AI2615 算法设计与分析 2020-2021 春季学期

Lecture 16 – Fine-Grained Complexity

2021 年 6 月 11 日

Lecturer: 张驰豪 *Scribe:* 陶表帅

In the previous two lectures, we have seen problems that are solvable in polynomial time (P) and those problems that are unlikely to be solved in polynomial time (NP-complete). In this lecture, we deal with problems in P, and we will see results like "a problem admits an $O(n^2)$ algorithm, but unlikely to be solved in $O(n^{2-\varepsilon})$ time (for any $\varepsilon > 0$)." This area is called *fine-grained complexity.*

When we are considering if a problem can be solved in a polynomial time (e.g., P versus NP), different computation models normally do not make a difference, and we can assume we are using a Turing machine. However, when we are dealing with fine-grained complexity, we need to specify a particular computation model, and we will use the word-RAM model in this lecture.

To show that a problem can be solved in $O(n^2)$ time but not in $O(n^{2-\varepsilon})$ time, we first need to assume a complexity hypothesis. In previous two lectures, we have used the hypothesis P $\neq$ NP to show certain problems are hard. Can we use the hypothesis P $\neq$ NP to show a result in fine-grained complexity? This turns out to be quite challenging.

To use the hypothesis P $\neq$ NP, we want to reduce a problem A from a NP-complete problem, say, SAT. To show that A can be solved in $O(n^2)$ time but not in $O(n^{2-\varepsilon})$ time, we need to prove A can be solved in $O(n^{2-\varepsilon})$ time if and only if SAT is in P. Let $\mathcal{R}$ be the reduction, and T be an algorithm for problem A. This would imply $T(\mathcal{R}(x))$ is computable in a polynomial time if T can be computed in $O(n^{2-\varepsilon})$ time, and $T(\mathcal{R}(x))$ is not computable in a polynomial time if T can only be computed in $O(n^2)$ time. We need to somehow amplify the gap between $n^{2-\varepsilon}$ and n^2 to a gap between a polynomial function and an exponential function. This is unlikely. Therefore, those results in find-grained complexity are usually based on hypotheses stronger than P $\neq$ NP.

1 (Strong) Exponential-Time Hypothesis

1.1 Exponential-Time hypothesis

Hypothesis 1 (Exponential-Time Hypothesis (ETH)). There is no $2^{o(n)}$ time algorithm for 3-SAT, where n is the number of variables.

An equivalent way to state ETH is, *there exists $\delta > 0$ such that 3-SAT has no $O(2^{\delta n})$ time algorithm.* Obviously, ETH is a stronger hypothesis than P $\neq$ NP (i.e., ETH implies P $\neq$ NP). ETH basically says that

3-SAT has no sub-exponential time algorithm, while $P \neq NP$ only says that 3-SAT has no polynomial time algorithm. As a remark, the current best algorithm for 3-SAT runs in 1.3071^n time.

Let us first exam what ETH will imply for the complexity of other NP-complete problems. Ideally, we would like that ETH implies that many other NP-complete problems do not admit sub-exponential time algorithms as well. If this is the case, when we are doing fine-grained complexity, we can reduce a problem from many other NP-complete problems, not just 3-SAT.

Let us first look at the independent set problem. Recall that, for a 3-cnf formula with n variables and m clauses, we have constructed an independent set instance $G=(V,E)$ with $|V|=3m$ vertices (remember that we have constructed a triangle for each clause). Since we only have $|V|=3m \leq 3\cdot\binom{n}{3}=O(n^3)$, ETH only implies the independent set problem do not admit a $2^{o(\sqrt[3]{n})}$ time algorithm, where $n=|V|$ is the number of vertices. Ideally, we would like to show that the independent set problem do not admit any sub-exponential time algorithm. That is, we would like to show that the independent set problem cannot be solved in $2^{o(n)}$ time. To show this, the following *sparsification lemma* due to Impagliazzo et al. [2001] is powerful.

Theorem 2 (Sparsification Lemma). *If there is a $2^{\delta n}$ time algorithm for 3-SAT for any $\delta > 0$, then, for any $\delta' > 0$, there is a $2^{\delta' n}$ time algorithm for the 3-SAT problem with the restriction $m = O(n)$ (where n is the number of variables and m is the number of clauses).*

Now we can show that the independent set problem do not have a sub-exponential time algorithm.

