Homework 6. Due June 8th, 10PM

CS180: Algorithms and Complexity
Spring 2018

GUIDELINES:

- Upload your assignments to Gradescope by 9:59 PM.
- Follow the instructions mentioned on the course webpage for uploading to Gradescope very carefully (including starting each problem on a new page and matching the pages with the assignments); this makes it easy and smooth for everyone. As the guidelines are simple enough, bad uploads will not be graded.
- You may use results and algorithms from class without proofs or too many details as long as you state what you are using clearly.
- Most importantly, make sure you adhere to the policies for academic honesty set out on the course webpage. The policies will be enforced strictly. Homework is a stepping stone for exams; keep in mind that reasonable partial credit will be awarded and trying the problems will help you a lot for the exams.

1. Consider the problem FIND-IS defined as follows: "Given a graph G and a number k as input, find an independent set of size k in G if one exists." Recall the INDEPENDENT-SET decision problem from class: "Given a graph G and a number k, does G contain an independent set of size at least k?". Give a polynomial-time reduction from FIND-IS to INDEPENDENT-SET. [.75 points]

   Solution. The idea is to use black-box calls to IS to figure out if a vertex v is in the IS or not. For a vertex v in a graph H, let $H - \{v\}$ denote the graph obtained by deleting the vertex v (and all its edges) from H.

   Now, start with a vertex $v \in G$. If $(G - \{v\})$ has a independent set of size k - which can be checked by a black-box call to IS, then we can look for the independent set in $H = G - \{v\}$ and we've made progress as we reduced the size of the graph. On the other hand, if $G - \{v\}$ has no independent set of size k, then we know that v is an *essential* vertex, so we add v to our candidate independent set of size k.

   INPUT: Graph $G = (V, E)$ and integer k.
   OUTPUT: A independent set $I \subseteq V$ of size k or NO if there is no such independent set.

   (a) Use black-box to check if $(G, k) \in$ IS. If not, RETURN NO.
   (b) Set $H = G$, set $I = \emptyset$, and set $size = 0$.
   (c) For each vertex $v \in V$:

```
i. If $(H - \{v\}, k) \in$ IS (using black-box):
  - Set $H = H - \{v\}$.
ii. Else:
  - Add $v$ to $I$.
  - Set $size = size + 1$.
iii. If $size == k$, RETURN $I$.
```

The reason the algorithm works is that when we remove a vertex v from our candidate graph, we know there will still be an independent set in the graph left over (as we explicitly check so in step (c).(i)). On the other hand, an independent set in the graphs H at any point is also an independent set in G as when we remove a vertex, we only remove the edges involving that vertex.

2. Give a polynomial-time Karp reduction from 3SAT to Vertex-Cover. For full-credit for this problem, you should not use transitivity but have to show the three steps as we did in class for the reduction from 3SAT to Independent-Set. [.75 points]

Solution. Let $G_\phi = (V, E)$ be the same graph as we used in class and let $r_\phi = |V| - m$.

Let us argue that ϕ is satisfiable if and only if G_ϕ has a vertex cover of size at most r_ϕ.

Proof of $\Rightarrow$: Suppose ϕ is satisfiable and let x^* be satisfying solution. Then, x^* should satisfy at least one literal from each clause. Pick a vertex from each blob corresponding to each satisfied literal to get a set I. Let $S = V \setminus I$. Then,

(a) I has size exactly m (as we pick one vertex from each blob). So S has size exactly $|V| - m$.

(b) S is a vertex cover as if not, then there must be an edge between two vertices in I (an 'uncovered' edge) which cannot happen as the only edges crossing different blobs denote conflicts and the literals picked out in I are not conflicting.

Proof of $\Leftarrow$: Suppose that G_ϕ has a vertex cover S of size at most r_ϕ. Let $I = V \setminus S$. Then, I has size exactly m and cannot have two vertices in the same blob (as else the edge between them would not be covered by S). Therefore, I should contain exactly one vertex from each blob. Let x^* be the assignment obtained by setting the literals in I to be true (as we did in class). Then, as I does not have any edges, x^* is a valid assignment and as it sets at least one literal from each clause to be true, x^* is a satisfying assignment for ϕ.

