**6.891 Approximation Algorithms** April 4, 2000

# Problem Set 3 Solutions

1. Consider the problem of scheduling $n$ jobs on $m$ identical machines, where $p_j$ is the amount of processing time required for job $j$. The goal is to minimize the maximum completion time $C_j$ over all jobs $j$. This problem is denote $P||C_{\max}$. It is not hard to show that a simple "list scheduling" algorithm, which takes the list of jobs, and schedules the next job in the list on the least loaded machine, is a 2-approximation algorithm. (See pages 21–22 from last years notes for this argument).

   Suppose we begin by sorting the list so that $p_1 \geq p_2 \geq \cdots \geq p_n$. Prove that this gives a $\frac{4}{3}$-approximation algorithm. (Extended hint: Suppose job $k$ finishes last. What can we say if $p_k \leq OPT/3$? Now suppose $p_k > OPT/3$. Argue that the performance of the algorithm on jobs $J_1, J_2, \ldots, J_k$ will be no better than that on $J_1, J_2, \ldots, J_n$. Now argue that the algorithm does well in this case when $p_j > OPT/3$ for all jobs $j$.)

   **Solution**: By the analysis of Graham's list scheduling algorithm, we know that the length of the schedule produced is no more than $\frac{1}{m}\sum_j p_j + p_k$, where $k$ is the last job to finish. We also know that $\frac{1}{m}\sum_j p_j \leq OPT$. Thus if $p_k \leq OPT/3$, we have that the length of the schedule is at most $\frac{4}{3}OPT$.

   Now suppose that $p_k > OPT/3$. Since job $k$ is the last job to finish, and jobs $j$ for $j < k$ cannot be scheduled before job $k$, the length of the schedule generated by the list scheduling algorithm on jobs $J_1, \ldots, J_k$ will be the same as the length of the schedule for jobs $J_1, \ldots, J_n$. Since the optimal schedule for jobs $J_1, \ldots, J_k$ will be no longer than that for $J_1, \ldots, J_n$, the performance of the algorithm on $J_1, \ldots, J_k$ will be no better than that on $J_1, \ldots, J_n$.

   Now since $p_k > OPT/3$, in the optimal schedule there can be at most two jobs per machine, implying that $k \leq 2m$. Say $k = 2m - h$. It is not hard to see, then, that an optimal schedule will put jobs $J_1, \ldots, J_h$ on machines by themselves, then schedule jobs $J_{h+1}$ and $J_k$ on a machine together, $J_{h+2}$ and $J_{k-1}$ on a machine together, and so on. Furthermore, this is the same schedule returned by the list scheduling algorithm, which is therefore optimal in this case.

2. As with linear programs, semidefinite programs have duals. The dual of the MAX CUT SDP we looked at is

$$\frac{1}{2}\sum_{i<j} w_{ij} + \frac{1}{4}\text{Min} \quad \sum_i \gamma_i$$

$$(D) \qquad \text{subject to:} \qquad W + diag(\gamma) \text{ symmetric, positive semidefinite,}$$

where the matrix $W$ is the symmetric matrix of the edge weights $w_{ij}$ and $diag(\gamma)$ is the matrix of zeroes with $\gamma_i$ as the $i$th entry on the diagonal. Show that the value of any feasible solution for this dual is an upper bound on the cost of any cut.

**Solution**: We first translate the SDP into a vector program:

$$\begin{aligned} \frac{1}{2}\sum_{i<j} w_{ij} + \frac{1}{4}\text{Min} \quad & \sum_i c_i \cdot c_i \\ \text{subject to:} & \\ & c_i \cdot c_j = w_{ij} \qquad \forall i,j \\ & c_i \in \Re^n. \end{aligned}$$