Theorem 3. *Assuming ETH, the independent set problem cannot be solved in $2^{o(n)}$ time, where n is the number of vertices.*

Proof. Consider the 3-SAT problem with the restriction $m=O(n)$. By ETH and the sparsification lemma, there is no $2^{o(n)}$ time algorithm that can solve this restricted version of the 3-SAT problem. Consider the reduction we have seen in Lecture 13 where, for a 3-SAT instance ϕ with n variables and m clauses such that $m=O(n)$, we have constructed an independent set instance $G=(V,E)$ with $|V|=3m=O(n)$. If we have an algorithm for the independent set problem that runs in $2^{o(n)}$ time, we can use this reduction to obtain an algorithm for the (restricted) 3-SAT problem that runs in $2^{o(n)}$ time, which leads to a contradiction. $\square$

1.2 Strong Exponential-Time Hypothesis

Hypothesis 4 (Strong Exponential-Time Hypothesis (SETH)). SAT cannot be solved in $O((2-\varepsilon)^n)$ time for any $\varepsilon > 0$.

SETH is even stronger than ETH. In fact, many researchers do not believe it is true. SETH implies many impossibility results in fine-grained complexity. If any of these impossibility results fails, then SETH is refuted. However, we have not seen the failure of one such impossibility result yet. In the next two sections, we will see two fine-grained complexity results based on SETH.

2 Fine-Grained Complexity for Orthogonal Vectors

Problem 5 (Orthogonal Vectors). Let $A, B \subseteq \{0,1\}^d$ be two sets of d-dimensional binary vectors with $|A| = |B| = n$ Decide if there exists $\mathbf{a} \in A$ and $\mathbf{b} \in B$ such that $\mathbf{a}^\top \mathbf{b} = 0$.

Clearly, the orthogonal vectors problem can be solved in $O(n^2 d)$ time by enumerating all pairs of vectors $(\mathbf{a}, \mathbf{b}) \in A \times B$. The following hypothesis says that this is essentially the best we can do if we are considering the exponent 2 of n in the time complexity. This hypothesis holds if SETH holds.

Hypothesis 6 (Orthogonal Vectors Hypothesis (OVH)). The orthogonal vectors problem cannot be solved in $O(n^{2-\varepsilon} d^c)$ time for any $\varepsilon > 0$ and any c.

Theorem 7. *SETH implies OVH.*

The general framework for showing a result like this is still a reduction. Consider a reduction $\mathcal{R}$ that maps an instance a of problem A to an instance b of problem B, and let $r(a)$ be the running time for computing $b = \mathcal{R}(a)$. Suppose we have an algorithm that solves an instance b of problem B in $T_B(b)$ time. Then, we can solve the instance a of problem A in $r(a) + T_B(b)$ time.

Proof of Theorem 7. Given a SAT instance ϕ with n variables and m clauses, we construct a orthogonal vectors instance as follows. We assume n is an even number without loss of generality. Let $V_A = \{x_1, \dots, x_{\frac{n}{2}}\}$ be the set of the first $\frac{n}{2}$ variables in the SAT instance, and $V_B = \{x_{\frac{n}{2}+1}, \dots, x_n\}$ be the set of the remaining $\frac{n}{2}$ variables. Given a Boolean assignment $\sigma \in \{0,1\}^{V_A}$ to the variables in V_A, let $\mathbf{x}_\sigma$ be the m-dimensional binary vector whose i-th entry is defined as follows:

$$x_\sigma[i] = \begin{cases} 0 & \text{if } \sigma \text{ satisfies the } i\text{-th clause} \\ 1 & \text{if } \sigma \text{ fails to satisfy the } i\text{-th clause} \end{cases}.$$

Notice that σ is only a partial assignment to the first $\frac{n}{2}$ variables. By saying σ satisfies the i-th clause, we mean that at least one literal in the clause evaluates to true based on the partial assignment σ. By saying σ fails to satisfy the i-th clause, we mean that, for each literal in the clause, either the literal evaluates to false or the literal's value is not defined by σ. Given a Boolean assignment $\tau \in \{0,1\}^{V_B}$ to the variables in V_B, let $\mathbf{y}_\tau$ be defined similarly. Finally, the orthogonal vectors instance is given by $A = \{\mathbf{x}_\sigma \mid \sigma \in \{0,1\}^{V_A}\}$ and $B = \{\mathbf{y}_\tau \mid \tau \in \{0,1\}^{V_B}\}$.

