**18.415/6.854 Advanced Algorithms** October 10, 2001

# Problem Set Solution 2

1. **The Min $s$-$t$-Cut problem is the following:**

   ***Given an undirected graph*** $G = (V, E)$***, a weight function*** $w : E \to \mathbb{R}^+$***, and two vertices*** $s, t \in V$***, find***

   $$Min\ s - t - Cut(G) = \min\{w(\delta(S)) : S \subset V, s \in S, t \notin S\}.$$

   ***where*** $\delta(S)$ ***denotes the cut***

   $$\delta(S) = \{(i, j) \in E : |\{i, j\} \cap S| = 1\}$$

   ***and***

   $$w(\delta(S)) = \sum_{e \in \delta(S)} w(e).$$

   (a) **Argue (in just a few lines) that there is a polynomial-time algorithm to find a Min $s - t$-cut based on linear programming (remember Problem Set 1). (Be careful; problem set 1 defined the Min $s-t$-cut problem for a directed graph, while this problem considers undirected graphs.) [We will see a much more efficient algorithm for it (not based on linear programming) later this semester.]**

   The description of the (directed) Min $s-t$-cut problem as a linear program from Problem 1 shows immediately that we can solve it in polynomial time, for example by the ellipsoid algorithm. Given an undirected graph, we can transform it into a directed graph simply by producing two directed edges (one each way) for every undirected edge. Then cuts in the directed graph correspond exactly to cuts in the undirected graph, with the same weight.

   **We are going to develop an algorithm for a generalization of the problem:**

   ***Given an undirected graph*** $G = (V, E)$, $w : E \to \mathbb{R}^+$***, and an even cardinality subset of vertices*** $T \subseteq V$***, find***

   $$Min\ T - Odd - Cut(G) = \min\{w(\delta(S)) : S \subset V, |S \cap T| = \textbf{\textit{odd}}\}$$

   **That is, we want to optimize over all cuts that separate $T$ into two parts of odd size (since $|T|$ is even, $|S \cap T|$ odd implies that $|T \setminus S|$ odd as well).**

(b) **Suppose that $|T| = 2$, say $T = \{s,t\}$. What is the Min $T$-Odd-Cut then?**

Since any $T$-odd cut must split $T$ into two odd parts, there is only one way to do that - have $s$ and $t$ on opposite sides of the cut. Thus the Min $T$-odd cut is exactly the Min $s-t$ cut.

(c) **For a given $T \subseteq V$, call a cut $\delta(S)$ $T$-splitting if $\emptyset \neq S \cap T \neq T$.**

**Using a $s$-$t$-Min-Cut algorithm, show how we can find the minimum $T$-splitting cut in polynomial time. Can you do it in at most $|T|$ calls to a Min $s$-$t$-Cut algorithm?**

Choose a fixed $s \in T$. For every cut $\delta(S)$, we can assume $s \in S$ (the graph is undirected and $V \setminus S$ defines the same cut). Moreover, for every $T$-splitting cut, there exists a vertex $t \in T \setminus S$ and such a cut is an $s-t$-cut as well. It is sufficient to find the minimum $s$-$t$-cut for every $t \in T \setminus \{s\}$ and take the minimum of all these cuts.

(d) **For any two sets $C$ and $D$ ($\emptyset \neq C, D \subset V$), prove the inequality that**

$$w(\delta(C \setminus D)) + w(\delta(D \setminus C)) \leq w(\delta(C)) + w(\delta(D)).$$

We apply the definition of a cut - $\delta(X)$ is the set of all edges between $X$ and $V \setminus X$:

$$\begin{aligned}
&w(\delta(C \setminus D)) \;+\; w(\delta(D \setminus C)) = \\
w(\delta(C)) \;+\; &\sum_{i \in C \cap D} \Big( \sum_{j \in C \setminus D} w_{ij} \;-\; \sum_{j \in V \setminus C} w_{ij} \Big) \;+ \\
w(\delta(D)) \;+\; &\sum_{i \in C \cap D} \Big( \sum_{j \in D \setminus C} w_{ij} \;-\; \sum_{j \in V \setminus D} w_{ij} \Big) \;\leq \\
w(\delta(C)) \;+\; &\sum_{i \in C \cap D} \Big( \sum_{j \in V \setminus D} w_{ij} \;-\; \sum_{j \in V \setminus C} w_{ij} \Big) \;+ \\
w(\delta(D)) \;+\; &\sum_{i \in C \cap D} \Big( \sum_{j \in V \setminus C} w_{ij} \;-\; \sum_{j \in V \setminus D} w_{ij} \Big) \;= \\
&w(\delta(C)) + w(\delta(D)).
\end{aligned}$$

(e) **Prove that if $\delta(C)$ is a minimum $T$-splitting cut then there is a minimum $T$-odd-cut $\delta(D)$ such that either $D \subseteq C$ or $C \subseteq D$.**

**Hint: Use the inequality proved above.**

Suppose $\delta(C)$ is the minimum $T$-splitting cut. If $|C \cap T|$ is odd then $C$ is also the minimum $T$-odd cut and we can choose $D = C$. Otherwise suppose $|C \cap T|$ is even, while $\delta(D')$ is the minimum $T$-odd cut.

