6.854J / 18.415J Advanced Algorithms
Fall 2008

**18.415/6.854 Advanced Algorithms**

# Problem Set Solution 3

1. **Consider the following optimization problem:**

   ***Given*** $c \in \mathbb{R}^n$, $c \geq 0$, $n$ ***even, find***

$$\min\{c^T x : \textstyle\sum_{i \in S} x_i \geq 1 \quad \forall S \subset \{1, \ldots, n\}, |S| = \frac{n}{2}, \\ x_j \geq 0 \quad \forall j\}.$$

   **In class, it was shown that this can be solved by the ellipsoid method because there is an efficient separation algorithm. However, this problem has a more straightforward solution.**

   **Develop an algorithm which finds the optimum in $O(n \log n)$ time. Prove its correctness.**

   Let

$$P = \{x \geq 0 : \sum_{i \in S} x_i \geq 1, \forall S \subset [n]; |S| = \frac{n}{2}\}.$$

   We would like to describe the structure of $P$, which is an unbounded polyhedron. We prove that $x \in P$ exactly when $x$ can be written as

$$x = \sum_{A \subseteq [n]} \lambda_A \chi_A$$

   where $\chi_A$ denotes the characteristic vector of $A$, $\lambda_A \geq 0$, and additionally

$$(*) \qquad \sum_{|A| > n/2} (|A| - \frac{n}{2}) \lambda_A \geq 1.$$

   First, suppose $x$ satisfies this and consider $S$ of size $n/2$. Any set $A$ of size $|A| > n/2$ intersects $S$ in at least $|A| - n/2$ elements, therefore

$$\sum_{i \in S} x_i = \sum_{i \in S} \sum_{A; i \in A} \lambda_A = \sum_{A} |A \cap S| \lambda_A \geq \sum_{A; |A| > n/2} (|A| - \frac{n}{2}) \lambda_A \geq 1.$$

   Conversely, let $x \in P$. Let $\pi$ be a permutation such that

$$x_{\pi(1)} \leq x_{\pi(2)} \leq \ldots \leq x_{\pi(n)}.$$

Set

$$\lambda_1 = x_{\pi(1)}$$
$$\lambda_k = x_{\pi(k)} - x_{\pi(k-1)}$$

and

$$A_k = \{\pi(k), \pi(k+1), \ldots, \pi(n)\}$$

for $k = 1 \ldots n$. Then obviously $\lambda_k \geq 0$ and

$$x = \sum_{k=1}^{n} \lambda_k \chi_{A_k}.$$

Finally, we verify condition $(*)$:

$$\sum_{|A|>n/2} (|A| - \frac{n}{2})\lambda_A = \sum_{k=1}^{n/2} (|A_k| - \frac{n}{2})\lambda_k = (\frac{n}{2})x_{\pi(1)} + (\frac{n}{2} - 1)(x_{\pi(2)} - x_{\pi(1)})$$

$$+(\frac{n}{2} - 2))(x_{\pi(3)} - x_{\pi(2)}) + \ldots + (x_{\pi(n/2)} - x_{\pi(n/2-1)}) = \sum_{k=1}^{n/2} x_{\pi(k)} \geq 1.$$

Now we can optimize over $P$ much more easily. First, observe that for any optimal solution

$$x^* = \sum_A \lambda_A \chi_A,$$

we can assume $\lambda_A = 0$ for $|A| \leq n/2$ and

$$\sum_{|A|>n/2} (|A| - \frac{n}{2})\lambda_A = 1,$$

otherwise we decrease the coefficients until the equality holds. This won't increase the objective function $\sum c_i x_i$, since $c \geq 0$. Therefore an optimal solution always exists in the convex hull of $\{p_A : |A| > n/2\}$ where

$$p_A = \frac{1}{|A| - n/2}\chi_A.$$

We could evaluate the objective function at all these points but there are still too many of them. However, we can notice that for a given $k = |A|$, the only candidate for an optimum $p_A$ is the set $A$ which contains the $k$ smallest components of $c$. Therefore the algorithm is the following:

- Sort the components of $c$ and let $A_k$ denote the indices of the $k$ smallest components of $c$, for each $k > n/2$. This takes $O(n \log n)$ time.

