}^*$,

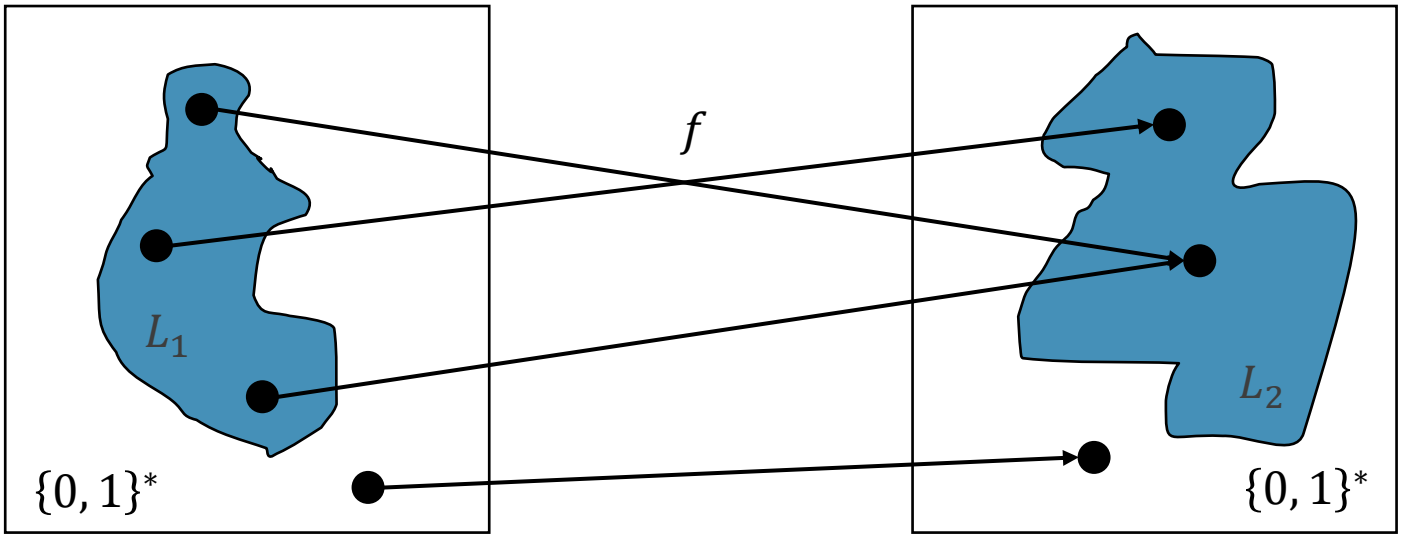
$$x \in L_1 \text{ if and only if } f(x) \in L_2$$

We call the function f the reduction function (约简函数), and a polynomial-time algorithm A that computes f is called a reduction algorithm (约简算法).

- The symbol $\leq_P$ means polynomial-time reducible. $L_1 \leq_P L_2$ indicates the difficulty of L_1 will not be higher than L_2.

Reducibility

- The reduction function f provides a polynomial-time mapping such that if $x \in L_1$, then $f(x) \in L_2$. Moreover, if $x \notin L_1$, then $f(x) \notin L_2$.



Reducibility

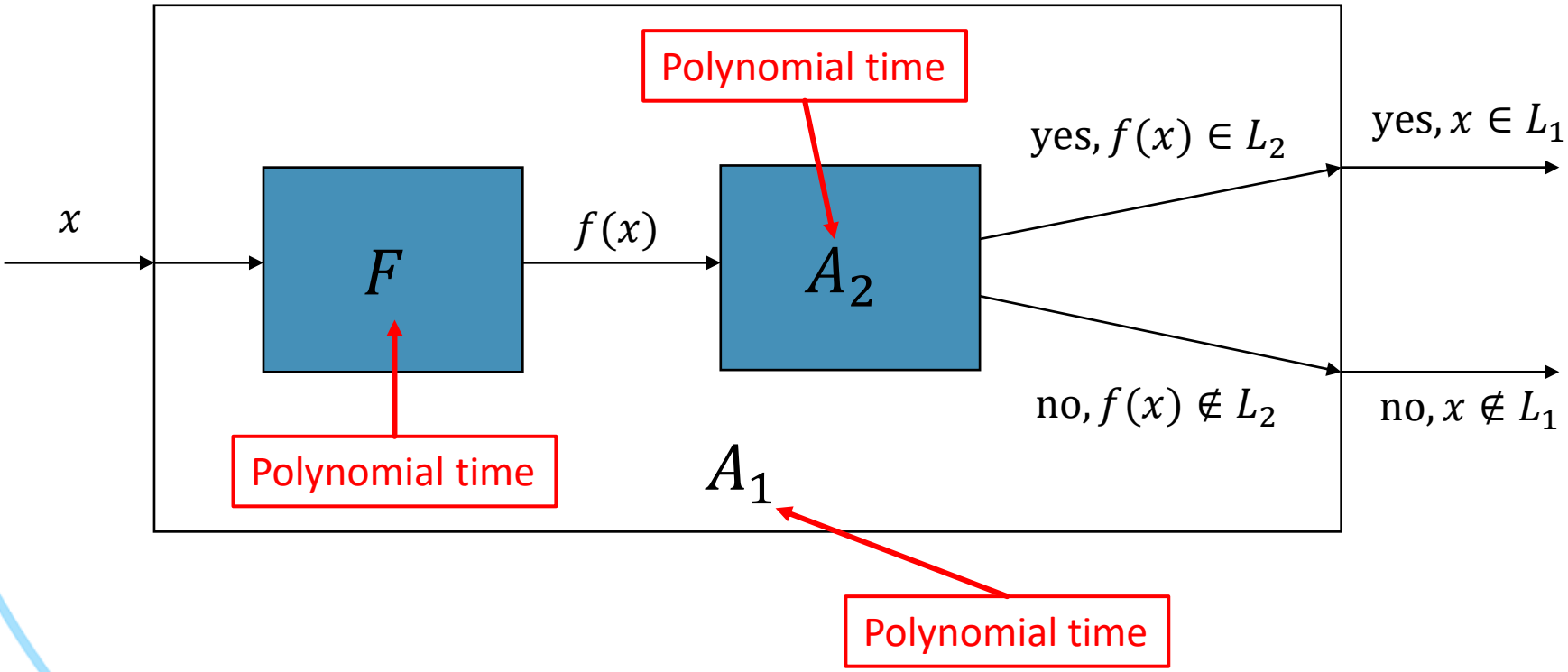
Lemma 11.1

If $L_1, L_2 \in \{0,1\}^*$ are languages such that $L_1 \leq_P L_2$, then $L_2 \in P$ implies $L_1 \in P$.

Proof:

- Let A_2 be a polynomial-time algorithm that decides L_2, and let F be a polynomial-time reduction algorithm that computes the reduction function f.
- We can construct a polynomial time algorithm A_1 that decides L_1.

Reducibility



NP-Completeness

- Polynomial-time reductions provide a formal means for showing that one problem is at least as hard as another, to within a polynomial-time factor.

Definition 11.7

A language $L \in \{0,1\}^*$ is NP-complete (NPC, NP完全) if

1. $L \in \mathrm{NP}$, and

2. $L' \leq_P L$ for every $L' \in \mathrm{NP}$.

If a language L satisfies property 2, but not necessarily （不一定满足） property 1, we say that L is NP-hard (NP难).

- Usually, the optimization problem of an NPC decision problem is NP-hard, because verifying an optimization problem is not in NP.

NP-Completeness

- A problem is NP-complete if it is both in NP and NP-hard.
- The NP-complete problems represent the hardest problems in NP. NPC问题是NP问题中最难的 （只是NP问题里面最难的）
- If any NP-complete problem has a polynomial time algorithm, all problems in NP do. 如果一个NPC问题有多项式时间算法，所有的NP问题都有多项式时间算法

Theorem 11.2

If any NP-complete problem is polynomial-time solvable, then P=NP. Equivalently, if any problem in NP is not polynomial-time solvable, then no NP-complete problem is polynomial time solvable.

