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 5. Polynomial-time Reductions

- The general notion of a reduction

  Cook-reduction; Karp-reduction; Levin reduction
- Reducing optimization problems to search problems
- Self reducibility of search problems
- Definitions and existence of NP-completeness

$$P \overset{?}{=} NP$$

This is a major unsolved problem in computer science.

Clay Mathematics Institute Millennium Prize Problems(千禧年大奖难题):

- P versus NP
- The Hodge conjecture
- The Poincar é conjecture - solved, by Grigori Perelman
- The Riemann hypothesis
- Yang – Mills existence and mass gap
- Navier – Stokes existence and smoothness
- The Birch and Swinnerton-Dyer conjecture.

The general notion of a reduction

在P与NP问题上的一个重大进展是20世纪70年代初由Stephen Cook和Leonid Levin完成的。他们发现NP中的某些问题的复杂性与整个类的复杂性相关联。这些问题中任何一个如果存在多项式时间按算法，那么所有的NP问题都是多项式时间可解的。这些问题称为NP完全的(NP-complete)。

理论意义：证明P不等于NP就可以集中到一个NP完全问题上。

实践意义：NP完全现象可以防止为某一具体问题浪费时间，寻找本不存在的多项式时间算法。

证明一个NP问题是NPC，需要用到多项式时间规约的概念。

Stephen Cook(1939-) and Leonid Levin(1948-)

关于Stephen Cook(史蒂芬库克):

1939年12月14日生于纽约州的布法罗(Buffalo)，1961年从University of Michigan获得其学士学位，于1962年和1966年从哈佛大学分别获得其硕士与博士学位。1966年到1970年，Stephen在加州Berkeley分校担任助理教授职务。1970年，Stephen加盟多伦多大学并工作直到现在。他是NP完全性理论的奠基人，1971年，在他的论文《The Complexity of Theorem Proving Procedures》，他整理了NP完备性的目标，亦产生了库克定理——布尔可满足性问题是NP完备的证明。

1982年，库克得到图灵奖.

关于Leonid Levin(莱文或列文, November 2, 1948):

He obtained his master's degree at Moscow University in 1970 where he studied under Andrey Kolmogorov and completed the Candidate Degree academic requirements in 1972.

Levin and Stephen Cook independently discovered the existence of NP-complete problems. This NP-completeness theorem, often called the Cook-Levin Theorem.

He emigrated to the U.S. in 1978 and also earned a Ph.D. at the Massachusetts Institute of Technology (MIT) in 1979.

Levin's journal article on this theorem was published in 1973.

Levin was awarded the Knuth Prize in 2012 for his discovery of NP-completeness and the development of average-case complexity.

He is currently a professor of computer science at Boston University.

Reductions are procedures that use "functionally specified" subroutines. That is, the functionality of the subroutine is specified, but its operation remains unspecified and its running time is counted at unit cost. Analogously to algorithms, which are modeled by Turing machines, reductions can be modeled as oracle Turing machines.

A reduction solves one computational problem by using oracle calls to another computational problem.

The key property of efficient reductions is that they allow for the transformation of efficient implementations of the subroutine into efficient implementations of the task reduced to it.

We say that two problems are **computationally equivalent** if they are reducible to one another.

We say that a class of problems, $\mathcal{C}$, is reducible to a problem Π if every problem in $\mathcal{C}$ is reducible to Π.

We say that the class $\mathcal{C}$ is reducible to the class $\mathcal{C}'$ if for every $\Pi \in \mathcal{C}$, there exists $\Pi' \in \mathcal{C}'$ such that Π is reducible to Π'.

Using an oracle

- *An* oracle machine *is a Turing machine with an additional tape, called the* oracle tape, *and two special states, called* oracle invocation *and* oracle spoke.
- *The* computation of the oracle machine M on input x and access to the oracle $f : \{0,1\}^* \to \{0,1\}^*$ *is defined based on the successive configuration function. For configurations with state different from* oracle invocation *the next configuration is defined as usual. Let* γ *be a configuration in which the machine's state is* oracle invocation *and suppose that the actual contents of the oracle tape is* q *(i.e.,* q *is the contents of the maximal prefix of the tape that holds bit values).*[20] *Then, the configuration following* γ *is identical to* γ*, except that the state is* oracle spoke, *and the actual contents of the oracle tape is* $f(q)$. *The string* q *is called* M*'s* query *and* $f(q)$ *is called the* oracle's reply.
- *The output of* M *on input* x *when given oracle access to* f *is denote* $M^f(x)$.