3. Consider the problem LPS defined as follows: "Given a matrix $A \in \mathbb{R}^{n \times n}$, a vector $b \in \mathbb{R}^n$ and an integer $k > 0$, does there exist a vector $x \in \mathbb{R}^n$ with at most k non-zero entries such that $A \cdot x \geq b$". Here $A \cdot x$ denotes the usual matrix-vector product and for two vectors u, v, we say $u \geq v$ if for every i, $u_i \geq v_i$. Give a polynomial-time reduction from 3SAT to LPS. [.75 points]
(Hint: Use the reductions done in class along-with transitivity of $\leq_P$ to first pick the "right" starting point and then design a reduction from this starting point.)

Solution. We will show that VERTEX-COVER $\leq_P$ LPS. If we show this, then as $3SAT \leq_P IS$, and $IS \leq_P VC$, we will get by transitivity that $3SAT \leq_P LPS$.

Let ALPS be a minor modification of the above problem defined as follows: Given a matrix $A' \in \mathbb{R}^{m \times n}$, a vector $b' \in \mathbb{R}^m$ and an integer $k > 0$, does there exist a vector $x \in \mathbb{R}^n$ with at most k non-zero entries such that $A' \cdot x \geq b'$. That is ALPS is a version of LPS where the matrix A' need not be square.

We will show that (1) VERTEX-COVER $\leq_P$ ALPS and (2) ALPS $\leq_P$ LPS.

(1): Given an instance of vertex cover with graph G and integer k, define an instance of ALPS as follows. Let $G = (V, E)$ where $E = \{e_1, \dots, e_m\}$ are the edges of the graph G and $V = \{v_1, \dots, v_n\}$ are the vertices. Let A' be the $m \times n$ matrix with $A'_{ij} = 1$ if the edge e_i is adjacent to vertex v_j and 0 otherwise. Let $b' \in \mathbb{R}^n$ be the vector all of whose entries are 1. We claim that G has a vertex-cover of size k if and only if there is a vector y with at most k non-zeros such that $A'y \geq b$.

To prove the forward direction, suppose there is a vertex cover $Y \subseteq [n]$ of size at most k. Let $y = 1(Y)$ be the vector with $y_j = 1$ if $j \in Y$ and 0 otherwise. Now, for any $i \in [m]$, $(A'y)_i = \sum_{j=1}^{n} (A'_{ij} y_j) = \sum_{j \in Y} A'_{ij} \geq 1$, where the last inequality follows because the edge e_i should be adjacent to at least one vertex of Y. Therefore, $A'y \geq b'$.

Similarly, suppose there is a vector $y \in \mathbb{R}^n$ with at most k non-zeros such that $A'y \geq b'$. Let $Y = \{j : y_j \neq 0\}$. Then, $|Y| \leq k$ and we claim that Y is a vertex-cover. To see this, consider an index $i \in [m]$. Then, since $(A'y)_i \geq b_i = 1$, we have that $(A'y)_i \neq 0$. On the other hand, we have $(A'y)_i = \sum_{j=1}^{n} A'_{ij} y_j$. Therefore, for $(A'y)_i$ to be non-zero, one of the summands $A'_{ij} y_j$ should be non-zero. This can only happen if the edge e_i is adjacent to a vertex in Y. Therefore, Y is a vertex-cover.

The above two arguments show that G has a vertex-cover of size k if and only if there is a vector y with at most k non-zeros such that $A'y \geq b'$. Since, we can build the instance of ALPS A', b', k efficiently (in polynomial time), we can use a black-box for ALPS to solve VERTEX-COVER. It follows that VERTEX-COVER $\leq_P$ ALPS.