NP-Completeness

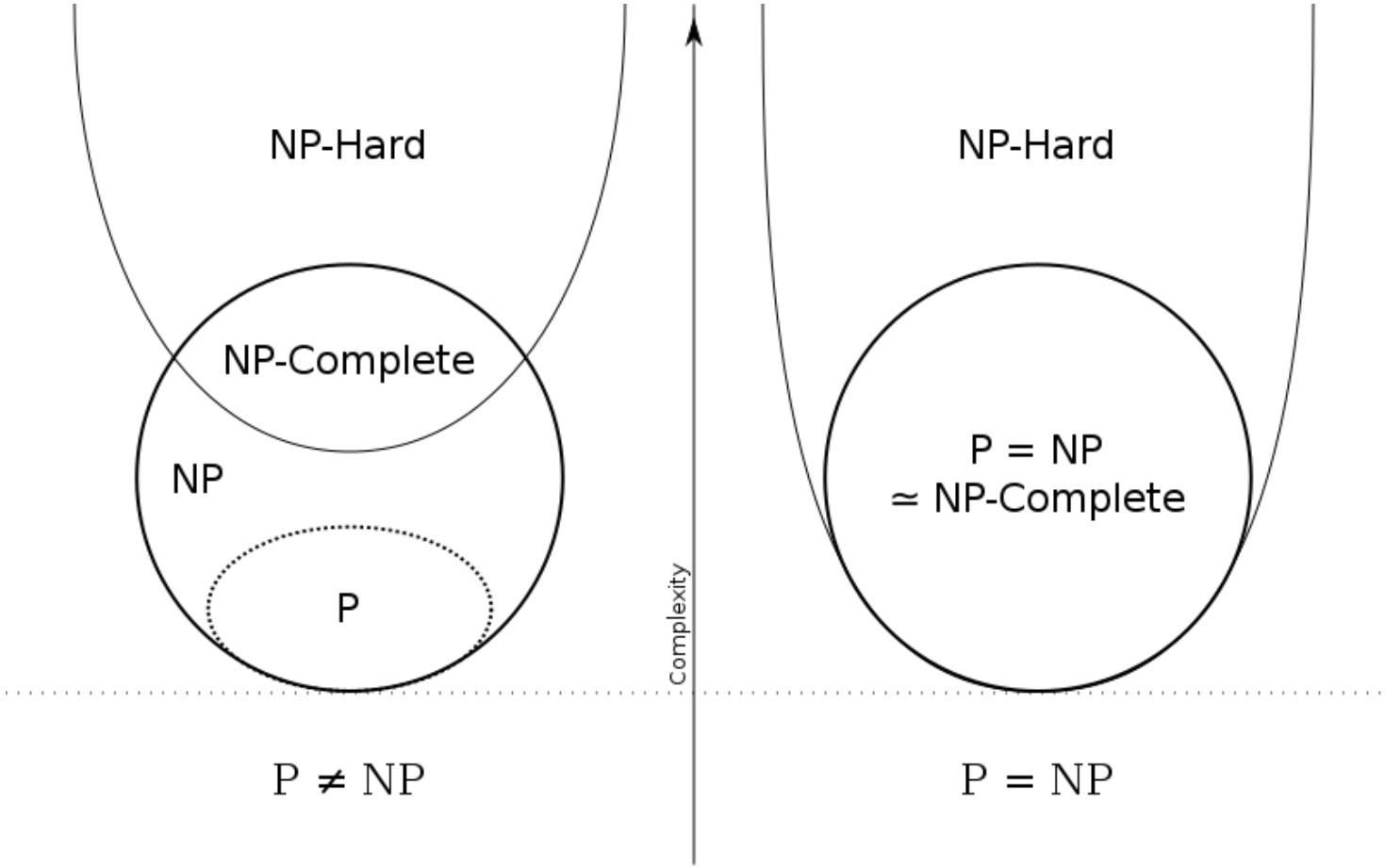


Image source: https://en.wikipedia.org/wiki/NP-completeness

NP-Completeness Proofs

- By Definition 11.7, if we want to prove **language L be NPC**, we should prove: 要证明一个语言是NPC，必须满足一下两个条件

  1. $L \in NP$，　L 属于NP

  2. $L' \leq_P L$ for every $L' \in NP$. 对于所有属于NP的语言L'，都能约简到这个语言L

- Property 1 is easy to prove by constructing a polynomial-time verification algorithm（多项式时间验证算法）.

- How to prove property 2? Find all $L' \in$ NP in the universe and show $L' \leq_P L$? 第二个性质咋证明?

NP-Completeness Proofs

如果能把一个NPC的语言约简为L，**那么L是NP-hard，如果L还是NP的，那么L是NPC**

Lemma 11.4

If L is a language such that $L' \leq_P L$ for some $L' \in \text{NPC}$, then L is NP-hard. Moreover, if $L \in \text{NP}$, then $L \in \text{NPC}$.

- This Lemma tells you: you don't need to prove $L' \leq_P L$ for all $L' \in \text{NP}$, just prove $L' \leq_P L$ for one $L' \in \text{NPC}$. 不需要证明所有的np问题都可以约简为L，只需证明有NPC问题可以被约简为L就可以了

Proof:

- Since L' is NPC, for all $L'' \in \text{NP}$, we have $L'' \leq_P L'$.
- By supposition (假设), $L' \leq_P L$, and thus by transitivity, we have $L'' \leq_P L' \leq_P L$ for all $L'' \in \text{NP}$, which shows that L is NP-hard.
- If $L \in \text{NP}$, we also have $L \in \text{NPC}$.

NP-Completeness Proofs 证明NPC的5个步骤

Method to prove that a language L is NP-complete:

1. Prove $L \in \mathrm{NP}$.
2. Select a known NP-complete language L'.
3. Describe an algorithm that computes a function f mapping every instance $x \in \{0,1\}^*$ of L' to an instance $f(x)$ of L.
4. Prove that the function f satisfies $x \in L'$ if and only if $f(x) \in L$ for all $x \in \{0,1\}^*$.
5. Prove that the algorithm computing f runs in polynomial time.

NP-Completeness Proofs

- Things seem to be easy now, we can use an NPC problem to prove more NPC problems.
- However, where is the first NPC problem?

Satisfiability Problem 可满足问题

- A Boolean formula φ composed of:
  - n Boolean variables: $x_1, x_2, \ldots, x_n$;
  - m Boolean connectives (连接符): such as ∧ (AND), ∨ (OR), ¬ (NOT), → (implication), ↔ (if and only if);
  - Parentheses 括号
- If we can find a set of values for the variables of φ that causes it to evaluate to 1, we call this formula φ is satisfiable (可满足的).
- The satisfiability problem (SAT, 可满足性问题) is to determine if a formula is satisfiable.

Example

- Given a Boolean formula:

$$\varphi = \Big((x_1 \to x_2) \vee \neg\big((\neg x_1 \leftrightarrow x_3) \vee x_4\big)\Big) \wedge \neg x_2$$

  is this formula satisfiable?

- Yes, assign $x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 1$, we get:

$$\begin{aligned} &\Big((0 \to 0) \vee \neg\big((\neg 0 \leftrightarrow 1) \vee 1\big)\Big) \wedge \neg 0 \\ =\ &\big(1 \vee \neg(1 \vee 1)\big) \wedge 1 \\ =\ &1 \end{aligned}$$

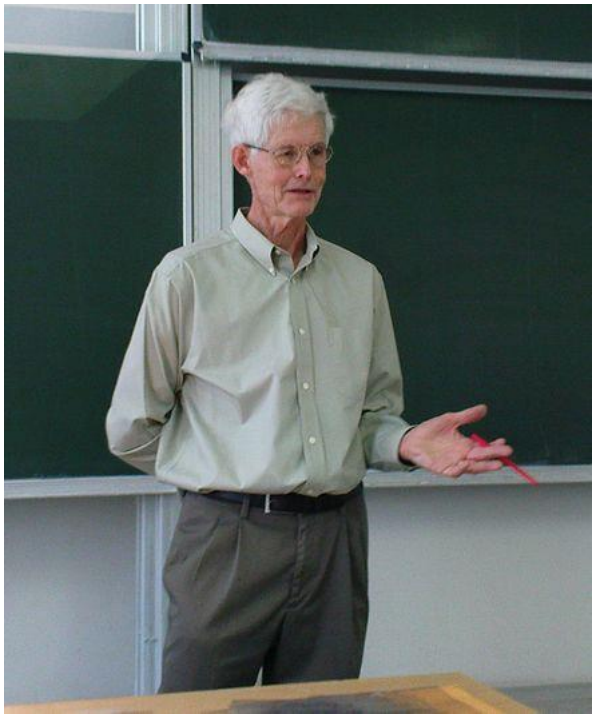
A → B 等价于 ¬A ∨ B（非A或B）。

Satisfiability Problem

Theorem 11.4

SAT is NP-complete.