For the size of the orthogonal vectors instance, we have $N := |A| = |B| = 2^{\frac{n}{2}}$, and $d = m$. The reduction takes $O(2^{\frac{n}{2}} m)$ time.

It is easy to see the validity of the reduction: the SAT instance ϕ is a yes instance if and only if the orthogonal vectors instance is a yes instance. In particular, it is straightforward to check that $\mathbf{x}_\sigma^\top \mathbf{y}_\tau = 0$ if and only if the Boolean assignment $(\mathbf{x}_\sigma, \mathbf{y}_\tau)$ is a satisfying assignment for ϕ.

Suppose OVH is false. We have an algorithm that solves the above orthogonal vectors instance in $O(N^{2-\varepsilon}d^c)$ for some $\varepsilon > 0$ and some c. Substituting $N = 2^{\frac{n}{2}}$ and $m = d$ into it, the SAT instance can be solved in time

$$O\left(2^{\frac{n}{2}}m\right) + O\left(\left(2^{\frac{n}{2}}\right)^{2-\varepsilon}m^c\right) = O\left(2^{(1-\frac{\varepsilon}{2})n}m^c\right) = O\left((2-\varepsilon')^n\right),$$

for some $\varepsilon' > 0$. This would invalidate SETH. The contra-positive of this implies the theorem. □

As an important remark, to make the reduction work, we need to make sure the running time for computing the reduction, $r(a)$, should not be asymptotically larger than $T_B(b)$. Our objective is to show that being able to solve problem B in $T_B(b)$ time implies being able to solve problem A in some $T_A(a)$ time. The reduction implies $T_A(a) = r(a) + T_B(b)$. If $r(a) = \omega(T_B(b))$, we can conclude nothing.

3 Fine-Grained Complexity for Longest Common Subsequence

Problem 8 (Longest Common Subsequence). Given two strings x and y, decide the maximum length of their common sub-string.

For example, if we have two strings "DIFFICULTIES" and "INDUSTRIES", the longest common subsequence is "IUTIES" or "DUTIES", which have length 6.
A simple dynamic programming algorithm can solve this problem. Let $F(i,j)$ be the length of the longest common subsequence for the string $x[0,1,\ldots,i]$ and the string $y[0,1,\ldots,j]$. We have the following recurrence relation:

$$F(i,j) = \max\{F(i-1,j-1) + \mathbf{1}(x[i] = y[j]), F(i,j-1), F(i-1,j)\}.$$

It is easy to check that the algorithm runs in $O(n^2)$ time (where n is the sum of the lengths of the two strings). In this section, we will show that this is the best we can do, assuming SETH. Since we have seen in the previous section that SETH implies OVH, the following theorem implies longest common subsequence cannot be solved in $O(n^{2-\varepsilon})$ time, assuming SETH.

Theorem 9. *Assuming OVH, longest common subsequence cannot be solved in $O(n^{2-\varepsilon})$ time for any $\varepsilon > 0$.*

Given an orthogonal vectors instance (A,B) with $|A| = |B| = n$ such that the dimension of the vectors is d, we construct a longest common subsequence instance (x,y,k) with $|x|,|y| = O(nd^2)$. Here, we have formulated the longest common subsequence problem as a decision problem: decide if x and y have a common subsequence with length at least k. The reduction must map yes instances to yes instances and no instances to no instances. That is, if there exists $\mathbf{a} \in A$ and $\mathbf{b} \in B$ such that $\mathbf{a}^\top\mathbf{b} = 0$, then x and y must have a common subsequence with length at least k. Otherwise, the longest common subsequence of x and y must be strictly less than k. We will use $L(x,y)$ to denote the length of the longest common subsequence of x and y.

Let us consider the simplest case with $n = 1$ and $d = 1$. In this case, there is only one vector in each of A and B, and the vector in A or B is just a binary number which is either 0 or 1. For the binary number in A, we need to construct a string representing it. Let $C_A(0)$ and $C_A(1)$ be the two strings representing 0 and 1 in A. Let $C_B(0)$ and $C_B(1)$ be the two strings representing 0 and 1 in B. By setting

$$C_A(0) = 001, \quad C_A(1) = 111, \quad C_B(0) = 011, \quad \text{and} \quad C_B(1) = 000, \tag{1}$$

