# CS 170 Homework 11

Due **4/18/2022, at 10:00 pm (grace period until 11:59pm)**

## 1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, you must explicitly write "none".

## 2 Max $k$-XOR

In the Max $k$-XOR problem, we are given $n$ boolean variables $x_1, x_2, \ldots, x_n$, a list of $m$ clauses each of which is the XOR of exactly $k$ distinct variables (that is, the clause is true if and only if an odd number of the $k$ variables in the clause are true), and an integer $r$. Our goal is to decide if there is some assignment of variables that satisfies at least $r$ clauses.

(a) In the Max Cut problem, we are given an undirected unweighted graph and integer $c$ and want to find a cut $S$ such that at least $c$ edges cross this cut (i.e. have exactly one endpoint in $S$). Give and argue correctness of a reduction from Max-Cut to Max 2-XOR.

(b) Give and argue correctness of a reduction from Max 3-XOR to Max 4-XOR.

**Solution:**

(a) We create a variable for each vertex $i$, and a clause $x_i$ XOR $x_j$ for each edge $(i, j)$. We choose $c = r$.

For any assignment, consider the cut such that $S$ contains all vertices $i$ for which $x_i$ is true in this assignment. For each edge $(i, j)$ crossing this cut, its corresponding clause is true because exactly one of $x_i, x_j$ is true. For each edge not crossing this cut, its corresponding clause is false because either both $x_i, x_j$ are true or neither is true. This proves correctness of the reduction.

(b) The reduction is to add a new variable $y$ and add $y$ to every clause in the Max 3-XOR instance and choose the same value of $r$ to get a Max 4-XOR instance.

Given an assignment that satisfies $r$ clauses in the Max 3-XOR instance, the same assignment plus $y$ set to false satisfies $r$ clauses in the Max 4-XOR instance. Given an assignment that satisfies $r$ clauses in the Max 4-XOR instance, if $y$ is false, then the same assignment minus $y$ satisfies $r$ clauses in the Max 3-XOR instance. Otherwise, $y$ is true, and the same assignment minus $y$ and negating all other variables satisfies $r$ clauses in the Max 3-XOR instance.

To see this, consider any clause that was true in Max 4-XOR. Deleting $y$ takes it from having an odd number of true variables to an even number. Then, since 3 is odd, negating all variables brings it back to having an odd number of true variables. Similarly, any clause that was false in the Max 4-XOR instance has an even number of true variables. Deleting $y$ causes it to have an odd number of true variables, and then negating all other variables brings it back to even.

## 3 Dominating Set

A dominating set of a graph $G = (V, E)$ is a subset D of V, such that every vertex not in $D$ is a neighbor of at least one vertex in $D$. Let the Minimum Dominating Set problem be the task of determining whether there is a dominating set of size $\leq k$. Show that the Minimum Dominating Set problem is NP-Complete. You may assume that $G$ is connected.

*Hint: Try reducing from Vertex Cover or Set Cover.*

**Solution:**

**Proof that Minimum Dominating Set is in NP.**

Given a possible solution, we can check that it is a solution by iterating through the vertices not in the solution subset $D$ and checking if it has a neighbor in $D$ by iterating through the adjacent edges. This would take at most $E$ time because you would at most check every edge twice. We should also check that the number of vertices in $D$ is less than k, which would take at most $V$ as k should be less than $E$. So, we can verify in $O(E)$ time which is polynomial.

**Proof that Minimum Dominating Set is NP-Hard.**

**Reduction from Vertex Cover to Dominating Set**

Minimum Vertex Cover is to find a vertex cover (a subset of the vertices) which is of size $\leq k$ where all edges have an endpoint in the vertex cover.

Suppose $G(V, E)$ is an instance of vertex cover and we are trying to find a vertex cover of size $\leq k$. For every edge $(u, v)$, we will add a new vertex $c$ and two edges $(c, u)$, and $(c, v)$. We will pass in the same $k$ to the dominating set. This is a polynomial reduction because we are adding $|E|$ vertices and $2|E|$ edges.

