CS120: Intro. to Algorithms and their Limitations Hesterberg & Vadhan

Lecture 19: NP and NP-completeness

Harvard SEAS - Fall 2022 *Nov. 8, 2022*

1 Announcements

Recommended Reading:

- MacCormick §12.0–12.3, Ch. 13, 14, 17

2 Recap

3 NP

Roughly speaking, NP consists of the computational problems where solutions can be *verified* in polynomial time. This is a very natural requirement; what's the point in searching for something if we can't recognize when we've found it?

Definition 3.1. A computational problem $\Pi = (\mathcal{I}, \mathcal{O}, f)$ is in $\mathsf{NP}_{\mathsf{search}}$ if the following conditions hold:

1. All solutions are of polynomial length:

2. All solutions are verifiable in polynomial time:

(Remark on terminology: $\mathsf{NP}_{\mathsf{search}}$ is often called FNP in the literature, and is closely related to, but slightly more restricted than, the class PolyCheck defined in the MacCormick text.)

Examples:

1. Satisfiability:

2. GraphColoring:

Non-Example:

1. IndependentSet-OptimizationSearch:

Even though this problem does not appear to be in $\mathsf{NP}_{\mathsf{search}}$ (why?), you saw on Problem Set 5 that it reduces in polynomial time to a problem in $\mathsf{NP}_{\mathsf{search}}$. (Which one?)

Every problem in $\mathsf{NP}_{\mathsf{search}}$ can be solved in exponential time:

Proposition 3.2. $\mathsf{NP}_{\mathsf{search}} \subseteq \mathsf{EXP}_{\mathsf{search}}$.

Proof.

$\square$

Every problem in $\mathsf{NP}_{\mathsf{search}}$ has a corresponding decision problem (deciding whether or not there is a solution). The class of such decision problems is called NP and we will study it more next time.

4 NP-Completeness

Unfortunately, although it is widely conjectured, we do not know how to prove that $\mathsf{NP}_{\mathsf{search}} \not\subseteq \mathsf{P}_{\mathsf{search}}$. As we will see next time, this is an equivalent formulation of the famous P vs. NP problem, considered one of the most important open problems in computer science and mathematics.

However, even without resolving the P vs. NP conjecture, we can give strong evidence that problems are not solvable in polynomial time by showing that they are NP-*complete*:

Definition 4.1 (NP-completeness, search version)**.** A problem Π is $\mathsf{NP}_{\mathsf{search}}$-*complete* if:

1.

2.

We can think of the NP-complete problems as the "hardest" problems in NP. Indeed:

Proposition 4.2. *Suppose* Π *is* $\mathsf{NP}_{\mathsf{search}}$*-complete. Then* $\Pi \in \mathsf{P}_{\mathsf{search}}$ *iff* $\mathsf{NP}_{\mathsf{search}} \subseteq \mathsf{P}_{\mathsf{search}}$.

Remarkably, there are natural NP-complete problems. The first one is Satisfiability:

Theorem 4.3 (Cook–Levin Theorem)**.** *SAT is* $\mathsf{NP}_{\mathsf{search}}$*-complete.*

This can be interpreted as strong evidence that SAT is not solvable in polynomial time. If it were, then *every* problem in $\mathsf{NP}_{\mathsf{search}}$ would be solvable in polynomial time. We won't cover (or expect you to know) the proof of the Cook–Levin Theorem, but we may provide you a proof sketch in the last set of lecture notes in the course.

5 More $\mathsf{NP}_{\mathsf{search}}$-complete Problems

Once we have one $\mathsf{NP}_{\mathsf{search}}$-complete problem, we can get others via reductions from it.

Theorem 5.1. *3-SAT is* $\mathsf{NP}_{\mathsf{search}}$*-complete.*

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

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

For part (2) we follow a general reduction template. First, we transform the problem from what we want to solve to what we have an oracle for.

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

Then we feed the instance φ' to our 3SAT oracle and obtain an assignment α' (or $\perp$ if none exists). Finally:

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

In this case, we need to give the reduction R. Intuitively, when we have long clause $(x_1 \vee x_2 \vee ... \vee x_k)$ for $k > 3$ we want to break it into multiple clauses of size 3. But simply breaking it up doesn't preserve information about φ being satisfiable. Our reduction R is as follows:

```
1 $R(\varphi)$ :
  Input  : A CNF formula $\varphi$
  Output : A 3-CNF formula $\varphi'$
2 $\varphi' = \varphi$
3 while $\varphi'$ has a clause $C = (\ell_0 \vee \ldots \vee \ell_k)$ of length $k > 3$ do
4     Remove $C$
5     Add clauses
6 return $\varphi'$
```

This is **not** an equivalent formula to the original (we introduced potentially many dummy variables), but it preserves what we care about — φ' is satisfiable iff φ is (as we'll prove below).

We need to check that R runs in polynomial time:

Claim 5.2. *If φ is satisfiable then $\varphi' = R(\varphi)$ is satisfiable.*

Proof of claim. Assume that φ is satisfiable. We prove that φ' is satisfiable throughout the algorithm by induction on the number of loop iterations.

Base case:

Induction step: By the induction hypothesis, we can assume that φ' is satisfiable after m loop iterations, and now we need to show that it will remain satisfiable after $m + 1$ iterations:

□

Finally, we need to show we can transform a satisyfing assignment α' to φ' into a satisfying assignment α to φ. Our S simply discards all introduced dummy y variables and takes the assignment to the x variables.

Claim 5.3. *If α' satisfies $\varphi' = R(\varphi)$, then α'_x also satisfies φ, where α'_x is the restriction of the assignment α' to the x variables.*

Proof of claim. We prove that α' satisfies φ' throughout the algorithm by *backward* induction on the loop. By assumption, we know that it satisfies φ' at the end of the last loop (this our base case) and we want to prove that it satisfies φ at the beginning of the first loop.

For the induction step:

□

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

Theorem 5.4. *IndependentSet is $\mathsf{NP}_{\mathsf{search}}$-complete.*

Proof. We'll do this proof less formally.

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

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

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.

$$\varphi(x_0, x_1, x_2, x_3) = (\neg x_0 \vee \neg x_1 \vee x_2) \wedge (x_0 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_3).$$

Our graph G consists of:

- Variable gadgets:
- Clause gadgets:
- Consistency edges:

An algorithm R can create this graph in polynomial time given φ. Here is an example for the formula φ above (using $!x$ to mean $\neg x$):

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 can go from solutions to one to solutions to the other:

Claim 5.5. *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.

$\square$

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