- In 1971, Stephen Cook proves the first NPC problem.
- This problem itself is not interesting at all.
- However, we can use it to prove NP-completeness of other problems.



Stephen Cook
Turing Award in 1982

Image source: https://en.wikipedia.org/wiki/Stephen_Cook

Satisfiability Problem

Proof:

- Show that SAT is in NP.
  - The verifying algorithm simply replaces each variable in the formula with its corresponding value and then evaluates the expression.
  - This task is easily done in polynomial time.
  - If the expression evaluates to 1, the formula is satisfiable.
- Show that SAT is NP-hard.
  - Not that easy to prove.

Satisfiability Problem

- It can be shown that:
  - SAT $\leq_P$ Hamiltonian cycle decision problem.
  - Hamiltonian cycle decision problem $\leq_P$ undirected traveling salesperson decision problem.
  - Undirected traveling Salesperson decision problem $\leq_P$ traveling salesperson decision problem.
  - …
- Now, we don't need to use the definition to prove the NP-completeness of a problem. Instead, simply find another NP-complete problem and show the reducibility.我们不需要按照NPC的定义来证明一个问题是NPC，我们只需证明一个NPC问题的语言能够多项式时间约简为这个问题

3-CNF Satisfiability

- A literal in a Boolean formula is an occurrence of a variable x_i or its negation $\neg x_i$. x1和非x1是不一样的文字
- A Boolean formula is in conjunctive normal form (CNF, 合取范式), if it is expressed as an AND of clauses, each of which is the OR of one or more literals. 如果是一系列子句的合取，而子句是一个或者多个文字的析取
- A Boolean formula is in 3-CNF, if each clause has exactly three distinct literals. For example:(每个子句精确含有3个不同的文字)

$$(x_1 \vee \neg x_1 \vee \neg x_2) \wedge (x_3 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4)$$

- In 3-CNF-SAT problem, we are asked whether a given Boolean formula φ in 3-CNF is satisfiable. 判断一个以3-CNF方式表达的布尔表达式φ是不是可满足

NP-Completeness of 3-CNF Satisfiability

Theorem 11.5

3-CNF-SAT is NP-complete.

Proof:

- **Step 1**: 3-CNF-SAT in NP is obvious, just like SAT problem. 给定一个3-CNF-SAT的证书（答案），将布尔公式的变元替换成实际赋值，这个过程显然是多项式时间的，如果结果是1，那么就是可满足的
- **Step 2**: Prove SAT$\leq_P$3-CNF-SAT.
- Therefore, we need to construct a reduction algorithm to transform any formula into 3-CNF.
  - We have learned how to transform any formula into CNF in discrete math.

  上学期上的离散数学里面有讲如何将任何的布尔公式转为CNF
- The reduction algorithm has three steps.

NP-Completeness of 3-CNF Satisfiability

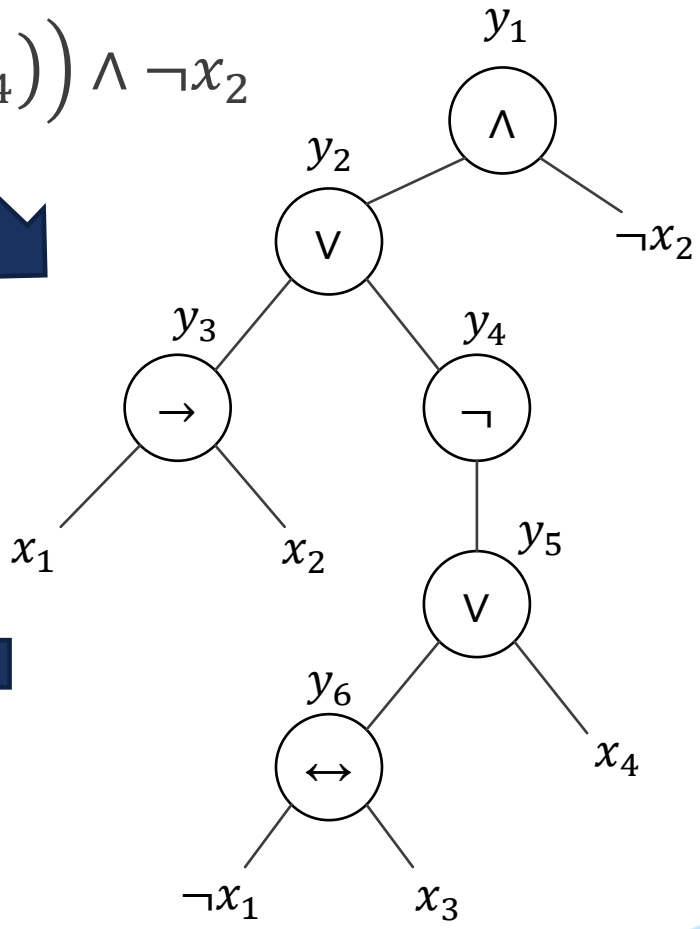
Proof (cont'd):

- The first step is to construct a binary "parse" tree for the input formula φ. 构建布尔公式的二叉树
- The literals 文字 are leaves and connectives 连接符 are internal nodes.
- Introduce a variable y_i for the output of each internal node. （连接符的输出）
- Then, we rewrite the original formula φ as the AND of the root variable and a conjunction of clauses describing the operation of each node.（把原始的布尔公式写成根变元和子句的合取）

NP-Completeness of 3-CNF Satisfiability

$$\varphi = \Big((x_1 \to x_2) \vee \neg\big((\neg x_1 \leftrightarrow x_3) \vee x_4\big)\Big) \wedge \neg x_2$$

$$\begin{aligned}\varphi' = {} & y_1 \wedge \big(y_1 \leftrightarrow (y_2 \wedge \neg x_2)\big) \\ & \wedge \big(y_2 \leftrightarrow (y_3 \vee y_4)\big) \\ & \wedge \big(y_3 \leftrightarrow (x_1 \to x_2)\big) \\ & \wedge \big(y_4 \leftrightarrow \neg y_5\big) \\ & \wedge \big(y_5 \leftrightarrow (y_6 \vee x_4)\big) \\ & \wedge \big(y_6 \leftrightarrow (\neg x_1 \leftrightarrow x_3)\big)\end{aligned}$$



φ' is conjunction of clause φ'_i. Each clause φ'_i has at most 3 literals.

NP-Completeness of 3-CNF Satisfiability

Proof (cont'd):

- The second step converts each clause φ_i into CNF. 第二步将每个子句都变成合取范式CNF
- Construct a truth table for φ_i by evaluating all possible assignments to its variables. 首先对每个子句φ_i构建真值表
- Build a formula in disjunctive normal form (DNF, 析取范式) that is equivalent to $\neg\varphi_i$. 之后基于真值表构建析取范式
- Convert $\neg\varphi_i$ into CNF φ_i by using DeMorgan's laws (德·摩根法则) to complement all literals and switch OR and AND.