Proof of Correctness of Reduction

1. **If minimum vertex cover has a solution, then minimum dominating set that corresponds to it has a solution.**

Suppose we have some minimum vertex cover of size $k$. By definition of vertex cover, every edge has either one or both endpoints in the vertex cover. Thus if we say that the vertex cover is a dominating set, every original vertex must either be in the dominating set or adjacent to it. Now we have to figure out whether the vertices we added for every edge are covered. Since every edge has to have one or both endpoint in the vertex cover, the added vertices must be adjacent to at least one vertex in the vertex cover. Thus the vertex cover maps directly to a dominating set in the transformed problem.

2. **If minimum dominating set in the transformed format has a solution, then the corresponding minimum vertex cover has a solution.**

Suppose we have some minimum dominating set of size $k$ of the transformed format. Vertices in the dominating set either must come from the original vertices or the vertices we added. If they don't come from the vertices we added in the transformation, then from the logic stated in the previous direction, the dominating set corresponds directly to the vertex cover. If some vertices come from the transformation, then we can substitute the vertex for either of the endpoints of the edge it corresponds to without changing the size of the dominating set. Thus, we can come up with a dominating set of size $k$ that only uses vertices from our original problem, which will directly match to a minimum vertex cover of size $k$ in the original

problem.

**Alternative Proof that Minimum Dominating Set is NP-Hard.**
**Reduction from Set Cover to Dominating Set**
Minimum Set Cover is to find a set cover (a subset of all the sets) which is of size $\leq k$ where all elements are covered by at least one set of the set cover.

Suppose $(S, U)$ is an instance of set cover where $U$ denotes the set of all distinct elements and $S$ is a set of subsets $S_i$ of $U$. We will construct a graph $G = (V, E)$ as follows. For each element $u$ in $U$ construct a vertex; we will call these "element vertices". For each possible $S_i$ construct a vertex; we will call these "set vertices". Connect each vertex $S_i$ to all $u$ in $S_i$.

Notice that if we were to run dominating set on the graph right now, we would be able to cover all the element vertices with any valid set cover, but we would have to pick every single set vertex in order to ensure that all set vertices are covered. To rectify this, connect every set vertex to every other set vertex, forming a clique. This ensures that we can cover all the set vertices by picking just one. This way we really only need to worry about covering the element vertices.

Proof of Correctness of Reduction
1. **If minimum set cover has a solution, then minimum dominating set that corresponds to it has a solution.**
Suppose we have some minimum set cover of size $k$. This will correspond to a dominating set of size $k$ as well. For each set in our minimum set cover, pick the corresponding set vertex. It follows directly from the construction of the graph and the definition of a set cover that all set and element vertices are covered.

2. **If minimum dominating set in the transformed format has a solution, then the corresponding minimum set cover has a solution.**
Suppose we have some dominating set $D$ of size $k$. We can find a set cover of size $\leq k$. To do this, we will construct a new dominating set $D'$ that contains only set vertices. Include every set vertex in $D$ in $D'$. For each vertex in our dominating set that is an element vertex, pick any random neighboring set vertex and add it to $D'$. Observe that $|D'| \leq |D|$. Thus if there is a dominating set in $G$ of size $\leq k$, there must be a set cover of size $\leq k$
Since this problem is in NP and is NP-Hard, it must be NP-Complete.

## 4 Independent Set Approximation

In the Max Independent Set problem, we are given a graph $G = (V, E)$ and asked to find the largest set $V' \subseteq V$ such that no two vertices in $V'$ share an edge in $E$.

Given an undirected graph $G = (V, E)$ in which each node has degree $\leq d$, give an efficient algorithm that finds an independent set whose size is at least $1/(d + 1)$ times that of the largest independent set. Only the main idea and the proof that the size is at least $1/(d + 1)$ times the largest solution's size are needed.