- For each $k > n/2$, calculate $s_k = \sum_{i \in A_k} c_k$. This can be done in $O(n)$ time, because the sets $A_k$ form a chain and we can use $s_k$ to calculate $s_{k+1}$ in constant time.
- Find the smallest value of

  $$c^T p_{A_k} = \frac{s_k}{k - n/2}$$

  for $k > n/2$. Return this as the optimum.

The algorithm runs in $O(n \log n)$ time and its correctness follows from the analysis above.

2. **Fill a gap in the analysis of the interior point algorithm:**

   **Suppose that $(x, y, s)$ is a feasible vector, i.e. $x > 0$, $s > 0$,**

   $$Ax = b,$$

   $$A^T y + s = c$$

   **and we perform one Newton step by solving for $\Delta x, \Delta y, \Delta s$:**

   $$A\Delta x = 0$$

   $$A^T \Delta y + \Delta s = 0$$

   $$\forall j; \quad x_j s_j + \Delta x_j s_j + x_j \Delta s_j = \mu$$

   **where $\mu > 0$. The proximity function is defined as**

   $$\sigma(x, s, \mu) = \sqrt{\sum_j (\frac{x_j s_j}{\mu} - 1)^2}.$$

   **Prove that if**

   $$\sigma(x + \Delta x, s + \Delta s, \mu) < 1$$

   **then $(x + \Delta x, y + \Delta y, s + \Delta s)$ is a feasible vector for $Ax = b, x > 0$ and $A^T y + s = c, s > 0$.**

   The equalities are satisfied directly by the assumptions:

   $$A(x + \Delta x) = Ax + A\Delta x = b$$

   $$A^T(y + \Delta y) + (s + \Delta s) = (A^T y + s) + (A^T \Delta y + \Delta s) = c.$$

   We have to verify the positivity conditions. First we prove that at least one of $x_j + \Delta x_j$, $s_j + \Delta s_j$ is positive. We have $x_j > 0, s_j > 0$ and

   $$x_j s_j + \mu = 2x_j s_j + \Delta x_j s_j + x_j \Delta s_j = (x_j + \Delta x_j)s_j + x_j(s_j + \Delta s_j) > 0$$

therefore either $x_j + \Delta x_j$ or $s_j + \Delta s_j$ must be positive.

Second, we use the proximity condition:

$$(\sigma(x + \Delta x, s + \Delta s, \mu))^2 = \sum_j (\frac{(x_j + \Delta x_j)(s_j + \Delta s_j)}{\mu} - 1)^2 < 1.$$

In particular, for each $j$

$$\frac{(x_j + \Delta x_j)(s_j + \Delta s_j)}{\mu} > 0$$

which means that $x_j + \Delta x_j$ and $s_j + \Delta s_j$ have the same sign. We know they can't be negative so they must be positive.

3. **Given a directed graph $G = (V, E)$ and two vertices $s$ and $t$, we would like to find the maximum number of edge-disjoint paths between $s$ and $t$ (two paths are edge-disjoint if they don't share an edge). Denote the number of vertices by $n$ and the number of edges by $m$.**

   (a) **Argue that this problem can be solved as a maximum flow problem with unit capacities. Explain.**

   Let $F$ be a union of $k$ edge-disjoint paths from $s$ to $t$. We define a flow of value $k$ in a natural way - an edge gets a flow of value 1 if it is contained in $F$ and and 0 otherwise. Since each path enters and exits any vertex (except $s$ and $t$) the same number of times, flow conservation holds. The value of the flow is the number of edges in $F$ leaving $s$ (or entering $t$) which is $k$.

   Conversely, let $f$ be the maximum flow with unit capacities. As we shall prove, there is always a $0-1$ maximum flow, therefore we can assume that $f_{ij}$ is either 0 or 1 for each edge. Let

   $$F = \{(i, j) \in E : f_{ij} = 1\}$$

   and $k$ be the value of the flow. Then we can decompose $F$ into $k$ edge-disjoint paths in the following way: We start from $s$ and follow a path of edges in $F$ until we hit $t$. (This is possible due to flow conservation.) When we have found such a path, we remove it from $F$ and consider the remaining flow of value $k - 1$. By induction, we find exactly $k$ such paths.