NP-Completeness of 3-CNF Satisfiability

- For example:

$$\varphi_2' = y_1 \leftrightarrow (y_2 \wedge \neg x_2)$$

- The DNF formula equivalent to $\neg\varphi_2{}'$ is

$(y_1 \wedge y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge \neg x_2) \vee (\neg y_1 \wedge y_2 \wedge \neg x_2)$

- Applying DeMorgan's laws, we get φ_2'':

$(\neg y_1 \vee \neg y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee x_2) \wedge (y_1 \vee \neg y_2 \vee x_2)$

y_1	y_2	x_2	$(y_1 \leftrightarrow (y_2 \wedge \neg x_2))$
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	1

NP-Completeness of 3-CNF Satisfiability

Proof (cont'd):

- The final step further transforms the formula so that each clause has exactly 3 distinct literals. 最后一步让每个子句都有3个文字
  - If φ_i'' has 3 distinct literals, then simply include φ_i'' as clauses of φ'''.
  - If φ_i'' has 2 distinct literals l_1 and l_2, that is, if $\varphi_i'' = (l_1 \vee l_2)$, then include $(l_1 \vee l_2 \vee p) \wedge (l_1 \vee l_2 \vee \neg p)$ as clauses of φ'''.
  - If φ_i'' has just 1 distinct literal l, then include $(l \vee p \vee q) \wedge (l \vee p \vee \neg q) \wedge (l \vee \neg p \vee q) \wedge (l \vee \neg p \vee \neg q)$ as clauses of φ'''.
- p and q can be arbitrary literal.

NP-Completeness of 3-CNF Satisfiability

Proof (cont'd):

- Thus, every formular can be transformed into 3-CNF φ''', and φ''' is satisfiable if and only if φ is satisfiable.
- This reduction algorithm is polynomial time.
- Therefore, 3-CNF is NPC.

Clique Problem 团问题

- Now, we have known two NPC problem: SAT and 3-CNF-SAT. We can use them to prove more NPC problems.
- A clique (团) is a complete subgraph of an undirected graph $G = (V, E)$.
  - Each pair of vertex in $V' \subseteq V$ is connected by an edge in E. 任意一对顶点之间都有边直接相连
  - The size of a clique is the number of vertices it contains. 一个团的大小指的是这个团里面的节点数量
- The clique problem is the optimization problem of finding a clique of maximum size in a graph. 团问题的优化问题指给定一个图，找一个团，使得其节点数量最多。
- As a decision problem, we ask simply whether a clique of a given size k exists in the graph. 团问题的判定问题是问到底指定大小的团是不是在图中存在

Clique Problem

- The brute force algorithm finds a 4-clique in this 7-vertex graph.
- Systematically check all $C(7,4) = 35$ subgraphs with 4 vertices for completeness.

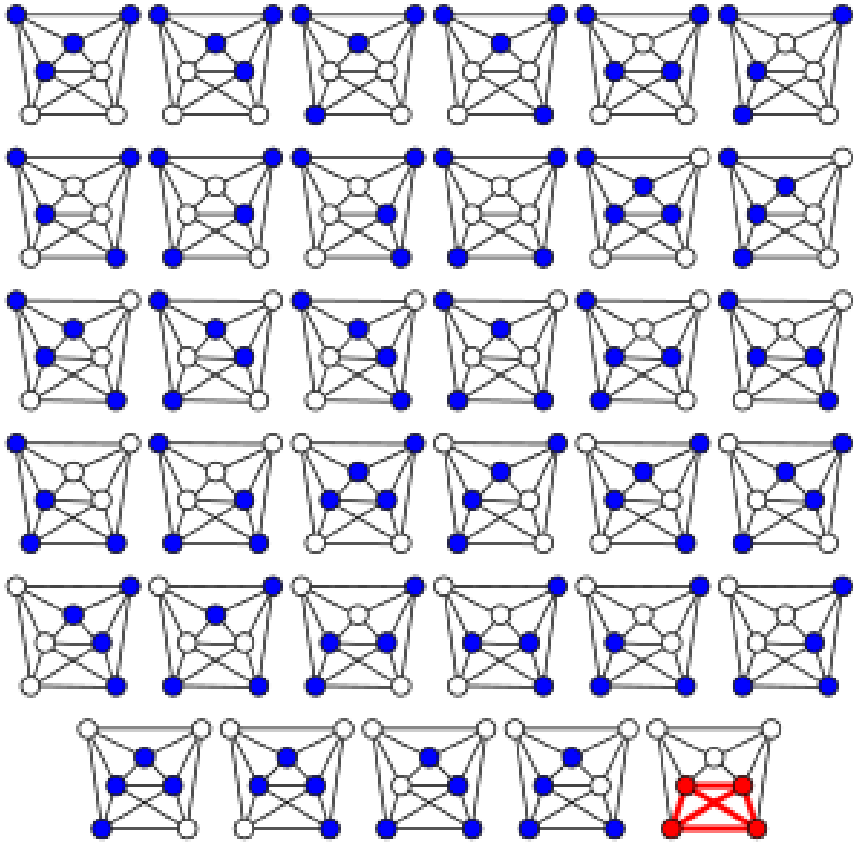


Image source: https://en.wikipedia.org/wiki/Clique_problem

Clique Problem

Theorem 11.6

Clique is NP-complete.

Proof:

- **Step 1**: We first prove clique is in NP.
- For a given graph $G = (V, E)$, we use the set $V' \subseteq V$ of vertices in the clique and a integer k as a certificate for G.
- Checking whether V' is a clique can be accomplished in polynomial time by checking whether $(u, v) \in E, \forall u, v \in V'$.
- Then, compare $|V'|$ with k and return True or False.

Clique Problem

Proof (cont'd):

- **Step 2**: We next prove that 3-CNF-SAT $\leq_P$ Clique, which shows that the clique problem is NP-hard.
- That we should be able to prove this result is somewhat surprising, since logical formulas seem to have little to do with graphs.
- We need to construct a reduction algorithm that transforms any instance of 3-CNF-SAT into an instance of clique.

Clique Problem

Proof (cont'd):

- Let φ be a Boolean formula in 3-CNF with k clauses: 这个CNF有k 个子句

$$\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_k$$

- For $r = 1,2, \ldots, k$, each clause C_r has exactly three distinct literals l_1^r, l_2^r and l_3^r:

$$C_r = l_1^r \vee l_2^r \vee l_3^r$$

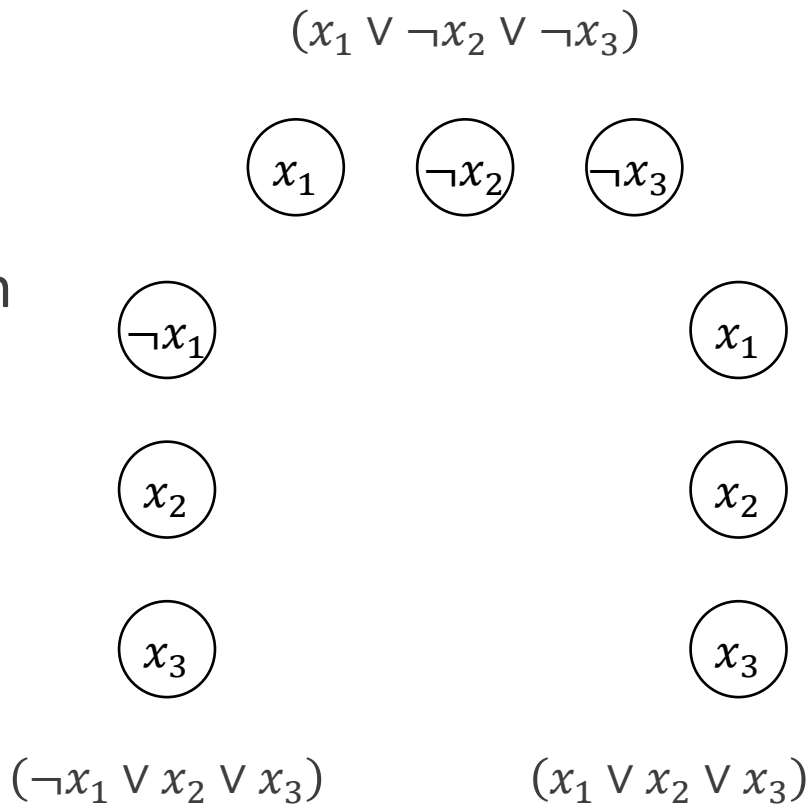
- Now, we construct a graph G such that φ is satisfiable if and only if G has a clique of size k. 构造一个图，φ可满足当且仅当G有大小为k的团

Clique Problem

Proof (cont'd):