**Solution:** Initially, let $G$ be the original graph and $I = \emptyset$. Repeat the process below until $G = \emptyset$:

1. Pick an arbitrary node $v$ in $G$ and let $I = I \cup \{v\}$.
2. Delete $v$ and all its neighbors from the graph.
3. Let $G$ be the new graph.

Notice that $I$ is an independent set by construction. At each step, $I$ grows by one vertex and we delete at most $d+1$ vertices from the graph (since $v$ has at most $d$ neighbors). Hence there are at least $|V|/(d+1)$ iterations. Let $K$ be the size of the maximum independent set. Since $K \leq |V|$, we can use the previous argument to get:

$$|I| \geq \frac{|V|}{d+1} \geq \frac{K}{d+1}$$

## 5 Coffee Shops

A rectangular city is divided into a grid of $m \times n$ blocks. You would like to set up coffee shops so that for every block in the city, either there is a coffee shop within the block or there is one in a neighboring block. (There are up to 4 neighboring blocks for every block). It costs $r_{ij}$ to rent space for a coffee shop in block $ij$.

Write an integer linear program to determine which blocks to set up the coffee shops at, so as to minimize the total rental costs.

(a) What are your variables, and what do they mean?

(b) What is the objective function?

(c) What are the constraints?

(d) Solving the non-integer version of the linear program gets you a real-valued solution. How would you round the LP solution to obtain an integer solution to the problem? Describe the algorithm in at most two sentences.

(e) What is the approximation ratio obtained by your algorithm? Briefly justify.

**Solution:**

(a) There is a variable for every block $x_{ij}$, i.e., $\{x_{ij} | i \in \{1, \ldots, m\}, j \in \{1, \ldots, n\}\}$. This variable corresponds to whether we put a coffee shop at that block or not.

(b) $\min \sum_{i=1}^{m} \sum_{j=1}^{n} r_{ij} x_{ij}$. Alternatively, $\min t$ is correct as well as long as the correct constraint is added.

(c) (i) $x_{ij} \geq 0$ : This constraint just corresponds to saying that there either is or isn't a coffee shop at any block. $x_{ij} \in \{0, 1\}$ or $x_{ij} \in \mathbb{Z}_+$ is also correct.

(ii) For every $1 \leq i \leq m, 1 \leq j \leq n$:

$$x_{ij} + x_{(i+1),j} \mathbb{1}_{\{i+1 \leq m\}} + x_{(i-1),j} \mathbb{1}_{\{i-1 \geq 1\}} + x_{i,(j+1)} \mathbb{1}_{\{j+1 \leq n\}} + x_{i,(j-1)} \mathbb{1}_{\{j-1 \geq 1\}} \geq 1$$

This constraint corresponds to that for every block, there needs to be a coffee shop at that block or a neighboring block.

$\mathbb{1}_{\{i+1 \leq m\}}$ means "1 if $\{i+1 \leq m\}$, and 0 otherwise". It keeps track of the fact that we may not have all 4 neighbors on the edges, for instance.

(iii) If the objective was $\min t$, then the constraint $\sum_{i=1}^{m} \sum_{j=1}^{n} r_{ij} x_{ij} \leq t$ needs to be added.

(d) Round to 1 all variables which are greater than or equal to 1/5. Otherwise, round to 0. In other words, put a coffee shop on $(i, j)$ iff $x_{i,j} \geq 0.2$.

(e) Using the rounding scheme in the previous part gives a 5-approximation.

Notice that every constraint has at most 5 variables. So for every constraint, there exists at least one variable in the constraint which has value $\geq 1/5$ (not everyone is below average). The total cost of the rounded solution is at most $5 \cdot$ LP-OPT, since $r_{ij} \leq 5 r_{ij} x_{ij}$ for any $x_{ij}$ that gets rounded up, and the other $i, j$ pairs contribute nothing to the cost of the rounded solution. Since Integral-OPT $\geq$ LP-OPT (the LP is more general than the ILP), our rounding gives value at most 5 LP-OPT $\leq 5$ Integral-OPT. So we get a 5-approximation.