we can check that the longest common subsequence for $C_A(a)$ and $C_B(b)$ is 2 if $C_A(a) \cdot C_B(b) = 0$ (i.e., the two vectors are orthogonal), and it is 0 if $C_A(a) \cdot C_B(b) = 1$ (i.e., the two vectors are not orthogonal). Therefore, for the instance $(A = \{a\}, B = \{b\})$, the longest common subsequence constructed is $(x = C_A(a), y = C_B(b), k = 2)$.
In the next step, we consider the more complicated case with $n = 1$ and general d. That is, we still assume each of A and B contains only one vector, but the dimension of the vector is not restricted. We need to check if $\mathbf{a} \in A$ and $\mathbf{b} \in B$ is orthogonal. Our reduction needs to map $\mathbf{a}$ and $\mathbf{b}$ to two strings $V_A(\mathbf{a})$ and $V_B(\mathbf{b})$ such that the longest common subsequence $L(V_A(\mathbf{a}), V_B(\mathbf{b}))$ is at least some number k if $\mathbf{a}^\top \mathbf{b} = 0$, and the longest common subsequence $L(V_A(\mathbf{a}), V_B(\mathbf{b}))$ is strictly less than k if otherwise.

Vector gadget. Below, we introduce a new character '2'. 2^{3d} denotes the string containing $3d$ consecutive '2's. C_A, C_B are those defined in (1). a_i and b_i denote the i-th bit of $\mathbf{a}$ and $\mathbf{b}$ respectively. The vector gadget is defined as follows:

$$\begin{aligned} V_A(\mathbf{a}) &= C_A(a_1)2^{3d}C_A(a_2)2^{3d}\cdots 2^{3d}C_A(a_d) \\ V_B(\mathbf{b}) &= C_B(b_1)2^{3d}C_B(b_2)2^{3d}\cdots 2^{3d}C_B(b_d) \end{aligned} \tag{2}$$

Lemma 10. *$V_A, V_B : \{0,1\}^d \to \{0,1,2\}^{3d^2}$ can be computed in $O(d^2)$ time. Moreover, the longest common subsequence $L(V_A(\mathbf{a}), V_B(\mathbf{b}))$ is α_d if $\mathbf{a}^\top \mathbf{b} = 0$, and it is at most $\alpha_d - 2$ if otherwise, where $\alpha_d = 3d^2 - d$.*

Proof. The first part of the lemma about the computation time is trivial.
We claim that

$$L(V_A(\mathbf{a}), V_B(\mathbf{b})) = 3d(d-1) + \sum_{k=1}^{d} L(C_A(a_k), C_B(b_k)).$$

To see this, we need to make sure those $d - 1$ segments of 2^{3d} from the two strings must be matched exactly. Otherwise, we have lost at least $3d$ in the length of the common subsequence, and this cannot be compensated from anywhere else. Given that those $d - 1$ segments being matched exactly, each $C_A(a_k)$ is then matched with $C_B(b_k)$. If $\mathbf{a}^\top \mathbf{b} = 0$, we know that $L(C_A(a_k), C_B(b_k)) = 2$ for each $k = 1, \ldots, d$, which implies $L(V_A(\mathbf{a}), V_B(\mathbf{b})) = 3d(d-1) + 2 \cdot d = 3d^2 - d$. Otherwise, we know that there exists k such that $L(C_A(a_k), C_B(b_k)) = 0$, and we have $L(V_A(\mathbf{a}), V_B(\mathbf{b})) \le 3d^2 - d - 2$. □

Normalized vector gadget. In Lemma 10, in the case where $\mathbf{a}^\top\mathbf{b} \neq 0$, we have that $L(V_A(\mathbf{a}), V_B(\mathbf{b}))$ is *at most* $\alpha_d - 2 = 3d^2 - d - 2$. We aim to construct *normalized vector gadgets* N_A and N_B to make it *exact*. Below, we have introduced a new character '3', and α_d is defined in Lemma 10.

$$\begin{aligned} N_A(\mathbf{a}) &= V_A(\mathbf{a})3^{\alpha_d-2} \\ N_B(\mathbf{b}) &= 3^{\alpha_d-2}V_B(\mathbf{b}) \end{aligned} \tag{3}$$

Lemma 11. *$N_A, N_B : \{0,1\}^d \to \{0,1,2,3\}^{6d^2-d-2}$ can be computed in $O(d^2)$ time. Moreover, $L(N_A(\mathbf{a}), N_B(\mathbf{b})) = \alpha_d$ if $\mathbf{a}^\top\mathbf{b} = 0$, and $L(N_A(\mathbf{a}), N_B(\mathbf{b})) = \alpha_d - 2$ if $\mathbf{a}^\top\mathbf{b} \neq 0$.*