- The graph $G = (V, E)$ is constructed as follows.
- For each clause $C_r = l_1^r \vee l_2^r \vee l_3^r$ in φ, we place a triple of vertices l_1^r, l_2^r and l_3^r into V.
- For example:

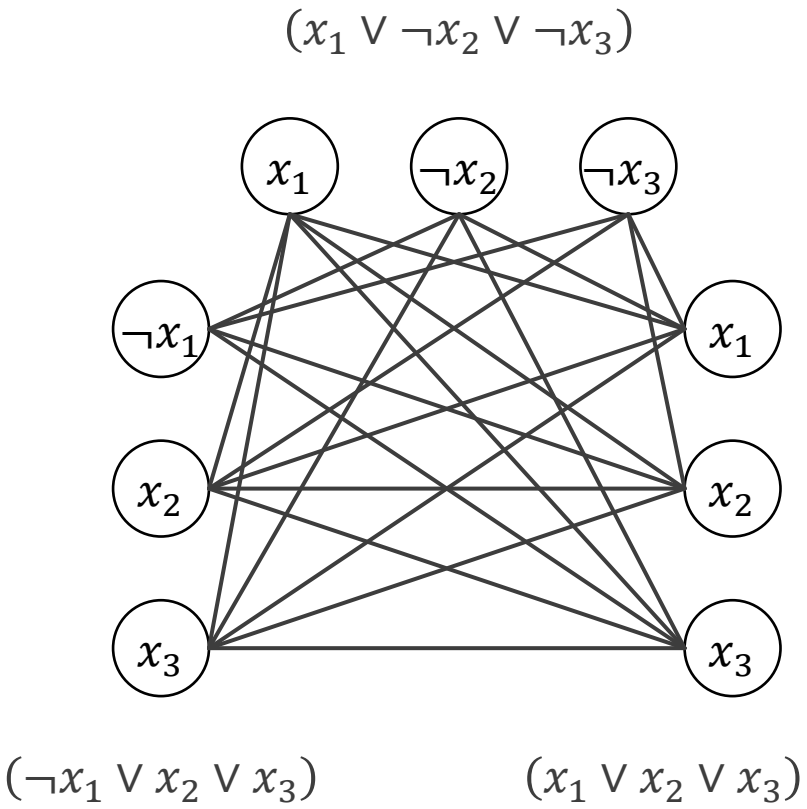
$$\varphi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$$



Clique Problem

Proof (cont'd):

- We put an edge between two vertices l_i^r and l_j^s if both of the following hold:
  - l_i^r and l_j^s are in different triples, that is, $r \neq s$.
  - l_i^r is not the negation of l_j^s.
- This graph can easily be computed from φ in polynomial time.



Clique Problem

Proof (cont'd):

- A satisfiable assignment is:

$$x_2 = 0, x_3 = 1$$

and x_1 cam be either 0 or 1. The corresponding 3-clique is:

$$\{\neg x_2, x_3, x_3\}$$

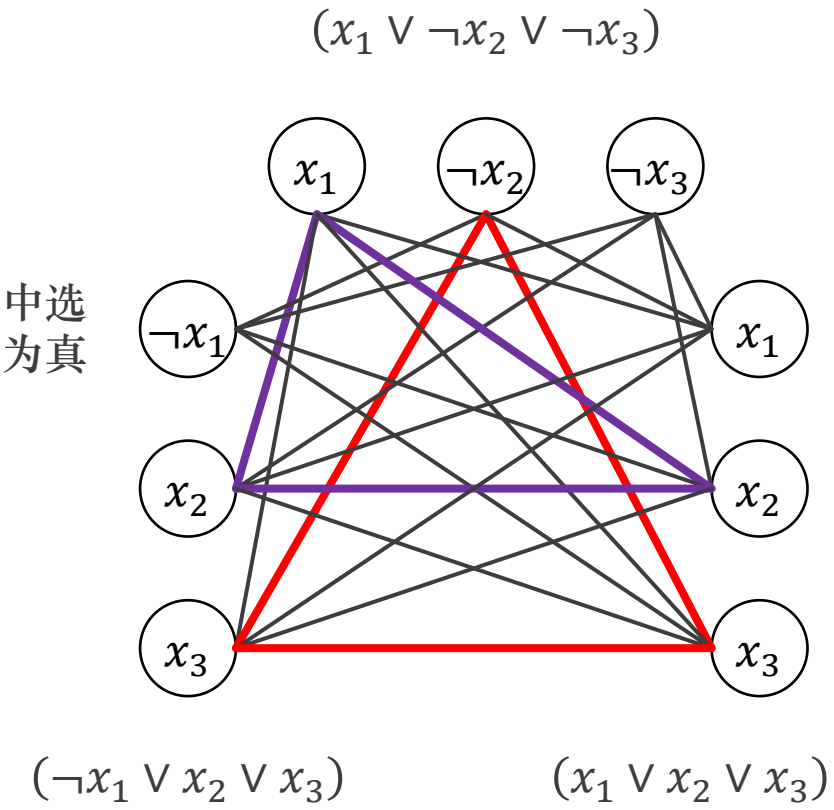
从不同子句中选取使得子句为真的文字

- Another satisfiable assignment is:

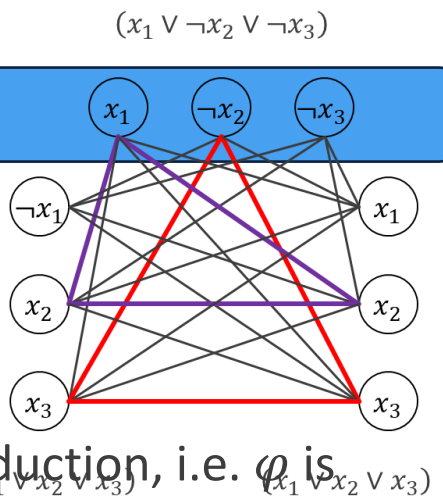
$$x_1 = 1, x_2 = 1$$

and x_3 cam be either 0 or 1. The corresponding 3-clique is:

$$\{x_1, x_2, x_2\}$$



Clique Problem



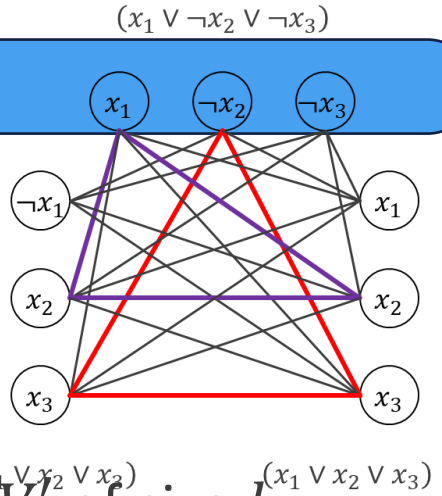
Proof (cont'd):

- We must show that this transformation of φ into G is a reduction, i.e. φ is satisfiable if and only if G has k-clique. 必须证明φ是可满足的，当且仅当G有K大小的团
- We first prove $\Longrightarrow$. Suppose that φ has a satisfying assignment.
- Then each clause C_r contains at least one literal that is assigned 1. 每个子句至少一个文字是真

$$\varphi = \overbrace{(x_1 \vee \neg x_2 \vee \neg x_3)}^{1} \wedge \overbrace{(\neg x_1 \vee x_2 \vee x_3)}^{1} \wedge \overbrace{(x_1 \vee x_2 \vee x_3)}^{1}$$

- Picking one such "true" literal from each clause yields a set V' of k vertices.
- It is easy to show that V' is a clique by the construction of G.（任意两个顶点
- 属于不同子句，赋值1不是文字的非，根据图的构造，是一个团）

Clique Problem



Proof (cont'd):

- Then we prove $\Longleftarrow$. Suppose that G has a clique V' of size k.
- No edges in G connect vertices in the same triple, and so V' contains exactly one vertex per triple. 同一个子句的文字，对应顶点在G中没有连接，V'包含了每个子句中的一个文字
- Assigning 1 to each literal is safe, because G contains no edges between complementary literals. 相反的文字不会在同一条边中出现，只需要对V'中文字进行赋值为1，那么整个φ取值为1
- Each clause is satisfied, and so φ is satisfied.