   (b) **Consider now the maximum flow problem on directed graphs $G = (V, E)$ with unit capacity edges (although some of the questions below would also apply to the more general case).**

   **Given a feasible flow $f$, we can construct the *residual network* $G_f = (V, E_f)$ where**

   $$E_f = \{(i, j) : ((i, j) \in E \ \& \ f_{ij} < u_{ij}) \textbf{ or } ((j, i) \in E \ \& \ f_{ji} > 0)\}.$$

**The residual capacity of an edge $(i,j) \in E_f$ is equal to $u_{ij} - f_{ij}$ or to $f_{ji}$ depending on the case above. Since we are dealing with the unit capacity case, all the $u_{ij}$'s are 1 and therefore for $0-1$ flows $f$ (i.e. flows for which the value on any edge is 0 or 1), all residual capacities will be 1.**

**We define the distance of a vertex $l_f(v)$ as the length of the shortest path from $s$ to $v$ in $E_f$ ($\infty$ for vertices which are not reachable from $s$ in $E_f$). Further, define the *levelled residual network* as**

$$E_f^l = \{(i,j) \in E_f : l_f(j) = l_f(i) + 1\}$$

**and a *saturating flow* $g$ in $E_f^l$ as a flow in $E_f^l$ (with capacities being the residual capacities) such that every directed $s-t$ path in $E_f^l$ has at least one saturated edge (i.e. an edge whose flow equals the residual capacity).**

**For a unit capacity graph and a given $0-1$ flow $f$, show how we can find the levelled residual network and a saturating flow in $O(m)$ time.**

First, we can find $E_f$ in $O(m)$ time simply by testing each edge and adding the edge or its reverse to $E_f$, depending on the current flow. Then we can label the vertices by $l_f(v)$ by a breadth-first search from $s$. This takes time $O(m)$, also. At the same time we find $d(f)$ as the length of the shortest path from $s$ to $t$.

Then, we create $E_f^l$ by keeping only the edges between successive levels. Thus all paths between $s$ and $t$ in $E_f^l$ have length $d(f)$. Now we produce flow $g$ by finding as many edge-disjoint $s$-$t$ paths as possible. We start with $E' = E_f^l$ and we perform a depth-first search from $s$. If we get stuck, we backtrack and remove edges on the dead-end branches since these are not in any $s$-$t$ path anyway. When we find an $s$-$t$ path, we set $g_{ij} = 1$ along that path, and remove it from $E'$. We continue searching for paths until $E'$ is empty. We spend a constant time on each edge before it's removed, which is $O(m)$ time total. When we are done, there is no $s$-$t$ path in $E_f^l$ without a saturated edge, otherwise it would still be in $E'$.

(c) **Prove that if the levelled residual network has no path from $s$ to $t$ ($l_f(t) = \infty$), then the flow $f$ is maximum.**

Suppose there is a flow $f^*$ of greater value. Then $f^* - f$ (where the difference is produced by either decreasing flow along an edge and increasing flow in the opposite direction) is a feasible flow in the residual network which has a positive value. This is easy to see because if $f^*_{ij} > f_{ij}$ then $(i,j)$ appears in $E_f$ and $f^*_{ij} - f_{ij} \leq u_{ij} - f_{ij}$ which is the capacity of this edge in $E_f$. If $f^*_{ij} < f_{ij}$ then $f_{ij} > 0$ and therefore the opposite edge $(j,i)$ appears in $E_f$. Also, $f_{ij} - f^*_{ij} \leq f_{ij}$ which is the capacity of $(j,i)$ in $E_f$.

When a non-zero flow exists in $E_f$, there exists a path from $s$ to $t$ using only edges in $E_f$. The shortest of these paths would appear in $E_f^l$ as well, which is a contradiction.

(d) **For a flow $f$, define**

$$d(f) = l_f(t)$$

**(the distance from $s$ to $t$ in the residual network). Prove that if $g$ is a saturating flow for $f$ then**

$$d(f+g) > d(f),$$

**where $f+g$ denotes the flow obtained from $f$ by either increasing the flow $f_{ij}$ by $g_{ij}$ or decreasing the flow $f_{ji}$ by $g_{ij}$ for every edge $(i,j) \in G_f$.**

