CS 170 Homework 13 (Optional)

1 Study Group

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

2 Multiway Cut

In the multiway cut problem, we are given a graph $G = (V, E)$ with k special vertices $s_1, s_2, \ldots, s_k$. Our goal is to find the smallest set of edges F which, when removed from the graph, disconnect the graph into at least k components, where each s_i is in a different component. When $k = 2$, this is exactly the min s-t cut problem, but if $k \geq 3$ the problem becomes NP-hard.

Consider the following algorithm: Let F_i be the set of edges in the minimum cut with s_i on one side and all other special vertices on the other side. Output F, the union of all F_i. Note that this is a multiway cut because removing F_i from G isolates s_i in its own component.

(a) Explain how each F_i can be found in polynomial time.

(b) Let F^* be the smallest multiway cut. Consider the components that removing F^* disconnects G into, and let C_i be the set of vertices in the component with s_i. Let F_i^* be the set of edges in F^* with exactly one endpoint in C_i. How many different F_i^* does each edge in F^* appear in? Which is larger: F_i and F_i^*?

(c) Using your answer to the previous part, show that $|F| \leq 2|F^*|$.

(d) **Extra Credit:** how could you modify this algorithm to output F such that $|F| \leq (2 - \frac{2}{k})|F^*|$?

3 Relaxing Integer Linear Programs

As discussed in lecture, Integer Linear Programming (ILP) is NP-complete. In this problem, we discuss attempts to approximate ILPs with Linear Programs and the potential shortcomings of doing so.

Throughout this problem, you may use the fact that the ellipsoid algorithm finds an optimal vertex (and corresponding optimal value) of a linear program in polynomial time.

(a) Suppose that $\vec{x}_0$ is an optimal point for the following arbitrary LP:

$$\begin{aligned} \text{maximize } & c^\top x \\ \text{subject to: } & Ax \leq b \\ & x \geq 0 \end{aligned}$$

Show through examples (i.e. by providing specific canonical-form LPs and optimal points) why we cannot simply (1) round all of the element in $\vec{x}_0$, or (2) take the floor of every element of $\vec{x}_0$ to get good integer approximations.

(b) The MATCHING problem is defined as follows: given a graph G, determine the size of the largest subset of disjoint edges of the graph (i.e. edges without repeating incident vertices).

Find a function f such that:

$$\begin{aligned} \text{maximize } & f \\ \text{subject to: } & \sum_{e \in E, v \in e} x_e \leq 1 \qquad \forall v \in V \\ & 0 \leq x_e \leq 1 \qquad \forall e \in E \end{aligned}$$

is an LP relaxation of the MATCHING problem. Note that the ILP version (which directly solves MATCHING) simply replaces the last constraint with $x_e \in \{0, 1\}$.

(c) It turns out that the polytope of the linear program from part (b) has vertices whose coordinates are all in $\{0, \frac{1}{2}, 1\}$. Using this information, describe an algorithm that approximates MATCHING and give an approximation ratio with proof.

Hint: round up, then fix constraint violations.

(d) There is a class of linear program constraints whose polytopes have only integral coordinates. Let $\mathcal{P}_{>2,\text{odd}}(V)$ be the set of subsets of the vertices with size that is odd and greater than 2. It turns out that, if we simply add to the LP from part (b) the following constraints:

$$\sum_{e \in E(S)} x_e \leq \frac{|S| - 1}{2} \qquad \forall S \in \mathcal{P}_{>2,\text{odd}}(V),$$

then all vertices of the new polytope are integral. First, interpret this constraint in words and explain why it still describes the MATCHING problem. Then, explain what this result implies about approximating ILPs with (special) LPs.

(e) Why doesn't the observation in part (d) imply that MATCHING $\in$ P?

4 Randomization for Approximation

Oftentimes, extremely simple randomized algorithms can achieve reasonably good approximation factors.

(a) Consider Max 3-SAT: given an instance with m clauses each containing exactly 3 distinct literals, find the assignment that satisfies as many of them as possible. Come up with a simple randomized algorithm that will achieve an approximation factor of $\frac{7}{8}$ in expectation. That is, if the optimal solution satisfies c clauses, your algorithm should produce an assignment that satisfies at least $\frac{7c}{8}$ clauses in expectation.

Hint: use linearity of expectation!