Vertex Cover Problem 顶点覆盖问题

- A vertex cover (顶点覆盖) of an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ such that if (u, v) is an edge of G, then either $u \in V'$ or $v \in V'$ (or both).

顶点覆盖问题中，任选一条边，他的两个顶点中，至少有一个在覆盖顶点V'中

- The size of a vertex cover is the number of vertices in it.
- The vertex-cover problem is to find a vertex cover of minimum size in a given undirected graph.
- We can prove the completeness of vertex cover problem by showing Clique $\leq_P$ VertexCover. 通过证明团问题可以归约到顶点覆盖问题，来证明他是NPC

A vertex cover

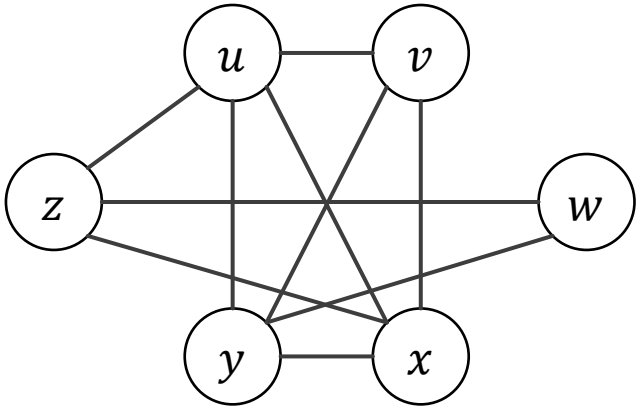
An minimum vertex cover

Image source: https://en.wikipedia.org/wiki/Vertex_cover

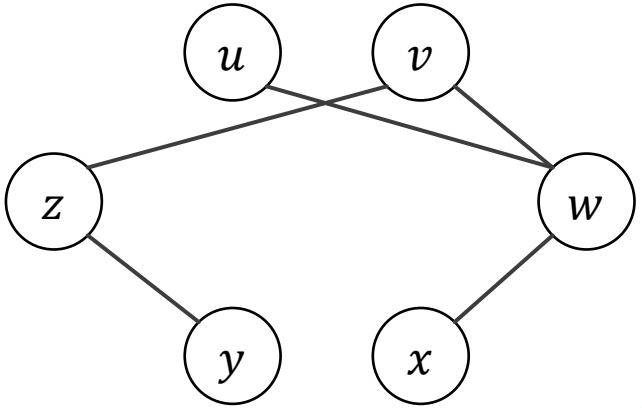
Vertex Cover Problem

- We first introduce the notion of complement graph (补图).
- Given an undirected graph $G = (V, E)$, we define the complement of G as $\bar{G} = (V, \bar{E})$, where $\bar{E} = \{(u, v): u, v \in V \text{ and } (u, v) \notin E\}$.

补图：顶点还是原来的顶点，但是边都是原来图中不存在的边



G



$\bar{G}$

Vertex Cover Problem

Proof:

- We first show that VertexCover ∈ NP.
- Suppose we are given a graph $G = (V, E)$ and an integer k.
- The certificate we choose is the vertex cover $V' \subseteq V$ itself.
- The verification algorithm affirms that $|V'| = k$, and then it checks:

$$u \in V' \text{ or } v \in V',\ \forall (u, v) \in E$$

- This verification can be performed straightforwardly in polynomial time.

Vertex Cover Problem

Proof (cont'd):

- The reduction algorithm takes as input an instance $\langle G, k \rangle$ of the clique problem.
- It computes the complement $\bar{G}$, which is easily done in polynomial time. 计算图的补图$\bar{G}$
- The output of the reduction algorithm is the instance $\langle \bar{G}, |V| - k \rangle$ of the vertex-cover problem. 对团问题进行约简，约简为其补图的顶点覆盖问题
- To complete the proof, we show that this transformation is indeed a reduction:

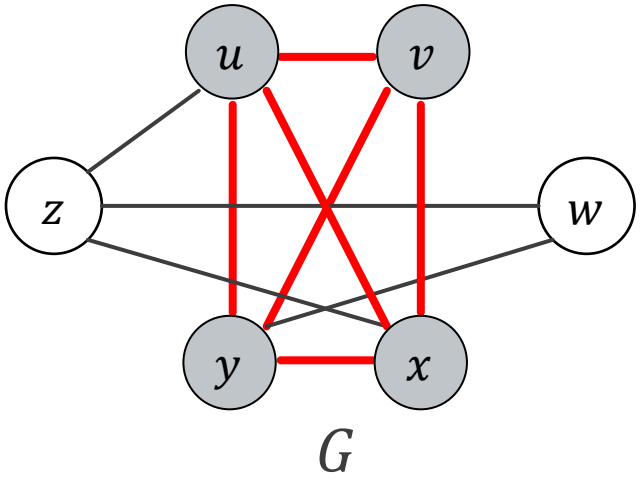
$\langle G,k \rangle$ of the clique problem

$\langle \bar{G}, |V| - k \rangle$ of the vertex-cover problem

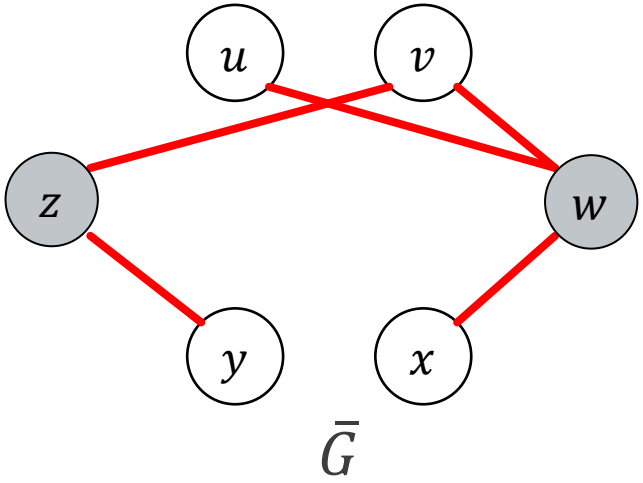
只需证明：图G有一个大小k的团当且仅当图$\bar{G}$有一个大小$|V| - k$的顶点覆盖

Vertex Cover Problem

Proof (cont'd):



G



$\bar{G}$

Clique size = 4

Vertex cover = 6 – 4 = 2

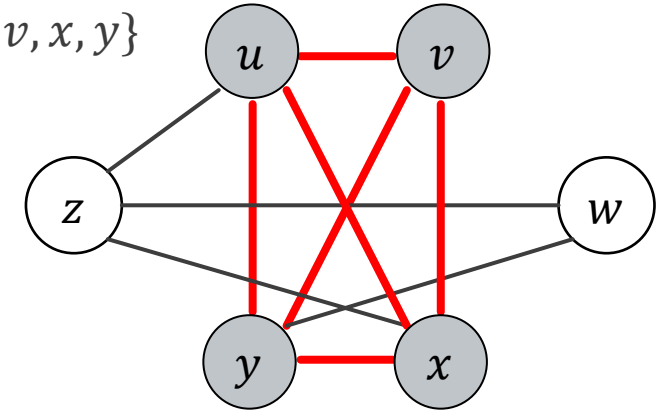
只需证明：图G有一个大小k的团当且仅当图$\bar{G}$有一个大小$|V| - k$的顶点覆盖

Vertex Cover Problem

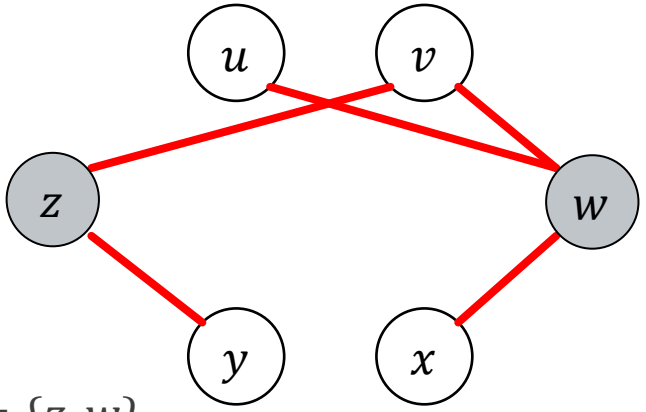
Proof (cont'd):