(2): Consider an instance of ALPS given by A', b', k. If $m = n$, there is nothing to prove as this is also an instance of LPS. If $m > n$, build a matrix $A \in \mathbb{R}^{m \times m}$ by adding $m - n$ columns of length m filled with zeros to the matrix A'. Our new instance of LPS will be specified by A, b', k. It is easy to check that the instance of ALPS has a solution if and only if our constructed instance A, b', k of LPS has a solution. Similarly, if $m < n$, build a matrix $A \in \mathbb{R}^{n \times n}$ by adding $n - m$ rows of length n filled with zeros to the matrix A'. Let $b \in \mathbb{R}^n$ be the vector obtained by adding $n - m$ zeros to the entries of b'. Our new instance of LPS will be specified by A, b, k. As before, it is easy to check that the instance of ALPS has a solution if and only if our constructed instance A, b', k of LPS has a solution. As the new instances of LPS can be constructed efficiently, we can use a black-box for LPS to solve ALPS. Therefore, ALPS $\leq_P$ LPS.

Combining (1) and (2) implies that VERTEX-COVER $\leq_P$ LPS as we wanted to show.

4. Show that the following problems are in NP.

   (a) Given two graphs $G = (V, E)$ and $H = (W, F)$, we say G, H are *isomorphic* if there exists a bijection $\sigma : V \to W$ such that for any two $u, v \in V$, $\{u, v\} \in E$ if and only if

$\{\sigma(u), \sigma(v)\} \in F$ (i.e., two vertices $u, v \in V$ are adjacent if and only if $\{\sigma(u), \sigma(v)\}$ are adjacent).

Let GraphIsomorphism $= \{(G, H) :$ G,H are isomorphic graphs$\}$. Show that GraphIsomorphism is in NP.

(b) Given a graph $G = (V, E)$, a subset of vertices I is a dominating set if every vertex in the graph is either in I or adjacent to a vertex in I. Let DominatingSet $= \{(G, k) : G$ has a dominating set of size $k\}$. Show that DominatingSet is in NP.

Solution. (1) You can take the certificate to be a candidate bijection $\sigma : V \to W$ (as this is just $|V|$ numbers, the size is polynomial). For the verification algorithm B which gets as input (G, H) and σ, you can do the following:

- First check if σ is a bijection. This can be done for instance in $O(|V||W|)$ time easily by counting how many times each element of W appears in the range of σ.
- Form the adjacency-matrices A, C of G, H respectively.
- For all $u, v \in V$: If $A_{u,v} \neq C_{\sigma(u),\sigma(v)}$, then output NO. (We are basically using adjacency-matrices to check that $\{u, v\}$ is an edge in G if and only if $\{\sigma(u), \sigma(v)\}$ is an edge in H.)
- Output YES.

The correctness follows immediately from the definition of the algorithm.

(2). You can take a candidate set I as a certificate. The verification algorithm B takes (G, k) and a set I as input and does the following:

- If $|I|! = k$, return NO.
- For each vertex $v \in G$: Check if v is in I (directly) or if it is adjacent to a vertex in I (you can scan the adjacency-matrices of the vertices in I for v.). If not, return NO.
- Return YES.

The correctness follows immediately from the definition of the algorithm.

Additional Problems. DO NOT turn in answers for the following problems - they are meant for your curiosity and understanding.

1* Given a graph $G = (V, E)$, a valid 3-coloring of a G is a mapping $\chi : V \to \{1, 2, 3\}$ such that for any adjacent vertices u, v, $\chi(u) \neq \chi(v)$. Consider the problem Find-3Coloring defined as follows: "Given a graph G, find a valid 3-coloring of G if one exists." and the 3Coloring decision problem from class: "Given a graph G, is there a valid 3-coloring of G". Give a polynomial-time reduction from Find-3Coloring to 3Coloring. You don't have to prove correctness or analyze time-complexity.

[Hint: Note that if $|V| \geq 4$, then two vertices must get the same color in any proper 3-coloring. Use the black-box access to 3Coloring to find two vertices that can get the same color and use this to decrease the size of the graph you are working with and repeat.]

Solution. There are several ways to solve the problem:

Algorithm 1. This algorithm tries to figure out which vertices can get same color. For H a graph and two vertices u, v in H, let $Merge(H, u, v)$ denote the graph on $|H| - 1$ vertices obtained by combining the two vertices u, v into one vertex that keeps the edges of both vertices. Here, you don't rename the vertices other than u, v. For instance, you could the delete the vertex with the higher index out of the two after copying its edges to the other vertex. Note that given H, u, v you can compute the graph obtained by merging u, v in polynomial-time.

