INFR11102: Computational Complexity 10/10/2019

Lecture 8: More on NP-completeness

Lecturer: Heng Guo

1 Cook-Levin Theorem (proof is non-examinable)

$L_{\mathtt{NP}}$ is not a very useful NP-complete problem. The surprising discovery in the 70s, by Stephen Cook [Coo71] and Leonid Levin [Lev73], independently, is that the following natural problem is NP-complete.

Name: Sat

Input: A CNF formula φ

Output: Is φ satisfiable?

Recall that a CNF (Conjunction Normal Form) formula is a conjunction of a number of disjunction clauses, like, $(x_1 \vee x_2) \wedge (\overline{x_1} \vee x_3 \vee x_4) \wedge \cdots$. To satisfy a CNF formula, we need to find an assignment so that all clauses are satisfied.

Given an assignment $\sigma : X \to \{0, 1\}$, where X is the variable set, it is straightforward to check whether σ satisfies φ. This means Sat $\in$ NP. (Recall the verification characterization of NP.)

Theorem 1 (Cook-Levin)**.** Sat *is* NP*-complete.*

Proof sketch. The basic goal of the proof is that, given a polynomial time NTM N and an input s, the computation of N on s can be encoded into a Boolean formula φ_s so that N accepts s if and only if φ_s is satisfiable. Additionally, the length of the formula is polynomial if the machine runs in polynomial time.

We may assume that N is single-tape, since it can simulate k-tapes NTMs with at most quadratic slowdown. We may also assume that the tape is one-sided, since we can always "fold" the tape by enlarging the alphabet size. Moreover, we assume that N always has 2 choices at every step. This is okay since we can always add $t - 2$ new states to mimic a t choices non-deterministic step. If there is only one choice, then we consider the two coincide. Now the non-deterministic choices are simply a $0, 1$-string: $\mathbf{c} = c_1, c_2, \cdots, c_T$ where T is the running time.

We form a T-by-$O(T)$ "computational table" as follows. Rows are time indices, and each row is the encoding of the configuration at the corresponding time. So the ith row encodes the configuration at time i. If we fix the choices $\mathbf{c}$, then the computation of N on x is completely deterministic and this table can be constructed. Equivalently, we may add an additional column of the table to reflect the choices $\mathbf{c}$.

We introduce one variable x for each cell of this conceptual table. Thus, we have $O(T^2)$ variables. We introduce subformulas to verify the following three things,

1. Every row is a valid encoding;
2. The initial row is correct;
3. The final row is accepting;
4. Every two consecutive rows are a valid transition.

Here by "verify" we mean that the subformula ψ is true if and only if the property to be verified is true.

It is tedious to go through all the constructions. The crucial part of the construction of φ is how to encode the transition function, namely to verify that two consecutive rows are valid. This is possible because computation is *local*. Basically, to determine whether two such rows are "compatible", we only need to look at $12 + 2\log|Q| + 1$ cells, 12 for the cell contents and positions of the heads, $2\log|Q|$ to check the consecutive states, and 1 extra to check c_i. We know that any Boolean function can be encoded as a (possibly exponential size) CNF. The saving grace is that exponential of a constant is still a constant. We do this for every 3 consecutive cells of the tapes, resulting in $O(T)$ many clauses.

As of the total size of φ, notice that T is a polynomial in n, and thus $O(T^2)$ is still a polynomial. The number of clauses, as explained above, is also bounded by a polynomial (in fact also $O(T^2)$). □

Full proof details can be found in [AB09, Theorem 2.10] or [Pap94, Theorem 8.2], as well as in many other books.

2 3-Sat

After Cook's paper [Coo71] published, Dick Karp immediately realized that the notion of NP-hardness captures a large amount of intractable combinatorial optimization problems. In [Kar72], he showed 21 problems to be NP-complete. This list quickly increased and by the time of 1979, Garey and Johnson [GJ79] wrote a whole book on NP-complete problems. This book has became a classic nowadays, and thousands of NP-hard problems were discovered during the past four decades. These intractable problems spread over all kinds of areas, even beyond computer science.

The canonical hard problem $L_{\mathtt{NP}}$ defined last time is not very useful to show NP-hardness of other problems, and Sat is much more handier in this sense. What is even more useful is the following variant of Sat. Let k-CNF formulas be those whose clauses involve at most k literals. For example, $(\overline{x_1} \vee \overline{x_2} \vee x_3 \vee x_4) \wedge (x_2 \vee x_5)$ is a 4-CNF.

Name: k-Sat

Input: A k-CNF formula φ.

Output: Is φ satisfiable?

Theorem 2. 3-Sat *is* NP-*complete.*

Proof. We give a reduction SAT $\leq_p$ 3-SAT. Namely, given a CNF formula φ, we construct (in polynomial time) another 3-CNF formula φ', such that φ is satisfiable, if and only if φ' is satisfiable.

The only thing we need to do is for every clause c in φ, we replace it by a conjunction of clauses of size at most 3 and preserve satisfying assignments. We do this inductively. For a clause c of size $k > 3$, say its first two literals are x_1 and x_2. So c has the form $x_1 \vee x_2 \vee c'$, where c' is a clause of size $k-2$. We introduce a new variable y_1 and consider the following formula: $\varphi_c := (x_1 \vee x_2 \vee y_1) \wedge (\overline{y_1} \vee c')$.

- If an assignment σ satisfies c, then at least one of x_1, x_2, and literals in c' is true under σ. Hence, we can assign y_1 accordingly to make φ_c true. For example, if x_1 is true, then we assign y_1 to be false.
- If an assignment σ satisfies φ_c, then depending on y_1's value, at least one of $x_1 \vee x_2$ and c' is true. Hence, c is satisfied under σ.

In this construction, the sizes of the new clauses (namely, $x_1 \vee x_2 \vee y_1$ and $\overline{y_1} \vee c'$) decrease at least 1. To continue, we apply this construction to $\overline{y_1} \vee c'$ and reduce clause sizes by 1 again (if still > 3). (Or equivalently, invoke the induction hypothesis.)

We finish with a 3-CNF formula φ' which contains a number of new variables, and φ is satisfiable if and only if φ' is. In fact, if there are at most k variables in each clause in φ, and there are n variables and m clauses in φ, then there are $\leq (k-2)m$ clauses and $\leq n+(k-3)m$ variables in φ'. It is also easy to see that the construction only takes polynomial time. □

There are two key points of the reduction above: 1. local transformations (from a clause to a conjunction of clauses, without affecting other clauses); 2. introducing new variables.

Since trivially 3-SAT $\leq_p$ k-SAT for any $k \geq 3$, k-SAT is NP-hard for any $k \geq 3$. On the other hand, the proof above does not work for 2-SAT. In fact, 2-SAT $\in$ P.

Remark (Bibliographic)*.* These reductions were first shown by Karp [Kar72]. Relevant chapters are [AB09, Chapter 2] and [Pap94, Chapter 8 and 9].

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach.* Cambridge University Press, 2009.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC*, pages 151–158. ACM, 1971.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

[Lev73] Leonid A. Levin. Universal sequential search problems. *Probl. Peredachi Inf. (in russian)*, 9(3):115–116, 1973.

[Pap94] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.