Now, since $c_i \cdot c_j = w_{ij}$, we can rewrite the objective function:

$$\begin{aligned} \frac{1}{2}\sum_{i<j} w_{ij} + \frac{1}{4}\sum_i c_i \cdot c_i &= \frac{1}{2}\sum_{i<j} c_i \cdot c_j + \frac{1}{4}\sum_i c_i \cdot c_i \\ &= \frac{1}{4}\sum_{i\neq j} c_i \cdot c_j + \frac{1}{4}\sum_i c_i \cdot c_i \\ &= \frac{1}{4}\sum_{i,j} c_i \cdot c_j \\ &= \left(\frac{1}{2}\sum_i c_i\right) \cdot \left(\frac{1}{2}\sum_i c_i\right). \end{aligned}$$

Suppose we have a feasible solution $c$ for the vector program, and let $S \subseteq V$ be any cut in the graph. The value of the cut is then

$$\sum_{i\in S, j\notin S \text{ or } i\notin S, j\in S} w_{ij} = \left(\sum_{i\in S} c_i\right) \cdot \left(\sum_{i\notin S} c_i\right).$$

So we now want to show that

$$\left(\frac{1}{2}\sum_i c_i\right) \cdot \left(\frac{1}{2}\sum_i c_i\right) \geq \left(\sum_{i\in S} c_i\right) \cdot \left(\sum_{i\notin S} c_i\right)$$

for any $S$. To simplify notation, we let the vector $\vec{x} = \frac{1}{2}\sum_i c_i$ and the vector $\vec{y} = \sum_{i\in S} c_i$, so that we are trying to prove that

$$\vec{x} \cdot \vec{x} \geq (2\vec{x} - \vec{y}) \cdot \vec{y}.$$

This follows since

$$\begin{aligned} \vec{x} \cdot \vec{x} - (2\vec{x} - \vec{y}) \cdot \vec{y} &= \vec{x} \cdot \vec{x} - 2\vec{x} \cdot \vec{y} + \vec{y} \cdot \vec{y} \\ &= (\vec{x} - \vec{y}) \cdot (\vec{x} - \vec{y}) \\ &\geq 0. \end{aligned}$$

3. Semidefinite programming can also be used to give improved approximation algorithms for the MAX SAT problem. First we start with the MAX 2SAT problem, in which every clause has at most two literals.

   (a) As in the case of MAX CUT, we'd like to express the MAX 2SAT problem as a "integer quadratic program" in which the only constraints are $y_i \in \{-1, 1\}$ and the objective function is quadratic in the $y_i$. Show that the MAX 2SAT problem can be expressed this way. (Hint: it may help to introduce a variable $y_0$ which indicates whether the value $-1$ or $1$ is "TRUE").

   (b) Derive a .878-approximation algorithm for the MAX 2SAT problem.

   (c) (Bonus problem) Use this .878-approximation algorithm for MAX 2SAT to derive a $(\frac{3}{4}+\epsilon)$-approximation algorithm for MAX SAT, for some $\epsilon > 0$. How large an $\epsilon$ can you get?

**Solution**:

(a) We introduce variables $y_i \in \{1, -1\}$ for each boolean variable $x_i$, and let $y_0 \in \{1, -1\}$ denote the value of "TRUE": that is, we set $x_i$ to TRUE iff $y_i = y_0$. We now construct a function $f$ on MAX 2SAT clauses in terms of the $y_i$ variables such that $f(C) = 1$ if clause $C$ is satisfied and $f(C) = 0$ if it is not. It is not hard to see that $f(x_i) = \frac{1}{2}(1 + y_i y_0)$ and $f(\bar{x}_i) = \frac{1}{2}(1 - y_i y_0)$. Also,

$$\begin{aligned} f(x_i \vee x_j) &= 1 - f(\bar{x}_i) f(\bar{x}_j) \\ &= 1 - \frac{1}{4}(1 - y_i y_0)(1 - y_j y_0) \\ &= \frac{1}{4}(1 + y_i y_0) + \frac{1}{4}(1 + y_j y_0) + \frac{1}{4}(1 - y_i y_j). \end{aligned}$$

Similarly,

$$\begin{aligned} f(\bar{x}_i \vee x_j) &= 1 - f(x_i) f(\bar{x}_j) \\ &= 1 - \frac{1}{4}(1 + y_i y_0)(1 - y_j y_0) \\ &= \frac{1}{4}(1 - y_i y_0) + \frac{1}{4}(1 + y_j y_0) + \frac{1}{4}(1 + y_i y_j). \end{aligned}$$

Cases for other length 2 clauses are similar.