We can use the black-box once to check if G has a valid 3 coloring at the very beginning and say NO if there is no such coloring. In the following, we assume G has a valid 3 coloring and we have to find one.

Let $G = (V, E)$ with $V = \{1, 2, \ldots, n\}$. Now, for two vertices u, v with $u < v$ that do not have an edge in G, if $H = Merge(G, u, v)$ is 3-colorable, then so is G and in fact we can given a coloring χ for H extend it to a coloring for G by setting $\chi(v) = \chi(u)$. This will create a valid coloring as u, v are not adjacent in G and as H as the merger of u, v. Thus, if we had a coloring for H we can get a coloring for G. Here's how we can use it recursively as an algorithm:

FIND3COLORING: Input is a graph $H = (W, F)$ and the output is a valid coloring $\chi : W \rightarrow \{red, blue, green\}$.

(a) If $|W| \leq 3$, just return a coloring by giving a color each to each of the three vertices.

(b) For every two vertices $u < v$ that have no edge between them:

   i. Check if $Merge(H, u, v)$ has a valid 3 coloring (by a call to the black-box). If No, do nothing.

   ii. If Yes, let $\chi \leftarrow$ FIND3COLORING($Merge(H, u, v)$). Set $\chi(v) = \chi(u)$ and RETURNχ.

We would just call FIND3COLORING(G) to find the coloring for G.

The above algorithm runs in polynomial-time barring the calls to the black-box and makes at most n^2 calls to the black-box. Correctness follows from discussion above. Hence, FIND-3COLORING is polynomially easier than 3COLORING.

Algorithm 2. We can use the black-box once to check if G has a valid 3 coloring at the very beginning and say NO if there is no such coloring. In the following, we assume G has a valid 3 coloring and we have to find one. Another idea is to now keep adding edges to the graph until we can't add more edges without violating 3-colorability. When this happens, we will have a graph H that is still 3-colorable, which contaings G as a sub-graph (edges of G are a subset of edges of H) and adding any more edge in H destroys 3-colorability. This intuitively means that H is *tripartitite* the vertices of H can be partitioned into three sets A_1, A_2, A_3 such that a vertex in A_1 is adjacent to every vertex in A_2, A_3 and so on for A_2, A_3. We can use this to get an algorithm as follows:

(a) Let $H \leftarrow G$.

(b) For each $u \neq v \in V$:

   i. If $\{u, v\}$ is an edge, do nothing.

ii. Else, let H' be the graph obtained by adding the edge and check if H' is 3-colorable (by querying the black-box). If Yes, set $H \leftarrow H'$ (add the edge $\{u, v\}$ to H).

(c) Pick a vertex $u_1 \in G$. Set $\chi(u) = \mathit{red}$. For each vertex v that is not adjacent to u in H, set $\chi(v) = \mathit{red}$.

(d) Pick an uncolored u_2 vertex if any. Set $\chi(u_1) = \mathit{blue}$. For each vertex v that is not adjacent to u_1 in H, set $\chi(v) = \mathit{blue}$.

(e) Pick an uncolored u_3 vertex if any. Set $\chi(u_3) = \mathit{green}$. For each vertex v that is not adjacent to u_3 in H, set $\chi(v) = \mathit{green}$.

(f) χ is the required coloring.

The above algorithm runs in polynomial-time barring the calls to the black-box and makes at most n^2 calls to the black-box. Correctness follows from discussion above. Hence, FIND-3COLORING is polynomially easier than 3COLORING.

2 For this problem we need the notion of multi-variate polynomials over variables $x_1, \ldots, x_n$ and how they are specified. To review some terminology, we say a *monomial* is a product of a real-number co-efficient c and each variable x_i raised to some non-negative integer power a_i: we can write this as $cx_1^{a_1}x_2^{a_2}\cdots x_n^{a_n}$. A polynomial is then a sum of a finite set of monomials. (For example, $2x_1^2x_2x_3^4 + x_1x_3 - 6x_2^2x_3^2$ is a polynomial.)

