CS 120: Intro to Algorithms and their Limitations

Problem Set 8

Due: November 29, 2023 11:59pm **Denny Cao**

Collaborators:
No. of late days used on previous psets: 3
No. of late days used after including this pset: 6

§1 Positive Monotone SAT

(a) *Proof.* Let $\Gamma = k$-`False PositiveMonotoneSAT`.

We will show that Γ is $\texttt{NP}_{\texttt{search}}$-complete by showing that

1. $\Gamma \in \texttt{NP}_{\texttt{search}}$:
   - Solutions are of polynomial size, as an assignment α is of n variables that correspond to the value associated for each of the n inputs.
   - Our verifier can check if at least $k = n/2$ variables are set to 0 in polynomial time $O(n)$ by iterating over an assignment α and incrementing a variable `count` if the current literal is 0 and after iterating through α, checking if `count` $\leq k$. We can then check if α satisfies the input to Π, the positive monotone CNF formula φ (This uses the same verifier as for SAT, which from the Cook-Levin Theorem, is $\texttt{NP}_{\texttt{search}}$-complete) which will be polynomial time $O(m)$ where m is the amount of clauses. Thus, it runs in polynomial time, as the total runtime is $O(n+m)$.
2. $\Gamma \in \texttt{NP}_{\texttt{search}}$-hard: Since every problem in $\texttt{NP}_{\texttt{search}}$ reduces to `SAT`, all we need to show is `SAT` $\leq_p \Gamma$ since reductions are transitive. We provide a reduction R:

$$\texttt{SAT instance } \varphi \xrightarrow{\text{polytime } R} k\texttt{-False PositiveMonotoneSAT instance } \varphi'$$

We first describe the reduction R for a `CNF` formula $\varphi(x_0, \ldots, x_{n-1})$:

- Create new variables $x'_0, \ldots, x'_{n-1}$ to represent the negation of the literals $x_0, \ldots, x_{n-1}$ respectively.
- Form a new `CNF` formula φ' by replacing all negated literals $\neg x_i, i \in [n]$ with x'_i and adding a new clause $(x_i \vee x'_i)$.
- Make an oracle call to Γ with instance φ' and $k = n'/2 = n$, where $n' = 2n$ is the number of literals in φ', as we keep the n literals from φ and add another literal corresponding to each.
- If the oracle returns a satisfying assignment α' to φ', we transform α' into a satisfying assignment φ. If it returns $\perp$, we return $\perp$.

Runtime: We check that R runs in polynomial time. Creating n literals takes $O(n)$ time. We then iterate over each clause of φ and replace at most n literals in each clause, and thus takes time $O(nm)$. We then add n clauses for each x_i, x'_i pair which takes $O(n)$ time. We assign the constant k a value n which takes $O(1)$. We then make a single oracle call taking $O(1)$ time. Thus, in total, the run time is $O(n) + O(nm) + O(n) + O(1) + O(1) = O(n(1+m))$ which is polynomial time.

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

Proof of claim. As $R(\varphi)$ transforms the `CNF` formula by replacing negated literals $\neg x_i$ with x'_i, it follows that the assignment for φ would satisfy φ'.

$R(\varphi)$ then adds n clauses of the form $(x_i \vee x'_i)$. As $k = n$, there are at least n literals that are 0. There are $2n$ literals with each of the n clauses consisting of 2 literals, and thus each clause must have 1 literal that is 0 and the other that is 1, as if both are 0, then the clause would not be satisfied. If $x'_i = \alpha'_i$, then it follows that $x_i = \neg\alpha'_i$, and thus x'_i is in fact the negation of x_i. Thus, we have shown that the first m clauses of φ' are satisfied if φ, since x'_i is the negation of x_i, and the last n clauses are satisfied since one of x_i and x'_i is 1, and the other is 0. ■

> **Claim —** If α' satisfies $R(\varphi)$, then $\alpha' \mid_\varphi$ also satisfies φ, where $\alpha' \mid_\varphi$ is the restriction of the assignment α' to the variables in φ, and $\alpha' \mid_\varphi$ can be obtained in polynomial time.

Proof of claim. Assume $\alpha' = (\alpha_0, \dots \alpha_{n-1}) \cup (\alpha'_0, \dots, \alpha'_{n-1})$, where α_i is the assignment for x_i and $\alpha'_i = \neg\alpha_i$ is the assignment for x'_i. Let m be the number of clauses in φ. Then, φ' will have $m+n$ clauses, as $R(\varphi)$ adds n more clauses.