We can thus formulate the MAX 2SAT problem over clauses $C_1, \ldots, C_m$ as

$$\begin{aligned} &\text{Max} \quad \sum_j w_j f(C_j) \\ &\quad \text{subject to:} \\ &\qquad y_i \in \{1, -1\} \qquad\qquad i = 0, \ldots, n. \end{aligned}$$

(b) As in the case of MAX CUT, we relax the "integer quadratic program" of part (a) into a vector program. That is, the program of part (a) is of the form

$$\begin{array}{ll} \text{Max} & \sum_{0 \le i < j \le n} [c_{ij}(1 + y_i y_j) + d_{ij}(1 - y_i y_j)] \\ & \text{subject to:} \\ & \quad y_i \in \{1, -1\} \qquad\qquad\qquad\qquad i = 0, \ldots, n, \end{array}$$

and we relax it to

$$\begin{array}{ll} \text{Max} & \sum_{0 \le i < j \le n} [c_{ij}(1 + v_i \cdot v_j) + d_{ij}(1 - v_i \cdot v_j)] \\ & \text{subject to:} \\ & \quad v_i \cdot v_i = 1 \qquad\qquad\qquad\qquad i = 0, \ldots, n \\ & \quad v_i \in \Re^{n+1} \qquad\qquad\qquad\qquad i = 0, \ldots, n. \end{array}$$

We will use the same **VectorRound** algorithm as we did for MAX CUT to obtain a solution to the original integer quadratic program from a solution to the vector program. Let $v$ denote the solution to the vector program, and let $\hat{y}$ be the solution obtained.

We know that

$$\frac{1}{\pi} \arccos(z) \ge .878 \cdot \frac{1}{2}(1 - z),$$

for $-1 \le z \le 1$. Thus it is also the case that

$$\frac{1}{\pi} \arccos(-z) \ge .878 \cdot \frac{1}{2}(1 - (-z)),$$

or

$$1 - \frac{1}{\pi} \arccos(z) \ge .878 \cdot \frac{1}{2}(1 + z),$$

for $-1 \le z \le 1$.

Thus it follows that

$$\begin{aligned} E[1 - \hat{y}_i \hat{y}_j] &= 2 \Pr[\hat{y}_i \ne \hat{y}_j] \\ &= \frac{2}{\pi} \arccos(v_i \cdot v_j) \\ &\ge 2 \cdot .878 \cdot \frac{1}{2}(1 - v_i \cdot v_j) \\ &= .878(1 - v_i \cdot v_j), \end{aligned}$$

and

$$\begin{aligned}
E[1+\hat{y}_i\hat{y}_j] &= 2\Pr[\hat{y}_i = \hat{y}_j] \\
&= 2(1-\frac{1}{\pi}\arccos(v_i\cdot v_j)) \\
&\geq 2\cdot .878\cdot\frac{1}{2}(1+v_i\cdot v_j) \\
&= .878(1+v_i\cdot v_j).
\end{aligned}$$

Then

$$\begin{aligned}
&E[\sum_{0\leq i<j\leq n}[c_{ij}(1+\hat{y}_i\hat{y}_j)+d_{ij}(1-\hat{y}_i\hat{y}_j)] \\
&\geq .878\sum_{0\leq i<j\leq n}[c_{ij}(1+v_i\cdot v_j)+d_{ij}(1-v_i\cdot v_j)] \\
&\geq .878\cdot OPT,
\end{aligned}$$

since the vector program is a relaxation of the original integer quadratic program.