## 6 Orthogonal Vectors (optional)

In the 3-SAT problem, we have $n$ variables and $m$ clauses, where each clause is the OR of (at most) three of these variables or their negations. The goal of the problem is to find an assignment of variables that satisfies all the clauses, or correctly declare that none exists.

In the orthogonal vectors problem, we have two sets of vectors $A, B$. All vectors are in $\{0,1\}^m$, and $|A| = |B| = n$. The goal of the problem is to find two vectors $a \in A, b \in B$ whose dot product is 0, or correctly declare that none exists. The brute-force solution to this problem takes $O(n^2 m)$ time: We compute all $|A||B| = n^2$ dot products between two vectors in $A, B$, and each dot product takes $O(m)$ time.

Show that if there is a $O(n^c m)$-time algorithm for the orthogonal vectors problem for some $c \in [1, 2)$, then there is a $O(2^{cn/2} m)$-time algorithm for the 3-SAT problem. For simplicity, you may assume in 3-SAT that the number of variables must be even. *Hint: Try splitting the variables in the 3-SAT problem into two groups.*

**Solution:** We use an $O(2^{n/2} m)$-time reduction from 3-SAT to orthogonal vectors. We split the variables into two groups of size $n/2$, $V_1, V_2$. For each group, we enumerate all $2^{n/2}$ possible assignments of these variables. For each assignment $x$ of the variables in $V_1$, let $v_x$ be the vector where the $i$th entry is 0 if the $i$th clause is satisfied by one of the variables in this assignment, and 1 otherwise. We ignore the variables in the clause that are in $V_2$. For example, if clause $i$ only contains variables in $V_2$, then $v_x(i)$ for all $x$. Let $A$ be the $2^{n/2}$ vectors produced this way.

We construct $B$ containing $2^{n/2}$ vectors in a similar manner, except using $V_2$ instead of $V_1$.

We claim that the 3-SAT instance is satisfiable if and only if there is an orthogonal vector pair in $A \times B$. Given this claim, we can solve 3-SAT by making the orthogonal vectors instance in $O(2^{n/2} m)$ time, and then solving the instance in $O((2^{n/2})^c m) = O(2^{cn/2} m)$ time.

Suppose there is a satisfying assignment to 3-SAT. Let $x_1$ be the assignment of variables in $V_1$, and $x_2$ be the assignment of variables in $V_2$. Let $v_1, v_2$ be the vectors in $A, B$ corresponding to $x_1, x_2$. Since every clause is satisfied, one of $v_1(i)$ and $v_2(i)$ must be 0 for every $i$, and so $v_1 \cdot v_2 = 0$. So there is also a pair of orthogonal vectors in the orthogonal vectors instance.

Suppose there is a pair of orthogonal vectors $v_1, v_2$ in the orthogonal vectors instance. Then for every $i$, either $v_1(i)$ or $v_2(i)$ is 0. In turn, for the corresponding assignment of variables in $V_1, V_2$, the combination of these assignments must satisfy every clause. In turn, the combination of these assignments is a satisfying assignment for 3-SAT.

Comment: It is widely believed that SAT has no $O(2^{.999n}m)$-time algorithm - this is called the Strong Exponential Time Hypothesis (SETH). So it is also widely believed that orthogonal vectors has no $O(n^{1.99}m)$-time algorithm, since otherwise SETH would be violated. It turns out that we can reduce orthogonal vectors to string problems such as edit distance and longest common subsequence, and so if we belive SETH then we also believe those problems also don't have $O(n^{1.99})$-time algorithms. The field of research studying reductions between problems with polynomial-time algorithms such as these is known as fine-grained complexity, and orthogonal vectors is one of the central problems in this field.