Consider all assignments for φ' $x_i = \alpha_i, x'_i = \neg\alpha_i, i \in [n]$. We guarantee that for all x_i, x'_i is the corresponding negation through the last n clauses of the form $(x_i \vee x'_i)$ and assigning $k = 2n/2 = n$. In total, there are $2n$ literals, and this assignment of k ensures that, for each clause $(x_i \vee x'_i)$, if $x_i = \alpha_i$, then $x'_i = \neg\alpha_i$, as we join the n clauses with $\wedge$ and thus by the Pigeonhole Principle, at least one of the variables must be true since $k = n$, meaning at least n literals must be 0, and thus 1 literal in each clause will be 0 and the other will be 1.

As we have shown that $x_i = \neg x'_i$, it follows that, since the first m clauses are the same as the clauses of φ but the negations $\neg x_i$ replaced with x'_i, if we set $\alpha' \mid_\varphi = (a_0, \dots, a_{n-1})$, φ will be satisfied. The removal of $\alpha'_0, \dots, \alpha'_{n-1}$ takes polynomial time, as desired. ■

This completes the proof that k-`False PositiveMonotoneSAT` is $\mathrm{NP}_{\mathtt{search}}$-complete. □

(b) *Proof.* We first prove the following claim:

> **Claim —** There exists an assignment α' satisfying the `CNF` formula consisting of exactly 3 variables as 0 if and only if there exists an assignment α satisfying the `CNF` formula consisting of at least 3 variables as 0.

Proof of claim. We will prove the claim by proving both directions:

($\implies$) An assignment of 3 variables as 0 is an assignment of at least 3 variables as 0.

($\impliedby$) Assume there is an assignment of x variables that are 0, $x \geq 3$. As it is a `CNF`, if a clause is satisfied where one variable is 0, the other must be 1. Thus, the clause will remain satisfied if we replace the variable assigned to 0 with an assignment to 1. We can do this for all variables equal to 0 until we reach only 3 variables assigned to 0. ■

Using the claim, we can solve k-`False PositiveMonotoneSAT` with $k = 3$ by exhaustive search. We iterate through all possible combinations of setting 3 variables to 0, leaving the rest as 1. There are $\binom{n}{3}$ possible combinations, and for each combination we take $O(m)$ to verify if the assignment satisfies the `CNF` formula. Thus, the total runtime is

$$\begin{aligned}\binom{n}{3}O(m) &= \frac{n!}{(n-3)!3!}O(m) \\ &= \frac{n(n-1)(n-2)}{6}O(m) \\ &= O(n^3)O(m) \\ &= O(n^3 m)\end{aligned}$$

As $O(n^3m)$ is polynomial time, we have shown that, if we fix $k = 3$, then k-`False PositiveMonotoneSAT` $\in \mathtt{P}_{\mathtt{search}}$, and the proof is complete. □

(c) *Proof.* Let $\Gamma = k$-`False PositiveMonotone 2-SAT`.

We will show that Γ is $\mathtt{NP}_{\mathtt{search}}$-complete by showing that:

1. $\Gamma \in \mathtt{NP}_{\mathtt{search}}$: We use the same verifier as from 1.a for k=`False PositiveMonotoneSAT` which will check if there are at least k assignments with 0 and satisfies the `CNF` formula in polynomial time.
2. $\Gamma \in \mathtt{NP}_{\mathtt{search}}$-complete: Since every problem in $\mathtt{NP}_{\mathtt{search}}$ reduces to `Independent Set`, all we need to show is `Independent Set` $\leq_p \Gamma$ since reductions are transitive. We provide a reduction R:

$$\texttt{Independent Set instance } \varphi \xrightarrow{\text{polytime } R} \Gamma \text{ instance } \varphi'$$

We first describe the reduction R for $\varphi(G, k)$, where G is a graph and k is the size of the independent set:

- We form a `CNF` formula $\phi = \bigwedge_{v_i, v_j \in E} (x_i \vee x_j)$. This represents each edge. If we restrict the `CNF` to have k zeroes, then it will result in an independent set of size k, as, for k edges, only one of the two vertices are 0. We can thus create an instance $\varphi'(\phi, k)$.
- Make an oracle call to Γ on φ'. If the oracle returns a satisfying assignment, we select only k variables to set to 0 (if there are more than k) and then return an independent set $S = \{v_i \mid \alpha_i = 0\}$. If the oracle returns $\perp$, then return $\perp$.

Runtime: We check that R runs in polynomial time. To create the `CNF`, we iterate through all edges to create clauses, taking time $O(|E|)$. Setting k takes $O(1)$. Iterating through all vertices for post-processing takes time $O(|V|)$, and thus in total, the runtime is $O(|E| + |V|)$ which is polynomial time.

> **Remark.** No time for proof of correctness :(

> **Claim —** If φ has a solution, then $\varphi' = R(\varphi)$ is satisfiable.