预言机(谕示机)，是一个用来研究决定型问题的抽象电脑。可以被视为一个多了个黑盒子(神谕)的图灵机，这个黑盒子的功能是可以在单位时间之内解答特定问题。神谕可以解答的问题，根据给定可以是任何复杂度类之内的问题。甚至可以使用非决定型问题这一类，像是停机问题。

一个谕示机可以执行很多对一般图灵机来说很特殊的操作，并且可以借由询问神谕来获得"x是否在A内?"这种特定形式问题的解答。

除了图灵机之外，一个谕示机还包含了:

一个神谕纸带(oracle tape)，代表了一个可以计算神谕集合(oracle set)A的函式；一个神谕读取头(oracle head)，它不能写入，而且跑的纸带是神谕纸带。

Cook reduction

Note that when reducing to a decision problem, the oracle is determined as the single valid solver of the decision problem, i.e., the function $f : \{0,1\}^* \to \{0,1\}$ solves the decision problem of membership in S if, for every x, it holds that $f(x) = 1$ if $x \in S$ and $f(x) = 0$ otherwise.

When reducing to a search problem, there may be many different valid solvers, i.e., the function $f : \{0,1\}^* \to \{0,1\}^* \cup \{\perp\}$ solves the search problem of R if, for every x, it holds that $f(x) \in R(x)$ if $x \in S_R$ and $f(x) = \perp$ otherwise.

Definition

(Cook reduction): A problem Π is Cook-reducible to a problem Π'. if there exists a polynomial-time oracle machine M such that for every function f that solves Π' it holds that M^f solves Π, where $M^f(x)$ denotes the output of M on input x when given oracle access to f.

Note that: Cook reduction所考虑的问题可以是search型的也可以是decision型的。

Theorem

Every search problem in $\mathcal{PC}$ is Cook-reducible to some decision problem in $\mathcal{NP}$.

Cook-reductions have following properties:

- If Π is Cook-reducible to Π' and Π' is solvable in polynomial-time then so is Π.
- Cook-reductions are transitive.
- If Π is solvable in polynomial-time then it is Cook-reducible to any problem Π'.

Karp reduction

A Karp-reduction reduction is a special case of a reduction, from a decision problem to a decision problem.

Definition

(Karp reduction): A polynomial-time computable function f is called a **Karp reduction of** S **to** S' if, for every x, it holds that $x \in S$ if and only if $f(x) \in S'$.

Note: Each decision problem is Cook-reducible to its complement, some decision problems are not Karp-reducible to their complement.

Levin reduction

Karp-reductions may be augmented in order to handle reductions of search problems to search problems.

Definition

(Levin reduction): A pair of polynomial-time computable functions, f and g, is called a **Levin reduction of R to R'** if f is a Karp reduction of $S_R = \{x : \exists y s.t. (x, y) \in R\}$ to $S_{R'} = \{x' : \exists y' s.t. (x', y') \in R'\}$ and for every $x \in S_R$ and $y' \in R'(f(x))$ it holds that $(x, g(x, y')) \in R$, where $R'(x') = \{y' : (x', y') \in R'\}$.

The function f preserves the existence of solution, that is, for any x, it holds that $R(x) \neq \emptyset$ if and only if $R'(f(x)) \neq \emptyset$. As for the second function g, it maps any solution y' for the reduced instance $f(x)$ to a solution for the original instance x (where this mapping may also depend on x).

Reducing optimization problems to search problems

In such a case, one may be interested in finding a solution that has value exceeding some threshold or finding a solution of maximum value.

There are two different objectives(i.e., exceeding a threshold and optimizing) giving rise to two different search problems related to the same relation R. Specifically, for a binary relation R and a value function $f : \{0,1\}^* \times \{0,1\}^* \to \mathbb{R}$, we consider two search problems:

1. Exceeding a threshold: Given a pair (x, v), the task is to find $y \in R(x)$ such that $f(x, y) \geq v$, where $R(x) = \{y : (x, y) \in R\}$. That is, we are actually referring to the search problem of the relation

   $$R_f \stackrel{\text{def}}{=} \{(< x, v >, y) : (x, y) \in R \wedge f(x, y) \geq v\} \quad (*)$$

   where $< x, v >$ denotes a string that encodes the pair (x, v).
2. Maximization: Given x the task is to find $y \in R(x)$ such that $f(x, y) = v_x$, where v_x is the maximum value of $f(x, y')$ over all $y' \in R(x)$. That is, we are actually referring to the search problem of the relation

