CS1200: Intro. to Algorithms and their Limitations	Anshu & Vadhan
Lecture 20: NP-completeness	
Harvard SEAS - Fall 2024	*2024-11-12*

1 Announcements

- Anurag OH today moved to Zoom, 11:30-12:30.
- Salil OH Thu 11:15-12:00 in SEC 3.327.
- PS7 due tomorrow.
- PS8 out by Thursday, due 2024-11-20.
- Next SRE on Thursday.

Recommended Reading:

- MacCormick §14, 17
- Sipser §7.5

2 Recap

- Def of $\mathsf{NP}_{\mathsf{search}}$.
- Def of $\mathsf{NP}_{\mathsf{search}}$-completeness.
- Cook–Levin Theorem and significance.
- Transitivity of $\leq_p$.

In today's lecture, we will prove that several different problems are $\mathsf{NP}_{\mathsf{search}}$-complete.

3 3-SAT

Once we have one $\mathsf{NP}_{\mathsf{search}}$-complete problem, we can get others via reductions from it. Consider the computational problem 3-SAT, which is obtained when we restrict the number of literals in each clause of SAT.

Input	**:** A CNF formula φ on n variables $z_0, \ldots z_{n-1}$ in which each clause has width at most 3 (i.e. contains at most 3 literals)
Output	**:** An $\alpha \in \{0,1\}^n$ such that $\varphi(\alpha) = 1$ (if one exists)

Computational Problem 3-SAT

Theorem 3.1. *3-SAT is* $\mathsf{NP}_{\mathsf{search}}$*-complete.*

Proof. The proof follows in two steps.

1. 3SAT is in $\mathsf{NP}_{\mathsf{search}}$:

2. 3SAT is $\mathsf{NP}_{\mathsf{search}}$-hard: Since every problem in $\mathsf{NP}_{\mathsf{search}}$ reduces to SAT (Cook–Levin Theorem), all we need to show is SAT $\leq_p$ 3SAT (since reductions compose).

The reduction algorithm from SAT to 3SAT has the following components (as illustrated last time). First, we give an algorithm R which takes a SAT instance φ to a 3SAT instance φ'.

$$\text{SAT instance } \varphi \xrightarrow{\text{polytime R}} \text{3SAT instance } \varphi'$$

Then we feed the instance φ' to our 3SAT oracle and obtain a satisfying assignment α' to φ' or $\perp$ if none exists. If we get $\perp$ from the oracle, we return $\perp$, else we transform α' into a satisfying assignment to φ using another algorithm S.

$$\text{SAT assignment } \alpha \xleftarrow{\text{polytime S}} \text{3SAT assignment } \alpha'$$

Algorithm R: The intuition behind this algorithm is that when we have a clause $(\ell_0 \vee \ell_1 \vee ... \vee \ell_{k-1})$ in the SAT instance ϕ (with large width $k > 3$), we want to break it into multiple clauses of width 3.

$R(\varphi)$:
Input : A CNF formula φ
Output : A CNF formula φ' with each clause of width 3
$\varphi' = \varphi$
$i = 0$
while φ' *has a clause* $C = (\ell_0 \vee \ldots \vee \ell_{k-1})$ *of width* $k > 3$ **do**
    Remove C
    Add clauses
return φ'

Note that φ' is **not** an equivalent formula to φ. While φ is on variables $z_0, \ldots z_{n-1}$, the formula φ' is on variables $z_0, \ldots z_{n-1}, y_0, \ldots y_{t-1}$, where t is the number of iterations of the while loop.

Algorithm S: Given an assignment $\alpha' = (\alpha_0, \ldots, \alpha_{n-1}, \beta_0, \ldots, \beta_{t-1})$ to φ', the algorithm simply takes the first n bits, i.e. $\alpha = (\alpha_0, \ldots, \alpha_{n-1})$

Next we consider the runtime and correctness of the overall reduction algorithm.

Runtime of the reduction algorithm: We first consider the runtime of the algorithm R:

Then, we consider the runtime of the algorithm S, which is simply $O(n)$. Overall, the runtime of the reduction algorithm is $O(nm)$.

Proof of correctness: We will show that if φ is satisfiable, then the reduction algorithm produces a satisfying assignment and if φ is unsatisfiable, the reduction algorithm will output $\perp$. This is based on the following two claims. (In lecture, we combined the claims into one to streamline the proof.)

Claim 3.2. *If φ is satisfiable then $\varphi' = R(\varphi)$ is satisfiable.*

Proof of claim. Assume that φ is satisfiable. Let $\varphi = \varphi_0, \varphi_1, \ldots, \varphi_t = R(\varphi)$ be the formula as it evolves through the t loop iterations. We will prove by induction on i that φ_i is satisfiable for $i = 0, \ldots, t$. constructed through the t loop iterations.

Base case ($i = 0$):

Induction step: By the induction hypothesis, we can assume that φ_{i-1} is satisfiable, and now we need to show that φ_i is satisfiable:

$\square$

Claim 3.3. *If α' satisfies $R(\varphi)$, then $\alpha = S(\alpha')$ also satisfies φ.*

Proof of claim. We prove by "backwards induction" that α' satisfies φ_i for $i = t, \ldots, 0$. We can then drop the extra t variables that don't appear in φ without changing the satisfiability. (We call this "backwards induction" since our base cases is $i = t$.)

The base case ($i = t$) follows because α' satisfies $R(\varphi) = \varphi_t$ by assumption.