Since $|D' \cap T|$ is odd, either $|(D' \cap C) \cap T|$ or $|(D' \setminus C) \cap T|$ is odd. Denote by $C'$ either $C$ or $V \setminus C$ so that $|(D' \cap C') \cap T|$ is odd. Note that in any case $\delta(C') = \delta(C)$ and $|C' \cap T|$ is even.

Since $\delta(C')$ is a $T$-splitting cut, $(V \setminus C') \cap T$ is nonempty which implies that either $(D' \setminus C') \cap T$ is nonempty or $((V \setminus D') \setminus C') \cap T$ is nonempty. Without loss of generality, we can assume that $(D' \setminus C') \cap T$ is nonempty, otherwise we rename $D'$ to $V \setminus D'$ which doesn't change the cut (and the new $D'$ still satisfies $|(D' \cap C') \cap T|$ is odd since $|C' \cap T|$ is even).

Because $\delta(C')$ is the smallest $T$-splitting cut and $\delta(D' \setminus C')$ is a $T$-splitting cut, we have $w(\delta(C')) \leq w(\delta(D' \setminus C'))$. Since $|(C' \cap D') \cap T|$ is odd, $|(C' \setminus D') \cap T|$ is odd as well. The smallest $T$-odd cut is $\delta(D')$, so $w(\delta(D')) \leq w(\delta(C' \setminus D'))$. Comparing these two inequalities with the one given in the hint, $w(\delta(C')) + w(\delta(D')) \geq w(\delta(C' \setminus D')) + w(\delta(D' \setminus C'))$, we find that they must all hold with equality and $\delta(C' \setminus D')$ is a minimum $T$-odd cut as well.

In case $C' = C$, we can choose $D = C' \setminus D'$ and we have a minimum $T$-odd cut $D \subset C$. In case $C' = V \setminus C$, we can choose $D = V \setminus (C' \setminus D')$ and we have a minimum $T$-odd cut such that $C \subset D$.

(f) **Use the previous observation to design a recursive algorithm which solves Min $T$-Odd-Cut in polynomial time. (Hint: possibly think about modifying the graph.) How many calls (in $O(\cdot)$ notation) to a Min $s$-$t$-Cut algorithm does your algorithm perform?**

---

```
MinOddCut (G, T)
{
C = MinCut (G, T);
if (|C ∩ T| = odd) return C;
G_1 = Contract (G, C);
G_2 = Contract (G, V \ C);
C_1 = MinOddCut (G_1, T \ C);
C_2 = MinOddCut (G_2, T ∩ C);
if (weight(δ(C_1)) < weight(δ(C_2)) return C_1;
else return C_2;
}
```

---

Here, `MinCut(G, T)` is supposed to return the minimum $T$-splitting cut in $G$ and `Contract(G, C)` should merge $C$ into a single vertex and update the edges accordingly (i.e. any edge between a vertex $u$ of $C$ and a vertex

$v$ not in $C$ becomes a new edge between the new shrunk vertex and $v$; if there are multiple edges between two vertices, we can replace them by one edge with weight equal to the sum of the weights).

The correctness of the algorithm follows from the previous observations. Either the minimum cut $\delta(C)$ is $T$-odd, or we can assume that the minimum $T$-odd-cut is $\delta(D)$ where $C \subseteq D$ or $D \subseteq C$. Cuts $\delta(D)$ where $C \subseteq D$ are equivalent to cuts in the graph $G_1$ where $C$ is contracted to a single vertex. Cuts $\delta(D)$ where $D \subseteq C$ are equivalent to cuts in the graph $G_2$ where $V \setminus C$ is contracted to a single vertex. The smaller of the two cuts must be the minimum $T$-odd cut.

Finally, let's analyze the running time of this algorithm. The body of the function (excluding the recursive calls) runs in time polynomial in the size of the input graph (MinCut algorithm + elementary graph operations). It remains to estimate the number of recursive calls to `MinOddCut`. Denote the size of the input set $T$ by $t$. Note that if the function is called with parameter $T$ and it produces recursive calls with parameters $T_1$ and $T_2$, then $|T| = |T_1| + |T_2|$. Since $|T_i| \geq 2$ in the leaves of the recursion tree, the number of leaves is at most $\frac{t}{2}$. The tree is binary, so the number of nodes is at most $t$. Therefore the total number of recursive calls to `MinOddCut` is linear in $|T|$.