Theorem

For every polynomial-time computable $f : \{0,1\}^* \times \{0,1\}^* \to \mathbb{R}$ *and a polynomially bounded binary relation* R, *let* R_f *and* R'_f *be as in (*) and (**), respectively. Then the search problems of* R_f *and* R'_f *are computationally equivalent.*

Proof: The search problem of R_f is reduced to the search problem of R'_f by finding an optimal solution (for the given instance) and comparing its value to the given threshold value. That is, we construct an oracle machine that solves R_f by making a single query to R'_f. Specifically, on input (x, v), the machine issues the query x (to a solver for R'_f), obtaining the optimal solution y (or an indication $\perp$ that $R(x) = \emptyset$), computes $f(x, y)$, and returns y if $f(x, y) \geq v$. Otherwise (i.e., either $y = \perp$ or $f(x, y) < v$), the machine returns an indication that $R_f(x, v) = \emptyset$.

Turning to the opposite direction, we reduce the search problem of R_f to the search problem of R'_f by first finding the optimal value $v_x = \max_{y \in R(x)}\{f(x, y)\}$ (by binary search on its possible values), and next finding a solution of value v_x. In both steps, we use oracle calls to R_f. For simplicity, we assume that f assigns positive integer values, and let $\ell = \text{poly}(|x|)$ be such that $f(x, y) \leq 2^\ell - 1$ for every $y \in R(x)$. Then, on input x, we first find $v_x = \max\{f(x, y) : y \in R(x)\}$, by making oracle calls of the form $\langle x, v \rangle$. The point is that $v_x < v$ if any only if $R_f(\langle x, v \rangle) = \emptyset$, which in turn is indicated by the oracle answer $\perp$ (to the query $\langle x, v \rangle$). Making ℓ queries, we determine v_x (see Exercise 2.9). Note that in case $R(x) = \emptyset$, all answers will indicate that $R_f(\langle x, v \rangle) = \emptyset$, which we treat as if $v_x = 0$. Finally, we make the query (x, v_x), and halt returning the oracle's answer (which is $y \in R(x)$ such that $f(x, y) = v_x$ if $v_x > 0$ and an indication that $R(x) = \emptyset$ otherwise). ■

Self-reducibility of search problems

For many natural relations R, the question of whether or not the search problem of R is efficiently solvable (i.e., is in $\mathcal{PF}$) is equivalent to the question of whether or not the "decision problem implicit in R" (i.e., $S_R = \{x : \exists y, s.t. (x, y) \in R\}$ is efficiently solvable (i.e., is in $\mathcal{P}$).

A problem is said to be self reducible if the search problem reduces (by Cook reduction) in polynomial time to decision problem. In other words, there is an algorithm to solve the search problem that has polynomially many steps, where each step is either (A) A conventional computational step, or (B) A call to subroutine solving the decision problem.

Definition

(the decision implicit in a search and self-reducibility:) The decision problem implicit the search problem of R is deciding membership in the set $S_R = \{x : R(x) \neq \emptyset\}$, where $R(x) = \{y : (x, y) \in R\}$. The search problem of R is called self-reducible if it can be reduced to the decision problem of S_R.

Note that the search problem of R and the problem of deciding membership in S_R refer to the same instances: The search problem requires finding an adequate solution (i.e., given x find $y \in R(x)$), whereas the decision problem refers to the question of whether such solutions exist (i.e., given x, determine whether or not $R(x)$ is non-empty).

Thus, S_R is really the "decision problem implicit in R", because it is a decision problem that one implicitly solves when solving the search problem of R.

Indeed, for any R, the decision problem of S_R is easily reducible to the search problem for R(and if R is in $\mathcal{PC}$ then S_R is in $\mathcal{NP}$). It follows that if a search problem R is self-reducible then it is computationally equivalent to the decision problem S_R.

布尔电路可满足性问题(Boolean satisfiability problem; CSAT): 对于一个确定的逻辑电路，是否存在一种输入使得输出为true(或者说，给定一个Boolean formula，判定是否存在所有的变量一个赋值，使得该布尔公式在这个赋值下的值是1).

布尔公式可满足性问题: SAT。

合取范式的可满足性问题(CNF-SAT):

任何布尔表达式，都可以化为合取范式的形式，即化为：() and () and () ...and () 其中括号里面的是用析取符号or 连接的变量或者变量的非的形式。我们一般称，变量或者变量的非为“文字”,而括号里的叫做子句。

3元合取范式的可满足性问题(3-SAT): 就是化为合取范式后，每个子句最多有3个文字的可满足性问题。

(R_{SAT} is self-reducible:) The search problem of R_{SAT} is reducible to SAT.