> **Claim —** If α satisfies $R(\varphi)$, then $\alpha' \mid_\varphi$ is a solution for φ, where $\alpha \mid_\varphi$ is the conversion from α to an independent set, and $\alpha \mid_\varphi$ can be obtained in polynomial time.

This completes the proof that k-`False PositiveMonotone 2SAT` is $\texttt{NP}_{\texttt{search}}$-complete. □

§2 Reductions and Complexity Classes

(a) *Proof.* If $\Pi \in \texttt{P}_{\texttt{search}}$, then there exists a program P that can solve Π in polynomial time. Let G be an oracle that can solve Γ. Then, we can solve Π by calling G once and then running P.

Correctness: The proof of correctness is that we call G but do not use the output, and then run P which solves Π.

Runtime: The oracle call takes $O(1)$, and then P runs in polynomial time, and thus the total runtime is polynomial time.

Thus, if $\Pi \in \texttt{P}_{\texttt{search}}$, then $\Pi \leq_p \Gamma$, and the proof is complete. □

(b) *Proof.* Assume $\texttt{NP}_{\texttt{search}} \subseteq \texttt{P}_{\texttt{search}}$. Then, for all computational problems $\Pi \in \texttt{NP}_{\texttt{search}}$, it follows that $\Pi \in \texttt{P}_{\texttt{search}}$. To prove that, if $\Pi \in \texttt{NP}_{\texttt{search}}$ then $\Pi \in \texttt{NP}_{\texttt{search}}$-complete, we must show that:

1. $\Pi \in \texttt{NP}_{\texttt{search}}$: We know $\Pi \in \texttt{NP}_{\texttt{search}}$.
2. $\Pi \in \texttt{NP}_{\texttt{search}}$-hard: From the previous question, 2.1, we know that, if $\Pi \in \texttt{P}_{\texttt{search}}$, then $\Pi \leq_p \Gamma$. From our assumption, every problem Π in $\texttt{NP}_{\texttt{search}}$ is in $\texttt{P}_{\texttt{search}}$, and thus every problem in $\texttt{NP}_{\texttt{search}}$ can reduce to every other problem in $\texttt{NP}_{\texttt{search}}$, and thus all problems in $\texttt{NP}_{\texttt{search}}$ are $\texttt{NP}_{\texttt{search}}$-hard.

This completes the proof that, if $\texttt{NP}_{\texttt{search}} \subseteq \texttt{P}_{\texttt{search}}$, then all problems in $\texttt{NP}_{\texttt{search}}$ are $\texttt{NP}_{\texttt{search}}$-complete. □

(c) *Proof.*

Reduction: Since Π reduces in polynomial time to Γ, there exists a reduction R solving Γ which runs in time $O(n^c)$ for some c on inputs of size n given an oracle that solves Γ.

Algorithm, by oracle replacement: We assume that $\Gamma \in \texttt{EXP}_{\texttt{search}}$, or for some constant d, there is an $O(2^{n^d})$ algorithm A solving Γ on inputs of length n. Then, we can create an algorithm that, given an input x to Π, runs R on x. Whenever R calls the oracle on a value y_i, we instead evaluate $A(y_i)$.

Overall runtime: Since R runs in time $O(n^c)$ and calls the oracle at most once per step, it makes at most $O(n^c)$ calls to the oracle. If each of those calls takes time at most B, then the overall runtime, including reduction and oracle calls, is

$$O(n^c + n^c \cdot B)$$

We now bound B for the oracle calls.

Since R starts with memory of size n, runs in time $O(n^c)$, and increases the size of memory by at most w bits per `MALLOC` call, the size of R's memory is at most $O(n + n^c w)$. As the initial word size w_0 is at most $n + 1$ and w increases by at

most 1 per `MALLOC` call, after $O(n^c)$ steps, we will have $w \leq O(n^c)$. Thus, the size of memory is always $O(n^c n^c) = O(n^{2c})$. As each oracle calls' input is a subset of the memory, each oracle call is on an input of size at most $O(n^{2c})$ as well.

With an input size of $O(n^{2c})$, the algorithm A which we replaced the oracle with runs in time $O(2^{(n^{2c})^d})$. We substitute this bound for B, giving a runtime bound of

$$O(n^c + n^c \cdot (2^{(n^{2c})^d})) = O(n^c 2^{(n^{2c})^d})$$

which is exponential, and the proof is complete. □

§3 Variant of VectorSubsetSum

Proof. Let $\Pi =$ `VectorSubsetSum` and $\Gamma =$ `VectorSubsetSumVariant`.
We will show that Γ is $\texttt{NP}_{\texttt{search}}$-complete by showing that

1. $\Gamma \in \texttt{NP}_{\texttt{search}}$:
   - Solutions are of polynomial size, as the size of the vector subset S is bounded by n since $S \subseteq [n]$.
   - Our verifier can check if a vector subset S by summing corresponding entries for each vector and checking if the sum is equal to the corresponding entry for t_0. This takes $O(nd)$ time, as we sum n entries for each dimension of the vectors, which is polynomial time.