Each call to `MinOddCut` will require a number of calls to a Min $s-t-$cut algorithm less than $t = |T|$. Hence, the total number of calls to a Min $s-t-$cut algorithm is $O(|T|^2)$. (By studying the problem, one can actually solve the Min $T$-odd-cut problem with $O(|T|)$ calls to a Min $s-t$-cut algorithm.)

2. **Use the ellipsoid method to solve the minimum weight perfect matching problem (there is a more efficient combinatorial algorithm for it, but here we will use the power of the ellipsoid algorithm):**

   ***Given an undirected graph*** $G = (V, E)$ ***and a weight function*** $w : E \to \mathbb{N}$***, find a set of edges*** $M$ ***covering every vertex exactly once (a perfect matching) with the minimum total weight.***

   **In order to formulate this problem as a linear program, we define the perfect matching polytope:**

$$P = conv\{\chi_M \in \{0,1\}^E : M \textbf{ is a perfect matching}\}$$

   **where** $\chi_M$ **is the characteristic vector of** $M$ **(**$\chi_M(e) = 1$ **if** $e \in M$ **and 0 otherwise). The convex hull** $conv(A)$ **is defined as** $\{\sum_i \lambda_i x_i : x_i \in A, \lambda_i \geq 0, \sum_i \lambda_i = 1\}$ **(where the summation is finite).**

(a) **Argue that the vertices of $P$ are the characteristic vectors of perfect matchings. Deduce that if we can optimize $\sum_e w_e x_e$ over $P$, we would find a minimum weight perfect matching.**

Any point in $P$ can be written as

$$x = \sum_M \lambda_M \chi_M$$

where $\lambda_M \geq 0$ and $\sum_M \lambda_M = 1$. Clearly, $x$ can be a vertex only if we have exactly one $\lambda_M = 1$. We will show that all such vectors are indeed vertices. For a given $M$, consider the hyperplane

$$H_M = \{x : \sum_{e \in M} x_e = \frac{n}{2}\}$$

where $n = |V|$ (an even number). Note that every perfect matching has exactly $\frac{n}{2}$ edges. Then for any $x \in P$,

$$\sum_{e \in M} x_e \leq \frac{n}{2}$$

because $0 \leq x_e \leq 1$. Equality can hold only if $\forall e \in M; x_e = 1$ but then $x$ is the characteristic vector of $M$. Therefore $P \cap H_M = \{\chi_M\}$ which proves it is a vertex.

The optimum of $\sum_e w_e x_e$ can be assumed to be a vertex $\chi_M$ which means that for any other perfect matching $M'$, $w(M') \geq w(M)$.

(b) **Suppose now that we can decide (via linear programming or some other way) whether $P \cap \{x : w^T x \leq \lambda\}$ is empty or not, for any given $\lambda$ (remember all weights $w_e$ are integers). Show that by calling an algorithm for this decision problem a polynomial number of times (in the size of the input, i.e. $|V|$, $|E|$ and $\log(w_{max})$), we can find the weight of the minimum-weight perfect matching.**

We can find the minimum weight by binary search. If the graph has $n$ vertices and maximum edge weight $w_{max}$, the maximum possible weight of a perfect matching is $\frac{1}{2} n w_{max}$. For any $\lambda \in [0; \frac{1}{2} n w_{max}]$, we are able to test whether there exists a perfect matching of weight at most $\lambda$ (that's exactly when $P \cap \{x : w^T x \leq \lambda\} \neq \emptyset$). The weights are integers, so we can pinpoint the smallest such $\lambda$ in $O(\log(n w_{max}))$ steps.

(c) **With the same assumptions as in the previous part, can you also find a minimum-weight perfect matching (not just its weight, but also which edges are in it) in polynomial time? (There might be several perfect matching having the same minimum weight, but**

**here you need to produce only one of them. Also, the algorithm does not need to be extremely efficient, just polynomial.)**

For any edge, we can determine if we need it for the optimal perfect matching. First, find the minimum weight $W^*$. Then pick an edge $e_1$, remove it from the graph and test if there is still a perfect matching of weight $W^*$. If yes, we don't need edge $e_1$ and we continue on the graph $G \setminus \{e_1\}$. Otherwise we know that $e_1$ appears in any optimal perfect matching, so we remember it, remove its two vertices from the graph, and continue on the remaining graph with modified optimum weight $W' = W^* - w(e_1)$. In $|E|$ steps, we determine the optimal perfect matching.