(b) Given a Max 3-SAT instance I, let OPT_I denote the maximum fraction of clauses in I satisfied by any variable assignment. What is the smallest value of OPT_I over all instances I? In other words, what is $\min_I \mathsf{OPT}_I$?

Hint: use part (a), and note that a random variable must sometimes be at least its mean.

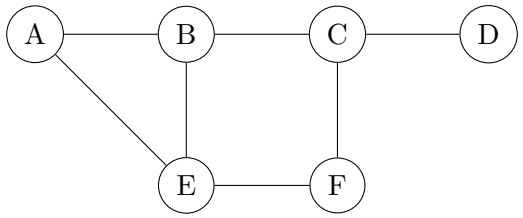
(c) **(Challenge:)** Derandomize your algorithm from part (a), i.e give a deterministic algorithm that achieves the same approximation factor. Justify the correctness of your algorithm.

Disclaimer: this subpart is particularly difficult, so please only attempt this after completing the rest of the homework and if you want extra practice.

5 Vertex Cover to Set Cover

To help jog your memory, here are some definitions:

> **Vertex Cover:** given an undirected unweighted graph $G = (V, E)$, a vertex cover C_V of G is a subset of vertices such that for every edge $e = (u, v) \in E$, at least one of u or v must be in the vertex cover C_V.
>
> **Set Cover:** given a universe of elements U and a collection of sets $\mathcal{S} = \{S_1, \ldots, S_m\}$, a set cover is any (sub)collection C_S whose union equals U.
>
> In the *minimum vertex cover problem*, we are given an undirected unweighted graph $G = (V, E)$, and are asked to find the smallest vertex cover. For example, in the following graph, $\{A, E, C, D\}$ is a vertex cover, but not a minimum vertex cover. The minimum vertex covers are $\{B, E, C\}$ and $\{A, E, C\}$.



> Then, recall in the *minimum set cover problem*, we are given a set U and a collection $\mathcal{S} = \{S_1, \ldots, S_m\}$ of subsets of U, and are asked to find the smallest set cover. For example, given $U := \{a, b, c, d\}$, $S_1 := \{a, b, c\}$, $S_2 := \{b, c\}$, and $S_3 := \{c, d\}$, a solution to the problem is $C_S = \{S_1, S_3\}$.

Give an efficient reduction from the minimum vertex cover problem to the minimum set cover problem. Briefly justify the correctness of your reduction (i.e. 1-2 sentences).

6 Dijsktra's Sort

Show how to use Dijsktra's algorithm to sort n real numbers (not necessarily non-negative or integral) in ascending order in $O(n \log n)$ time. You may use the intermediate outputs of Dijsktra's to do the sorting (as opposed to using it as a blackbox).

Argue that this means that improving upon the $O(m + n \log n)$ run-time of Dijsktra's algorithm (with Fibonacci heap) would lead to a faster sorting algorithm than merge and quick sort.
Hint: Given the numbers, construct a graph such that the order of vertices being extracted from priority queue by Dijsktra's algorithm corresponds to the right sorting order.

7 Orthogonal Vectors

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: 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.

8 Local Search for Max Cut

Sometimes, local search algorithms can give good approximations to NP-hard problems. Recall that in the Max-Cut problem, we have an unweighted graph $G = (V, E)$ and we want to find a cut (S, T) that maximizes the number of edges "crossing" the cut (i.e. with one endpoint in each of S, T). Consider the following local search algorithm:

1. Start with any cut (e.g. $(S, T) = (V, \emptyset)$).
2. While there is some vertex $v \in S$ such that more edges cross $(S \setminus \{v\}, T \cup \{v\})$ than (S, T) (or some $v \in T$ such that more edges cross $(S \cup \{v\}, T \setminus \{v\})$ than (S, T)), move v to the other side of the cut.

Now, let us prove a couple of guarantees that this algorithm achieves.

(a) Give an upper bound on the number of iterations this algorithm can run for (i.e. the total number of times we move a vertex).

(b) Show that when this algorithm terminates, it finds a cut where at least half the edges in the graph cross the cut.

Hint: when we move v from S to T, v must have more neighbors in S than T. What does this observation suggest about the neighbors of each vertex once the algorithm terminates? Then, what can we say about the number of edges crossing the cut vs. the number of edges within each side of the cut?