- We first prove $\Longrightarrow$. Suppose that G has a clique $V' \subseteq V$ with $|V'| = k$.
- Since every pair of vertices in V' is connected by an edge of E, if $(u, v) \notin E$, which implies that at least one of u or v does not belong to V'. 在团V'中，每个点都通过E中边相连，如果有一条边(u, v) 不属于E，那么意味着u或者v不属于V'

$V' = \{u, v, x, y\}$



G



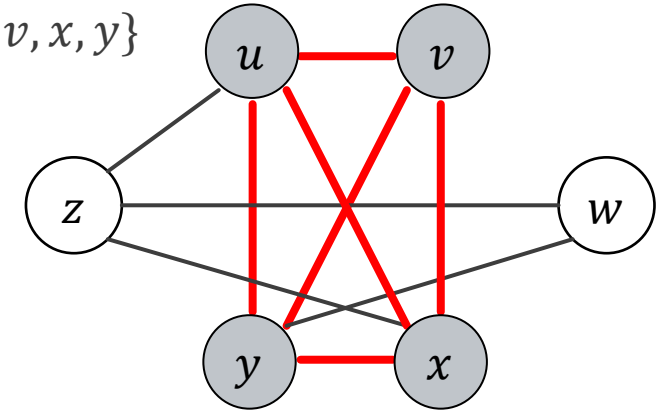
$V - V' = \{z, w\}$

$\bar{G}$

只需证明：图G有一个大小k的团当且仅当图$\bar{G}$有一个大小$|V| - k$的顶点覆盖

Vertex Cover Problem

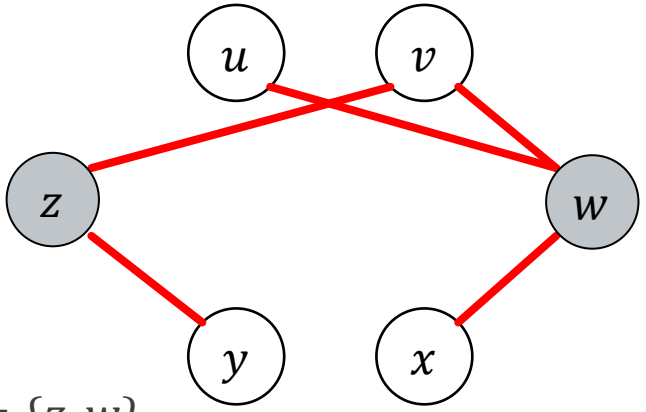
$V' = \{u, v, x, y\}$



G

Proof (cont'd):

- We get: $(u, v) \in \bar{E} \Rightarrow u \in V - V'$ or $v \in V - V'$
- It means that every edge $(u, v) \in \bar{E}$ is covered by $V - V'$. 这个就是顶点覆盖
- Hence, the set $V - V'$,which has size $|V| - k$, forms a vertex cover for $\bar{G}$.



$V - V' = \{z, w\}$

$\bar{G}$

只需证明：图G有一个大小k的团当且仅当图$\bar{G}$有一个大小$|V| - k$的顶点覆盖

Vertex Cover Problem

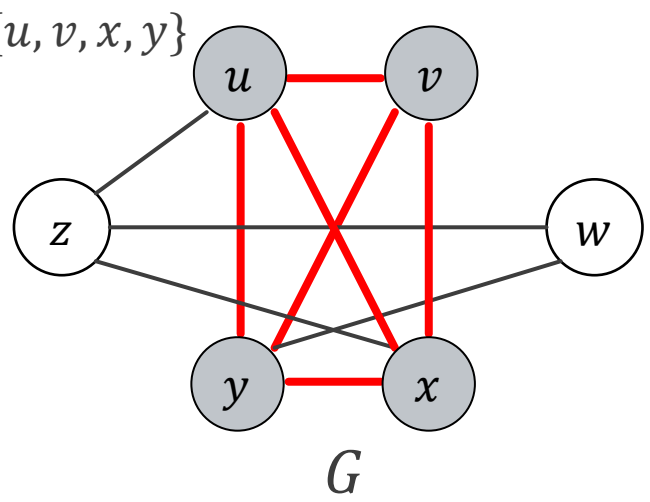
Proof (cont'd):

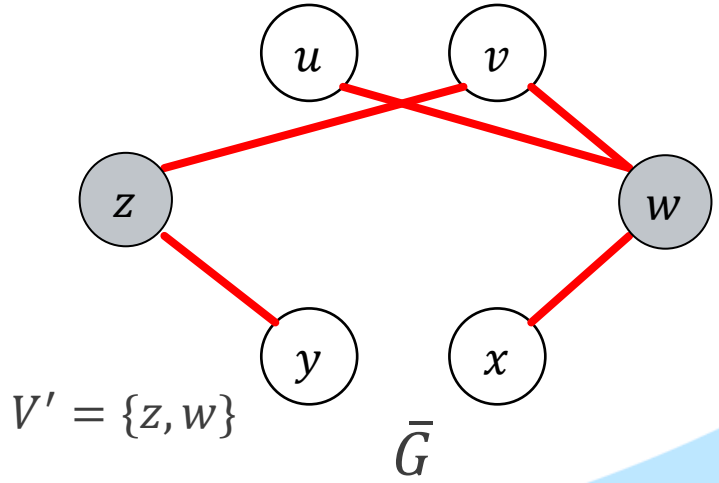
- Then we prove $\Longleftarrow$. Suppose that $\bar{G}$ has a vertex cover $V' \subseteq V$, where $|V'| = |V| - k$.
- Then, $\forall u, v \in V$:

$$(u, v) \in \bar{E} \Rightarrow u \in V' \text{ or } v \in V'$$

- The contrapositive of this implication is:

$$(u, v) \notin \bar{E} \Leftarrow u \notin V' \text{ and } v \notin V'$$

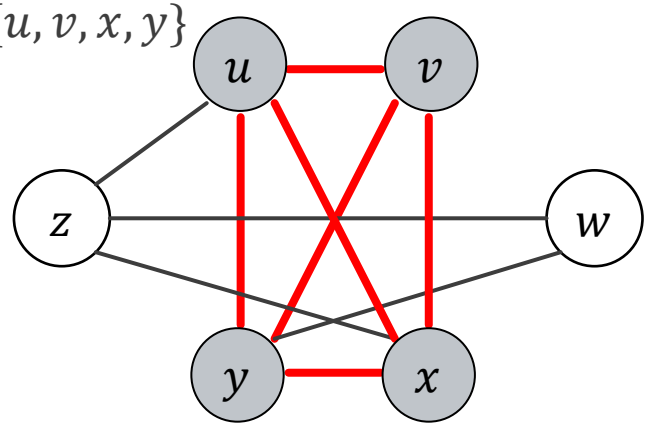




只需证明：图G有一个大小k的团当且仅当图$\bar{G}$有一个大小$|V| - k$的顶点覆盖

Vertex Cover Problem

$V - V' = \{u, v, x, y\}$



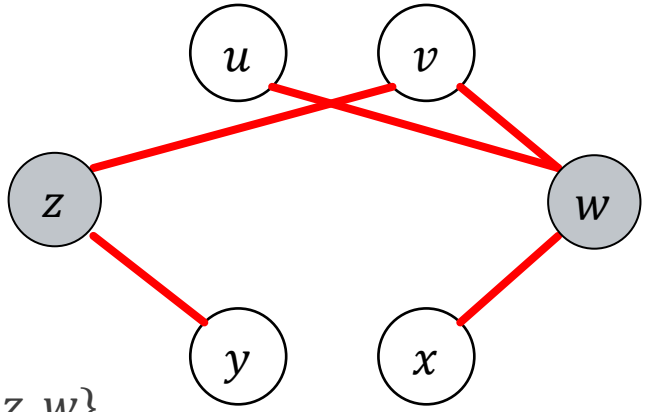
G

Proof (cont'd):

$$(u, v) \notin \bar{E} \Leftarrow u \notin V' \text{ and } v \notin V'$$

$$(u, v) \in E \Leftarrow u \in V - V' \text{ and } v \in V - V'$$

- In other words, $V - V'$ is a clique, and it has size $|V| - (|V| - k) = k$.