**Due to Jack Edmonds, the perfect matching polytope can be described by the following inequalities:**

- $\forall e \in E; x_e \geq 0$
- $\forall v \in V; \sum_{e\in\delta(\{v\})} x_e = 1$
- $\forall W \subset V, |W| = \mathbf{odd}; \sum_{e\in\delta(W)} x_e \geq 1$

(d) **Show that every vector in $P$ satisfies the above inequalities.**

Suppose $x$ is the characteristic vector of a perfect matching. Then the first two inequalities are satisfied by definition. For the last inequality, consider and odd-size subset $W \subset V$. All vertices of $W$ cannot be covered by edges inside $W$ because these edges cover disjoint pairs of vertices. At least one vertex must be covered by an edge $e \in \delta(W)$ and therefore

$$\sum_{e\in\delta(W)} x_e \geq 1.$$

Since these inequalities are valid for the vertices of $P$, they are also valid for any point inside $P$.

**Take the other implication for granted (every vector satisfying these inequalities is in $P$).**

(e) **How many inequalities do we have in this complete description of $P$? Can we just use any polynomial-time algorithm for linear programming to optimize over $P$?**

Unfortunately, the third condition generates $2^{n-2}$ inequalities (one for each odd subset, the same equality for $W$ and $V \setminus W$). Therefore a straightforward linear programming approach would be very inefficient (not polynomial in $n$, the number of vertices, and $\log w_{max}$).

(f) **Show how we can use the ellipsoid method to decide if there exists a perfect matching of weight at most $\lambda$ in polynomial time. How would you select the initial ellipsoid? How would you take care of the equality constraints in the description of $P$? When can you stop?**

By adding the inequality $w^T x \leq \lambda$, we get a polytope $P_\lambda$ which is nonempty exactly if there exists a perfect matching of weight at most $\lambda$.

The ellipsoid algorithm can be used to test whether $P_\lambda = \emptyset$ whenever we can:

- find a suitable bounding ellipsoid to start with,
- have a polynomial-time separation oracle, and
- estimate the minimum volume of $P_\lambda$, if it's nonempty.

The bounding ellipsoid here is simple. We can take for example the sphere with center in the origin and radius $\sqrt{\frac{n}{2}}$. This contains all characteristic vectors of perfect matchings.

If a point $x$ doesn't lie in $P_\lambda$, it's because it violates some of the conditions. The condition $w^T x \leq \lambda$ is easy, as well as the first two inequalities in the description of $P$. The third inequality seems to require an exponential number of inequality checks but here's where Problem 1 comes into play. For a given $x$, we can calculate in polynomial time

$$\gamma(x) = \min\{\sum_{e \in \delta(W)} x_e : W \subset V, |W| = \text{odd}\}$$

because this is just a min-$V$-odd-cut problem. Then we check whether $\gamma(x) \geq 1$. In case $\gamma(x) < 1$, we can report that $x$ violates the inequality for $W$ where $\delta(W)$ is the minimum $V$-odd cut. Otherwise, we are guaranteed that no such cut exists.

Finally, we have to make sure that $P_\lambda$ has some volume so we know when to stop. We do this by employing the theorem given in class (theorem 2 of the scribe notes of lecture 5) which states that $\{x : Ax \leq b\}$ is nonempty if and only if $\{x : Ax \leq b + \epsilon e\}$ is nonempty as well, where $\epsilon$ can be chosen as $2^{-L}$. The value $L$ as defined in class involves the number of rows as well, which is enormous, but this is not needed here. We can simply redo the proof more carefully. We have to consider a vertex of $y \geq 0$, $A^T y = 0$, and $b^T y = -1$, where the matrix $\begin{pmatrix} A^T \\ b^T \end{pmatrix}$ has entries all 0 and 1 except for one column containing $w_j$'s and $\lambda$ (which can be assumed to be at most $mw_{max}$). Most of the entries of such a basic feasible solution $y$ will be 0, the ones that are non-zero (basic, and thus at most $m$ of them) will be at most $m!mw_{max}$. Hence, following the proof of Theorem 5, we can choose $\epsilon$ to be say $\frac{1}{2m!m^2 w_{max}} = 2^{-Q}$ with $Q = O(m \log m + \log w_{max})$, which happens to be polynomial in $m$ and $\log w_{max}$. Therefore we replace each equality by a pair of inequalities and increase the right-hand size by $\epsilon = 2^{-Q}$. We have to slightly modify our separation algorithm (since now we are separating over this slightly modified polytope) but this is trivial since we simply compare the value of the minimum $V$-odd-cut to $1 - \frac{1}{2^Q}$ instead of 1. In summary, this guarantees that we can stop after a polynomial number of steps ($O(m^2 Q) = O(m^3 \log m + m^2 \log w_{max})$) and either find a point in $P'$ or declare it empty.