Proof. The first part of the lemma about the computation time is trivial. If $\mathbf{a}^\top\mathbf{b} = 0$, we have $L(V_A(\mathbf{a}), V_B(\mathbf{b})) = \alpha_d$ by Lemma 10, and matching $V_A(\mathbf{a})$ with $V_B(\mathbf{b})$ gives us the longest common subsequence, which has length α_d. If $\mathbf{a}^\top\mathbf{b} \neq 0$, we have $L(V_A(\mathbf{a}), V_B(\mathbf{b})) \le \alpha_d - 2$ by Lemma 10, and matching the segment 3^{α_d-2} in both strings gives us the longest common subsequence, which has length $\alpha_d - 2$. $\square$

So far, we have only considered the case where A and B contain only one vector. In the final step, we consider the most general case where $|A| = |B| = n$. We need to construct a single string x that aggregates all the vectors in A and a single string y that aggregates all the vectors in B such that $L(x, y) = k$ if there exist $\mathbf{a} \in A$ and $\mathbf{b} \in B$ with $\mathbf{a}^\top\mathbf{b} = 0$ and $L(x, y) < k$ if otherwise. The challenge here is, we need to make sure a particular segment in x that represents a vector in A is matched with a particular segment in y that represents a vector in B. Let $A = \{\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_{n-1}\}$ and $B = \{\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_{n-1}\}$. The longest common subsequence instance (x, y, k) is constructed as follows:

$$\begin{aligned} x &= N_A(\mathbf{a}_0)4^\gamma N_A(\mathbf{a}_1)4^\gamma \cdots 4^\gamma N_A(\mathbf{a}_{n-1})4^\gamma N_A(\mathbf{a}_0)4^\gamma N_A(\mathbf{a}_1)4^\gamma \cdots 4^\gamma N_A(\mathbf{a}_{n-1}) \\ y &= 4^{n\gamma} N_B(\mathbf{b}_0)4^\gamma N_B(\mathbf{b}_1)4^\gamma \cdots 4^\gamma N_B(\mathbf{b}_{n-1})4^{n\gamma} \end{aligned} \tag{4}$$

and $k = (2n-1)\gamma + n(\alpha_d - 2) + 2$, where we have used a new character '4' and we set $\gamma = 6d^2 - d - 2$.
After some observations, it is not hard to see that, in the longest common subsequence, all the $2n-1$ segments of 4^γ in x must be matched, and the segments $N_B(\mathbf{b}_0), N_B(\mathbf{b}_1), \ldots, N_B(\mathbf{b}_{n-1})$ in y must be matched to the segments $N_A(\mathbf{a}_\Delta), N_A(\mathbf{a}_{\Delta+1}), \ldots, N_A(\mathbf{a}_{n-1}), N_A(\mathbf{a}_0), N_A(\mathbf{a}_1), \ldots, N_A(\mathbf{a}_{\Delta-1})$ in x accordingly (for some $\Delta \in \{0, 1, \ldots, n-1\}$). In the case there do not exist $\mathbf{a}_i \in A$ and $\mathbf{b}_j \in B$ with $\mathbf{a}_i^\top\mathbf{b}_j = 0$, the length of the longest common subsequence is exactly $k - 2$. If there exist $\mathbf{a}_i \in A$ and $\mathbf{b}_j \in B$ with $\mathbf{a}_i^\top\mathbf{b}_j = 0$, by letting $\Delta = (i - j)$ mod n, the length of the longest common subsequence is at least k. This concludes the validity of the reduction: yes instances of the orthogonal vectors problem are always mapped to yes instances of the longest common subsequence problem, and no instances of the orthogonal vectors problem are always mapped to no instances of the longest common subsequence problem.
The reduction takes time $O(nd^2)$. Suppose we have an algorithm that solves the longest common subsequence problem in $O((|x| + |y|)^{2-\varepsilon})$ time. We can solve the orthogonal vector problem in time

$$O(nd^2) + O\left(\left(3n \cdot (6d^2 - d - 2) + \gamma \cdot (5n - 2)\right)^{2-\varepsilon}\right) = O(n^{2-\varepsilon}d^{4-2\varepsilon}),$$

which violates OVH.

参考文献

Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. 2