We say a polynomial P is of degree at most d, if for any monomial $cx_1^{a_1}x_2^{a_2}\cdots x_n^{a_n}$ appearing in P, $a_1 + a_2 + \cdots a_n \leq d$. (For example, the degree of the previous polynomial is 7). One can represent a n-variable polynomial of degree d by at most $(n+1)^d$ numbers (by d-dimensional arrays for instance - but let us ignore the actual details of the representation as this is not important and any representation will work for us).

Consider the problem POLY-ROOT defined as follows: "Given a polynomial with integer coefficients of degree at most 6 as input, decide if there exists a $x \in \mathbb{R}^n$ such that $P(x) = 0$." Show that 3SAT is polynomially easier than POLY-ROOT. You don't have to write down the coefficients of the polynomials explicitly in your reduction - you can leave them as summations if it is more convenient for you (one reason why we didn't specify the exact representation).

Hint: Try to define a polynomial equation for each clause in your 3SAT instance along with some other equations for variables and combine them into one big polynomial by using the fact that for any set of numbers $(a_1, \ldots, a_m)$, $\sum_i a_i^2 = 0$ if and only if $a_i = 0$ for all $i \in [m]$.

Solution. For a term y (recall that a term was a variable or its negation), let P_y be the polynomial which is defined as follows: if $y = x_i$ for a variable, then $P_y = 1 - x_i$; if $y = \bar{x_i}$, then $P_y = x_i$. Now, for every clause C, let P_C be the polynomial obtained by multiplying the polynomials P_y for every term y which appears in C. For example, if $C = x_1 \vee \bar{x_2} \vee x_3$, then $P_C = (1 - x_1)x_2(1 - x_3)$.

Given a 3SAT instance $\phi = C_1 \vee C_2 \vee \cdots \vee C_k$, let P_ϕ be the polynomial defined as $P_\phi = P_{C_1}^2 + P_{C_2}^2 + \cdots + P_{C_k}^2$. We claim that ϕ is satisfiable if and only if there exists an $x \in \mathbb{R}^n$ such that $P_\phi(x) = 0$.

Suppose that ϕ is satisfiable and let a be a satisfying assignment for ϕ. Then, $P_{C_j}(a) = 0$ for every clause C_j as at least one of the terms in C_j would be satisfied by a. Therefore, $P_\phi(a) = 0$.

On the other hand, suppose that there exists a vector $b \in \mathbb{R}^n$ such that $P_\phi(b) = 0$. Now define an assignment a as follows: for each $i \in [n]$, if $b_i \in \{0,1\}$, set $a_i = b_i$; else set a_i arbitrarily to 1 (say). We claim that a is a satisfying assignment for ϕ. To see this, first observe that as $P_\phi(b) = 0$, we must in fact have $P_{C_j}(b) = 0$ for every $1 \le j \le k$ (as a sum of squares of real numbers is zero if and only if all of them are zero). Now, let $C_j = y_{j_1} \vee y_{j_2} \vee y_{j_3}$. Since $P_{C_j} = P_{y_{j_1}} P_{y_{j_2}} P_{y_{j_3}}$, $P_{C_j}(b) = 0$ if and only if one of $P_{y_{j_1}}(b), P_{y_{j_2}}(b), P_{y_{j_3}}(b)$ is zero. On the other hand, any of these individual polynomials is zero if and only if the corresponding value of b lies in $\{0,1\}$ and actually satisfies the associated term. Therefore, from our definition of a (it agrees with b on values of b which are in $\{0,1\}$) it follows that a satisfies the clause C_j. As the above argument applies to every clause, we also get that a satisfies the CNF formula ϕ.

The above two arguments show that ϕ has a satisfying assignment if and only if the polynomial P_ϕ has a root. Since we can build the polynomial P_ϕ in polynomial time and it has degree at most 6, we can use a black-box for Poly-Root to solve 3SAT. Therefore, 3SAT $\le_P$ Poly-Root.

3 Chapter 8, Problems 1,3, 6, 26, 27, 30, 31, 41 from [KT]

4 Problems 8.3, 8.4, 8.6, 8.8, 8.14, 8.23 from Chapter 8 of [DPV].