(c) See attached pages from a journal article.

4. An important quantity in combinatorial optimization is called the *Lovász Theta Function*. The theta function is defined on undirected graphs $G=(V,E)$. One of its many definitions is given below as a semidefinite program:

$$\begin{aligned}
\vartheta(\bar{G}) = \quad & \text{Max} \ \sum_{i,j} b_{i,j} \\
& \text{subject to:} \\
& \qquad \sum_i b_{ii} = 1 \\
& \qquad b_{ij} = 0 && \forall (i,j)\notin E \\
& \qquad B = (b_{ij}) \succeq 0 && B \text{ symm}
\end{aligned}$$

Lovász showed that $\omega(G)\leq\vartheta(\bar{G})\leq\chi(G)$, where $\omega(G)$ is the size of the largest clique in $G$ and $\chi(G)$ is the minimum number of colors needed to color $G$.

(a) Show that $\omega(G)\leq\vartheta(\bar{G})$.

(b) The following is a small variation in the vector program we used for graph coloring:

$$\begin{aligned}
\text{Min} \quad & \alpha \\
\text{subject to:} \quad & v_i\cdot v_j = \alpha && \forall (i,j)\in E \\
& v_i\cdot v_i = 1 \\
& v_i \in \Re^n.
\end{aligned}$$

Its dual is

$$\begin{array}{lll} \text{Max} & -\sum_i u_i \cdot u_i & \\ \text{subject to:} & \sum_{i \neq j} u_i \cdot u_j \geq 1 & \\ & u_i \cdot u_j = 0 & (i,j) \notin E, i \neq j \\ & u_i \in \Re^n. & \end{array}$$

Show that the value of the dual (and thus the primal) is $1/(1 - \vartheta(\bar{G}))$.

**Solution**:

(a) We first translate the SDP for the theta function into a vector program:

$$\begin{array}{rll} \vartheta(\bar{G}) = & \text{Max} \ \sum_{i,j} b_i \cdot b_j & \\ & \text{subject to:} & \\ & \sum_i b_i \cdot b_i = 1 & \\ & b_i \cdot b_j = 0 & \forall (i,j) \notin E \\ & b_i \in \Re^n. & \end{array}$$

We renumber the vertices so that the largest clique is on vertices $\{1, 2, \ldots, l\}$ (and so $\omega(G) = l$). We set the vectors $b_i$ to be one-dimensional vectors such that $b_i = (\frac{1}{\sqrt{l}})$ for $i \leq l$ and $b_i = (0)$ otherwise. Then it is easy to see that $\sum_i b_i \cdot b_i = 1$ and $b_i \cdot b_j = 0$ for all $(i,j) \notin E$, since by the definition of a clique $(i,j) \in E$ for $i, j \leq l$. Finally, $\sum_{i,j} b_i \cdot b_j = l^2(\frac{1}{\sqrt{l}} \cdot \frac{1}{\sqrt{l}}) = l$. Thus we have a feasible solution to the vector program of value $l$, so it must be the case that $\vartheta(\bar{G}) \geq l = \omega(G)$.

(b) Let $b_i$ be an optimal solution to the vector program in part (a). We do not consider the vacuous case that $\vartheta(\bar{G}) = 1$, so we can assume that $\vartheta(\bar{G}) > 1$. Then let $u_i = \frac{b_i}{\sqrt{\vartheta(\bar{G}) - 1}}$ for all $i$. Note that since $b_i \cdot b_j = 0$ for $(i,j) \notin E$, it follows that $u_i \cdot u_j = 0$ for $(i,j) \notin E$. Also,

$$\begin{aligned} \sum_{i \neq j} u_i \cdot u_j &= \frac{1}{\vartheta(\bar{G}) - 1} \sum_{i \neq j} b_i \cdot b_j \\ &= \frac{1}{\vartheta(\bar{G}) - 1} (\sum_{i,j} b_i \cdot b_j - \sum_i b_i \cdot b_i) \\ &= \frac{1}{\vartheta(\bar{G}) - 1} (\vartheta(\bar{G}) - 1) \\ &= 1. \end{aligned}$$

Finally,

$$-\sum_i u_i \cdot u_i = -\frac{1}{\vartheta(\bar{G})-1}\sum_i b_i \cdot b_i = -\frac{1}{\vartheta(\bar{G})-1} = \frac{1}{1-\vartheta(\bar{G})}.$$

Since the $u_i$ are feasible for the dual, we know that the dual has value at least $1/(1-\vartheta(\bar{G}))$.