Consider $E_f$ and the labeling of vertices $l_f(v)$. For every edge $(i,j)$ of $E_f$ we have that $l_f(j) \leq l_f(i) + 1$. Since $g$ is a saturating flow in $E_f^l$, the only edges $(u,v)$ which are in $E_{f+g}$ and not in $E_f$ are such that $(v,u) \in E_f^l$, which implies that $l_f(v) = l_f(u) - 1$. In summary, every edge $(i,j)$ of $E_{f+g}$ satisfies $l_f(j) \leq l_f(i) + 1$ and, furthermore, the edges which are not in $E_f$ actually satisfy the inequality strictly $l_f(j) < l_f(i) + 1$. Consider now any path $P$ in $E_{f+g}$. Adding up $l_f(j) \leq l_f(i) + 1$ over the edges of $P$, we get that $d(f) \leq |P|$. Moreover, we can have $d(f) = |P|$ only if all edges of $P$ also belong to $E_f$, which is impossible since $g$ is a saturating flow. Hence, $d(f) < |P|$ and this is true for any path $P$ of $E_{f+g}$ implying that $d(f) < d(f+g)$.

(e) **Prove that if $f$ is a feasible $0-1$ flow with distance $d = d(f)$ and $f^*$ is an optimum flow, then**

$$\textbf{value}(f^*) \leq \textbf{value}(f) + \frac{m}{d}$$

**and also**

$$\textbf{value}(f^*) \leq \textbf{value}(f) + \frac{n^2}{d^2}.$$

Suppose $f$ has distance $d$ and $f^*$ is an optimal flow. As noted before, $g = f^* - f$ is a feasible flow in the residual network $E_f$.

Consider $s$-$t$ cuts $C_1, C_2, \ldots C_d$ defined by

$$C_k = \{(i,j) \in E_f : l_f(i) \leq k, l_f(j) > k\}.$$

There are at most $m$ edges in total and these cuts are disjoint, therefore

$$\exists k; |C_k| \leq \frac{m}{d}.$$

Since the value of $g$ cannot be greater than any $s$-$t$ cut in $E_f$,

$$\text{value}(f^*) - \text{value}(f) = \text{value}(g) \leq \frac{m}{d}.$$

Similarly, define $d+1$ sets of vertices $V_0, V_1, V_2, \ldots, V_d$:

$$V_k = \{i \in V : l_f(i) = k\}.$$

By double counting,

$$\exists k, 1 \leq k \leq n; |V_{k-1} \cup V_k| \leq \frac{2n}{d}.$$

Suppose that $|V_{k-1}| = a$, $|V_k| \leq \frac{2n}{d} - a$. Note that the edges of $C_k$ belong to $V_{k-1} \times V_k$. Therefore

$$\text{value}(f^*) - \text{value}(f) = \text{value}(g) \leq |C_k| \leq a(\frac{2n}{d} - a) \leq \frac{n^2}{d^2}.$$

(f) **Design a maximum flow algorithm (for unit capacities) which proceeds by finding a saturating flow repeatedly. Try to optimize its running time. Using the observations above, you should achieve a running time bounded by $O(\min(mn^{2/3}, m^{3/2}))$.**

The algorithm starts with a zero flow $f$. Then we repeat the following:

- Find the levelled residual network $E_f^l$.
- Find a saturating flow $g$.
- Add $g$ to $f$, reset the residual network and continue.

Each iteration takes $O(m)$ time. Since $d(f)$ increases every time and it cannot reach more than $n$ (the maximum possible distance in $G$), the running time is clearly bounded by $O(mn)$. However, we can improve this. Suppose we iterate only $d$ times and our flow after $d$ iterations is $f$. We know $d(f) \geq d$, and if $f^*$ is an optimal flow,

$$\text{value}(f^*) - \text{value}(f) \leq \min\{\frac{m}{d}, \frac{n^2}{d^2}\}.$$

Because the flow increases by at least 1 in each iteration, the remaining number of iterations is bounded by $\min\{\frac{m}{d}, \frac{n^2}{d^2}\}$. We choose $d$ in order to optimize our bound. It turns out that the best choice is $d_1 = m^{1/2}$ for the bound based on $m$ and $d_2 = n^{2/3}$ for the bound based on $n$. Thus the total running time is $O(\min\{m^{3/2}, mn^{2/3}\})$.

(g) **Can we now justify that, for $0-1$ capacities, there is always an optimum flow that takes values 0 or 1 on every edge?**

Our algorithm finds a $0-1$ flow and we have a proof of optimality, therefore there is always a $0-1$ optimal flow. This justifies our reasoning in part (a).