2. $\Gamma \in \texttt{NP}_{\texttt{search}}$-hard: Since `VectorSubsetSum` is $\texttt{NP}_{\texttt{search}}$-complete, it follows that every problem in $\texttt{NP}_{\texttt{search}}$ reduces to `VectorSubsetSum`, and thus all we need to show is that `VectorSubsetSum` $\leq_p \Gamma$ since reductions are transitive. We provide a reduction R:

   $$\texttt{VectorSubsetSum instance } \varphi \xrightarrow{\text{polytime } R} \texttt{VectorSubsetSumVariant instance } \varphi'$$

   We first describe the reduction R for instance φ with vectors $\vec{v}_0, \ldots, \vec{v}_{n-1}, \vec{t}$:
   - Add an additional coordinate 0 for each vector $\vec{v_i}, i \in [n]$, extending the dimension of each vector from d to $d+1$ to form $\vec{v_i'}$. Thus, $\vec{v_i'}[d] = 0$.
   - Find $m = \max(\vec{t}[0], \ldots, \vec{t}[n-1])$ and create a new vector $\vec{t'}$ of length $d+1$ and set each entry to m.
   - Add a new vector $\vec{v_n'}$ by adding an additional coordinate to $\vec{t}$, $\vec{t}[d] = 0$ and then computing coordinate-wise subtraction $\vec{t'} - \vec{t}$. Thus

     $$\begin{aligned} \vec{v_n'}[0] &= \vec{t'}[0] - \vec{t}[0] \\ &\vdots \\ \vec{v_n'}[d] &= \vec{t'}[d] - \vec{t}[d] \end{aligned}$$

   - Make an oracle call to Γ with instance φ' with $\vec{v'}_0, \ldots, \vec{v'}_{n-1}, \vec{v'}_n, \vec{t'}$.
   - If the oracle returns a subset S' to φ', we transform S' into a subset for φ. If it returns $\perp$, we return $\perp$.

Runtime: We check that R runs in polynomial time. Adding an additional coordinate for vectors $\vec{v_i}, i \in [n]$ to form $\vec{v_i'}$ as well as adding an additional coordinate for $\vec{t}$ to be used to form $\vec{v_n'}$ takes time $O(n)$. Finding the maximum value of the entries for $\vec{t}$ takes time $O(d)$, and we create $\vec{t'}$ by setting all $d+1$ entries to m which takes $O(d)$ time. Subtracting $\vec{t'}$ and $\vec{t}$ to form $\vec{v_n'}$ takes time $O(d)$. We make a single oracle call taking $O(1)$ time. Thus, in total, the run time is $O(n) + O(d) + O(d) + O(d) + O(1) = O(n + d)$, which is polynomial time.

> **Claim —** If φ has a solution, then $\varphi' = R(\varphi)$ has a solution.

Proof of claim. If φ has a solution, then if we set $\vec{t}$ to a vector $\vec{t'}$ where all entries are the same, we add a vector $\vec{v_n}$ to $\vec{t}$, where the entries no longer need to be in $\{0, 1\}$. Thus, if we add $\vec{v_n}$ to the solution set S, then it will satisfy φ'. We add an additional coordinate to the vectors, where for $\vec{v_i}, i \in [n]$ we add 0, and for $\vec{v_n'}$ as well as $\vec{t'}$ we add m. For this dimension, the sum will still remain m, as $0 + \cdots + 0 + m = m$, which is the value of $\vec{v_n'}[d]$ and thus φ' still has a solution. ■

> **Claim —** If φ' has a solution, then φ has a solution and we can transform valid solutions to φ' into valid solutions to φ in polynomial time.

Proof of claim. If φ' has a solution, then if we subtract $\vec{t'}$ by the vector whose last coordinate is equal to m and then remove the last coordinate, we can obtain a vector $\vec{t}$. We can then remove that vector from the solution set S' and then remove the last coordinate for all remaining vectors to obtain a solution set S for φ on $\vec{t}$.

The solution set S' has at most $n + 1$ vectors, and thus we can bound the time it takes to find the vector $\vec{v_i'}$ where $\vec{v_i'}[d] = m$ by $O(n)$. Subtracting this vector from $\vec{t'}$ will take $O(d)$, and then removing the vector $\vec{v_i'}$ from S' as well as removing the last coordinate for all remaining vectors will take $O(n)$, and thus the total runtime is $O(n) + O(d) + O(n) = O(n + d)$, which is polynomial time. ■

This completes the proof that `VectorSubsetSumVariant` is $\texttt{NP}_{\texttt{search}}$-complete.

□