Proof: Relation R_{SAT}: that is, $(\phi, \tau) \in R_{\mathsf{SAT}}$ if τ is a satisfying assignment to the formulae ϕ. The decision problem implicit in R_{SAT} is indeed SAT. Note that R_{SAT} is in $\mathcal{PC}$.

We present an oracle machine that solves the search problem of R_{SAT} by making oracle calls to SAT.

Given a formula ϕ, we find a satisfying assignment to ϕ(in case such an assignment exists) as follows: First, we query SAT on ϕ itself, and return an indication that there is no solution if the oracle answer is 0 (indicating $\phi \notin$ SAT. Otherwise, we let τ, initiated to the empty string, denote a prefix of a satisfying assignment of ϕ. We proceed in iterations extend τ by one bit. This is done as follows: First we derive a formula, denoted ϕ', by setting the first $|\tau|+1$ variables of ϕ according to the values $\tau 0$. We then query SAT on ϕ' (which means that we ask whether or not $\tau 0$ is a prefix of a satisfying assignment of ϕ). If the answer is positive then we set $\tau \leftarrow \tau 0$ else we set $\tau \leftarrow \tau 1$. This procedure relies on the fact that if τ is a prefix of a satisfying assignment of ϕ and $\tau 0$ is not a prefix of a satisfying assignment of ϕ then $\tau 1$ must be a prefix of a satisfying assignment of ϕ.

We wish to highlight a key point that has been blurred in the foregoing description. Recall that the formula ϕ' is obtained by replacing some variables by constants, which means that ϕ' *per se* contains Boolean variables as well as Boolean constants. However, the standard definition of SAT disallows Boolean constants in its instances.[6] Nevertheless, ϕ' can be simplified such that the resulting formula contains no Boolean constants. This simplification is performed according to the straightforward Boolean rules: That is, the constant `false` can be omitted from any clause, but if a clause contains only occurrences of the constant `false` then the entire formula simplifies to `false`. Likewise, if the constant `true` appears in a clause then the entire clause can be omitted, but if all clauses are omitted then the entire formula simplifies to `true`. Needless to say, if the simplification process yields a Boolean constant then we may skip the query, and otherwise we just use the simplified form of ϕ' as our query. ■

Self-reducibility of NPC

NP-completeness of a problem Π combines two conditions:

- Π is in the class (i.e., Π being in $\mathcal{NP}$ or $\mathcal{PC}$, depending on whether Π is a decision or a search problem).
- Each problem in the class is reducible to Π. This condition is called NP-hardness.

Definition

(NP-completeness of decision problem, restricted notion) A set S is $\mathcal{NP}$-complete if it is in $\mathcal{NP}$ and every set in $\mathcal{NP}$ is Karp-reducible to S.

Definition

(NP-completeness of search problem, restricted notion) A binary relation R is $\mathcal{PC}$-complete if it is in $\mathcal{PC}$ and every relation in $\mathcal{PC}$ is Levin-reducible to R.

All search problems that refer to finding NP-witnesses for any NP-complete decision problem are self-reducible.

Theorem

For every R in $\mathcal{PC}$ such that S_R is $\mathcal{NP}$-complete, the search problem of R is reducible to deciding membership in S_R.

Proof: First, reduce the search problem of R to deciding $S'_R = \{(x, y') : \exists y''\ s.t.(x, y'y'') \in R\}$.

Then, a reduction of S_R to S'_R.

It is well known in worst-case complexity that the hardness of search and decision versions of NP-complete problems are equivalent. Namely, if any NP-complete problem has an efficient decision algorithm (on all instances), then not only does all of NP have efficient decision algorithms, but also all of NP have efficient search algorithms.

NP完全问题的研究思路：

1，给出NPC的定义；

2，证明NPC存在；

3，依照定义给出一个具体的NPC问题（实际上就是CSAT）；

4，利用规约定理证明更多的NPC：如果A是一个NPC，B是一个NP问题，如果可以证明通过调用B就可以求解A的话，那么B也是一个NPC.

S. A. Cook. The complexity of theorem proving procedures, Proceedings, Third Annual ACM Symposium on the Theory of Computing. New York: ACM. 1971: 151-158.

NPC问题，又称NP完全问题或NP完备问题，是NP中最难的问题。

因此NP完备问题应该是最不可能被多项式归约为P问题的集合。

如果任何一个NPC问题多项式时间可解，则P=NP。

Thank You Very Much!