We now need to show that the dual has value $Z_D$ such that $Z_D \leq 1/(1-\vartheta(\bar{G}))$. Let $u_i$ be the vectors from an optimal solution to the dual. Let $l = \sum_{i \neq j} u_i \cdot u_j$, and note that if $l > 1$, we can create a feasible solution of greater value by setting $u_i' = u_i/\sqrt{l}$. Thus we can assume that $\sum_{i \neq j} u_i \cdot u_j = 1$. We set $b_i = \frac{1}{\sqrt{-Z_D}} u_i$, and note that this implies that $b_i \cdot b_j = 0$ when $(i,j) \notin E$ and that

$$\sum_i b_i \cdot b_i = \frac{1}{-Z_D}\sum_i u_i \cdot u_i = \frac{1}{-Z_D}(-Z_D) = 1.$$

Thus $b_i$ is a feasible solution of value

$$\begin{aligned}
\sum_{i,j} b_i \cdot b_j &= \sum_{i \neq j} b_i \cdot b_j + \sum_i b_i \cdot b_i \\
&= \frac{1}{-Z_D}\sum_{i \neq j} u_i \cdot u_j + 1 \\
&= \frac{1}{-Z_D} + 1,
\end{aligned}$$

so that $\vartheta(\bar{G}) \geq \frac{1}{-Z_D} + 1$ or $Z_D \leq 1/(1-\vartheta(\bar{G}))$. This proves that $Z_D = 1/(1-\vartheta(\bar{G}))$.

5. A metric space $(V,d)$ is a set of vertices $V$ together with a distance metric $d$ such that $d_{ij} \geq 0$ for all $i,j \in V$, $d_{ij} = d_{ji}$ for all $i,j \in V$, $d_{ij} = 0$ for $i,j \in V$ iff $i = j$, and $d_{ij} + d_{jk} \geq d_{ik}$ for all $i,j,k \in V$. A metric space is said to be $\ell_p$-embeddable if, for some $m$, there exists points $u_i \in \Re^m$ for $i \in V$ such that

$$d_{ij} = \|u_i - u_j\|_p,$$

and a metric space $(V,d)$ is $\ell_p$-embeddable with distortion $c$ if for some $m$ there exists points $u_i \in \Re^m$ for $i \in V$ such that

$$d_{ij} \leq \|u_i - u_j\|_p \leq c \cdot d_{ij}.$$

Show that given a metric space $(V,d)$, semidefinite programming can be used to find an $\ell_2$-embedding with the minimum possible distortion. (Recall that for $x \in \Re^m$, $\|x\|_p = \sqrt[p]{x_1^p + x_2^p + \cdots + x_m^p}$).

**Solution:** To do this, we simply observe that $\|u_i - u_j\|_2^2 = (u_i - u_j) \cdot (u_i - u_j) = u_i \cdot u_i + u_j \cdot u_j - 2u_i \cdot u_j$. Thus we can set up the following vector program:

$$
\begin{aligned}
\text{Min} \quad & c \\
\text{subject to:} \quad & \\
& d_{ij}^2 \le u_i \cdot u_i + u_j \cdot u_j - 2u_i \cdot u_j \le c d_{ij}^2 && \forall i, j \\
& u_i \in \Re^n && \forall i.
\end{aligned}
$$

Although $c$ in the vector program will give the value of the distortion squared, the program will still find an embedding with minimum distortion.