For the induction step:

$\square$

To finish the correctness proof, suppose φ is satisfiable. Then from Claim 3.2, φ' is also satisfiable. The 3SAT oracle returns a satisfying assignment α', which is turned into a satisfying assignment for φ via the algorithm S (Claim 3.3). If φ is unsatisfiable, then by Claim 3.3, φ' is also unsatisfiable. In this case, the 3SAT oracle returns $\perp$ - as a result the reduction algorithm also returns $\perp$.

This completes the proof that 3-SAT is $\mathsf{NP}_{\mathsf{search}}$-complete. $\square$

4 Mapping Reductions

The usual strategy for proving that a problem Γ in $\mathsf{NP}_{\mathsf{search}}$ is also $\mathsf{NP}_{\mathsf{search}}$-hard (and hence $\mathsf{NP}_{\mathsf{search}}$-complete) follows a structure similar to the proof of Theorem 3.1.

1. Pick a known $\mathsf{NP}_{\mathsf{search}}$-complete problem Π to try to reduce to Γ.
2. Come up with an algorithm R mapping instances x of Π to instances $R(x)$ of Γ.
3. Show that R runs in polynomial time.
4. Show that if x has a valid answer, then so does $R(x)$.
5. Conversely, show that if $R(x)$ has an answer, then so does x. Moreover, we can transform valid answers to $R(x)$ into valid answers to x through a *polynomial time* algorithm S.

Reductions with the structure outlined above are called *mapping reductions*, and they are what are typically used throughout the theory of NP-completeness. A formal definition will be given in the detailed notes.

5 Independent Set

Next we turn to IndependentSet. (Formally the IndependentSet-ThresholdSearch version.)

Theorem 5.1. *IndependentSet is* $\mathsf{NP}_{\mathsf{search}}$*-complete.*

Proof. We'll do this proof less formally than we did the proof of $\mathsf{NP}_{\mathsf{search}}$-completeness of 3SAT.

1. In $\mathsf{NP}_{\mathsf{search}}$:

2. $\mathsf{NP}_{\mathsf{search}}$-hard: We will show 3SAT $\leq_p$ IndependentSet.

We've previously encoded many other problems in SAT, but here we're going in the other direction and showing a graph problem can encode SAT.

Our reduction $R(\varphi)$ takes in a CNF and produces a graph G and a size k. We'll use as an example the formula

$$\varphi(z_0, z_1, z_2, z_3) = (\neg z_0 \vee \neg z_1 \vee z_2) \wedge (z_0 \vee \neg z_2 \vee z_3) \wedge (z_1 \vee z_2 \vee \neg z_3).$$

Our graph G consists of:

- Variable gadgets:

- Clause gadgets:

- Conflict edges:

We pick $k = m + n$, which implies that every independent set of size k must contain exactly one vertex from each variable gadget and one vertex from each clause gadget. An algorithm R can create this graph (and k) in polynomial time given φ. The graph for the formula φ is below.

Note that (analogously to the SAT to 3SAT case) the correspondence between 3SAT and ISET does not exactly preserve the set of satisfying solutions (they aren't even the same problem) but we will see how we can go from solutions to one to solutions to the other.

Remember that an IndependentSet problem consists of 1) a graph G and 2) a minimum size k of an independent set. How can we choose the size k for this reduction? Intuitively, we might think about assigning True to the variables whose corresponding vertices are selected as part of the independent set. Then, we'll choose $k = n + m$, where n is the number of variables and m is the number of clauses in the original Boolean formula. The hope is that the clause edges will force exactly n of the variable gadgets to be set to True, and at least one vertex in each of the m clause gadget is also True. We'll now prove that this claim is true.

Claim 5.2. *G has an independent set of size $k = n + m$ if and only if φ is satisfiable. Moreover, we can map independent sets of size k to satisfying assignments of φ in polynomial time.*

Proof of claim.

□

This completes the proof that IndependentSet is $\mathsf{NP}_{\mathsf{search}}$-complete. □

6 Longest Path

Finally, we consider the problem from SRE5:

Input	**:** A digraph $G = (V, E)$, two vertices $s, t \in V$, and a path-length $k \in \mathbb{N}$
Output	**:** A path from s to t in G of length k, if one exists

Computational Problem LongPath

In the Sipser text (optional reading), it is proven that LongPath is $\mathsf{NP}_{\mathsf{search}}$-complete, even in the special case where $k = n - 1$, i.e. the path visits all vertices in the graph:

Input	**:** A digraph $G = (V, E)$, two vertices $s, t \in V$
Output	**:** A path from s to t in G that includes every vertex in G, if one exists

Computational Problem HamiltonianPath

Theorem 6.1. *HamiltonianPath is* $\mathsf{NP}_{\mathsf{search}}$*-complete.*

In fact, the HamiltonianCycle problem (where $s = t$) is also $\mathsf{NP}_{\mathsf{search}}$-complete, even in undirected graphs. (The Sipser text has a reduction from HamiltonianPath to UndirectedHamiltonianPath; the reduction from HamiltonianPath to HamiltonianCycle is a good exercise.) HamiltonianCycle is a special case of the Travelling Salesperson Problem (TSP), where a salesperson wants to visit a set of cities and return to their start city in the minimum amount of travel time. If the possible trips between cities are given by a digraph G and every possible trip takes the same amount time, then the shortest possible route is of length n and is given by a HamiltonianCycle.

In contrast, *EulerianWalk*, where we seek a walk from s to t that uses every *edge* in G exactly once is known to be in $\mathsf{P}_{\mathsf{search}}$.

In optional reading in the detailed lecture notes, there is also a proof that 3DCompleteMatching (the problem from ps7) is $\mathsf{NP}_{\mathsf{search}}$-complete.