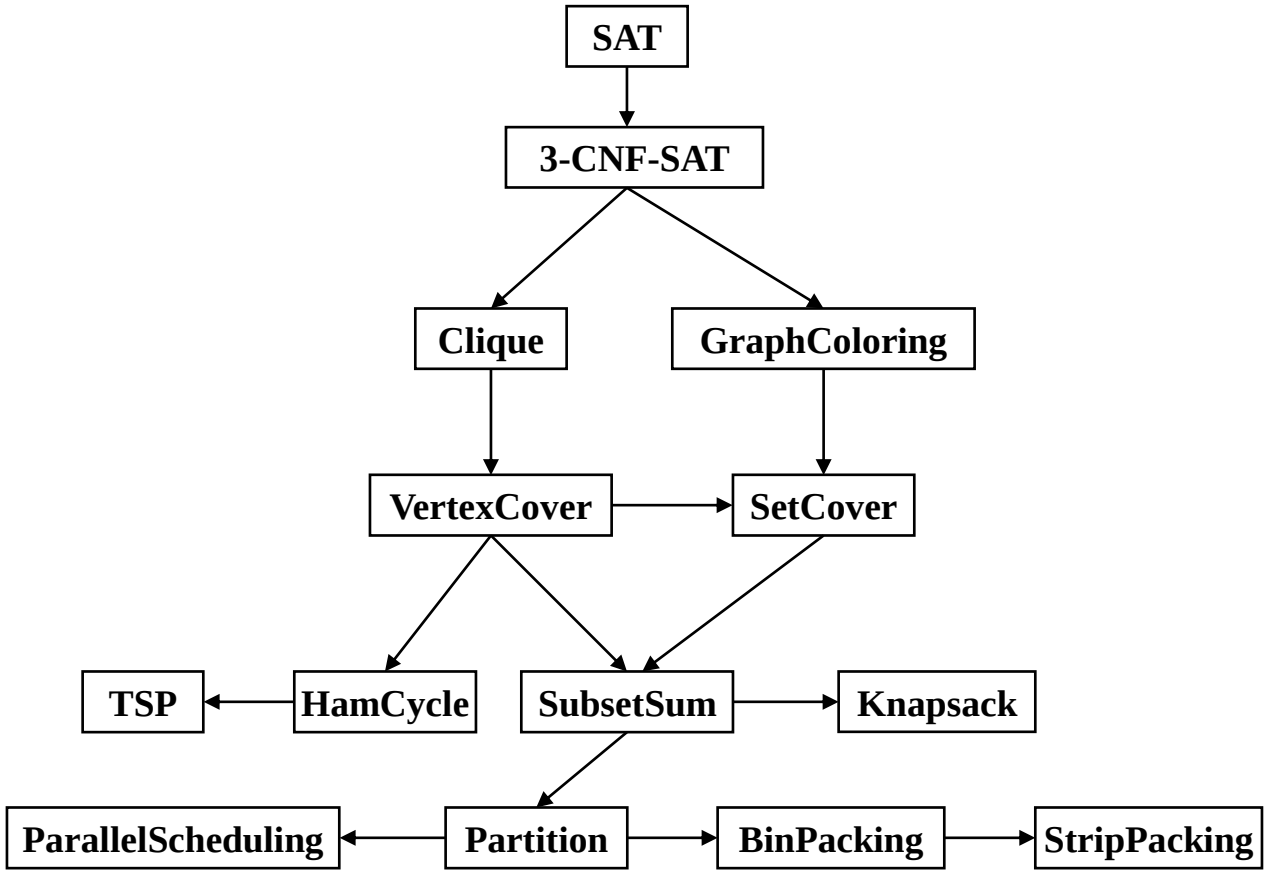


$V' = \{z, w\}$

$\bar{G}$

只需证明：图G有一个大小k的团当且仅当图$\bar{G}$有一个大小$|V| - k$的顶点覆盖

NP-Completeness Proof Graph



Classroom Exercise （课堂练习）

- The subset-sum problem (子集和问题) is defined as follows:
  - Given a sequence of integers $A = \{a_1, \dots, a_n, t\}$, determine whether there is a subset of the integers such that the sum is equal to t.
  - For example, given $\{a_1, a_2, a_3, a_4, a_5, t\} = \{1, 6, 4, 3, 2, 8\}$, we have $a_1 + a_3 + a_5 = 8$.
- The partition problem (划分问题) is defined as follows:
  - Given a sequence of integers $A = \{a_1, \dots, a_n\}$, determine whether there is a partition into two subsets such that their sums are equal.
  - For example, given $\{a_1, a_2, a_3, a_4, a_5\} = \{1, 6, 4, 3, 2\}$, we have $a_1 + a_3 + a_4 = a_2 + a_5 = 8$.
- If we know that subset-sum problem is NPC, prove that partition problem is NPC. 如果我们知道子集和问题是NPC，证明划分问题也是NPC

Classroom Exercise

Proof:

- **Step 1**: Show that partition decision problem is in NP by checking its verification stage in polynomial-time or not.
  - Input: A set of index S.
  - Output: Yes or No.
    - $sum_1 = \sum_{i \in S} a_i, sum_2 = \sum_{i=1}^{n} a_i - sum_1$.
    - If $sum_1 == sum_2$, return Yes; else return No.
- It is obviously polynomial-time.

Classroom Exercise

Proof (cont'd):

- **Step 2**: Prove that subset-sum $\leq_P$ partition.
- Compare the two problems:
  - Subset-Sum: Given $(a_1, \ldots, a_n, t)$, where t and all a_i are integers, whether there exists an answer $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} a_i = t$.
  - Partition: Given $(a_1, \ldots, a_n)$, where all a_i are integers, whether there exists an answer $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} a_i = \sum_{j \notin S} a_j$.

Classroom Exercise

Proof (cont'd):

- **Transform** every instance of subset-sum to an instance of partition.
- Let $x = (a_1, \dots, a_n, t)$ be an instance of subset-sum and $a = \sum_{i=1}^{n} a_i$. We define the transformation algorithm as

$$f(x) = (a_1, a_2, \dots, a_n, a_{n+1})$$

  where $a_{n+1} = 2t - a$.

- It is clear that this transform can be done in polynomial time.
- Then, we want to show that:

The answer S to subset-sum problem is "yes" for x

$\Leftrightarrow$ The answer S to partition problem is "yes" for $f(x)$.

Classroom Exercise

Proof (cont'd):

- Prove $\Longrightarrow$:
  - Let the answer $S \subseteq \{1,2,\ldots,n\}$ to subset-sum problem is "yes" for x such that $\sum_{i\in S} a_i = t$.
  - Let $T = \{1,2,\ldots,n+1\} - S$, we have

$$\sum_{j\in T} a_j = a + a_{n+1} - t = a + 2t - a - t = t = \sum_{i\in S} a_i .$$

  - Because $\sum_{i\in S} a_i = \sum_{j\in T} a_j = \sum_{j\notin S} a_j$, the answer S to partition problem is also "yes" for $f(x)$.

Proof (cont'd):

- Prove $\Leftarrow$:
  - If there exists the solution $S \subseteq \{1,2,\ldots,n+1\}$ to partition problem is also "yes" for $f(x)$, such that letting $T = \{1,2,\ldots,n+1\} - S$ we have

$$\sum_{i \in S} a_i = \sum_{j \in T} a_j = \frac{a + (2t - a)}{2} = t.$$

  - Without loss of generality, assume that $n+1 \in T$, then we have $S \subseteq \{1,2,\ldots,n\}$ and $\sum_{i \in S} a_i = t$.
  - Because $\sum_{i \in S} a_i = t$, the answer S to subset-sum problem is also "yes" for $f(x)$.

Conclusion

After this lecture, you should know:

- What is a polynomial-time algorithm.
- What is a decision problem.
- What are P and NP.
- How a problem can be reduced to another problem.
- What is an NP-complete problem.
- What is an NP-hard problem.
- How to prove NP-completeness of a problem.

Homework

Page 215-217

11.2

11.3

11.4

11.7

11.16

谢谢

有问题欢迎随时跟我讨论