**18.415/6.854 Advanced Algorithms** December 12, 2001

# Problem Set Solution 6

*Lecturer: Michel X. Goemans*

1. **The *betweenness problem* is defined as follows: We are given $n$ and a set $T$ of $m$ triples of the elements of $\{1, \ldots, n\}$. We say that an ordering $\pi$ of $\{1, \ldots, n\}$ satisfies a triple $(i, j, k)$, if $j$ is between $i$ and $k$ in $\pi$. (For example, the ordering $(5, 3, 1, 2, 4)$ satisfies the triples $(5, 1, 2)$ and $(1, 3, 5)$, but not $(3, 2, 1)$). The question is to find an ordering of $\{1, \ldots, n\}$ that satisfies the maximum number of triples in $T$.**

   **This problem is known to be NP-hard, even if we restrict to instances for which an ordering that satisfies all the triples exist.**

   (a) **Use randomization to find a simple $\frac{1}{3}$-approximation algorithm for this problem. Prove the correctness of your algorithm.**

   Let $\pi$ be an ordering of $\{1, \ldots, n\}$ chosen from the set of all such orderings uniformly at random. For every fixed triple $(i, j, k)$ in $T$, the ordering $\pi$ induces a random ordering on the elements $i, j, k$. Therefore the probability that $\pi$ satisfies this triple is the same as the probability that the induced ordering is one of $(i, j, k)$ and $(k, j, i)$. Thus, $\pi$ satisfies any fixed triple in $T$ with probability $1/3$. Therefore, by the linearity of expectation, a random ordering satisfies $\frac{1}{3}|T|$ triples. Since $|T|$ is an upper bound on the number of triples that can be satisfied, the algorithm that outputs a random ordering of $\{1, \ldots, n\}$ is a $\frac{1}{3}$-approximation.

   (b) **Use the method of conditional expectations to derandomize your algorithm.**

   We want to find an ordering $\pi_1, \pi_2, \ldots, \pi_n$ that satisfies at least one third of the triples. The idea is to pick a value for $\pi_i$ that maximizes the conditional expectation of the number of satisfied triples assuming the choices that we have already made for $\pi_1, \ldots, \pi_{i-1}$ (The expectation is over the random choice of the rest of the ordering). Here's the sketch of the algorithm:

```
for i := 1 to n do
    max := 0;
    for j := 1 to n do
        if j ∉ {π_1, π_2, ..., π_{i-1}} then
            E := Exp( number of satisfied triples |
                      The first i elements of the ordering are π_1, π_2, ..., π_{i-1}, j );
            if E > max then
                max := E;
                π_i := j;
    endfor
endfor
```

   To compute the conditional expectations of the number of satisfied triples assuming that the first $i$ elements of the ordering are $\pi_1, \pi_2, \ldots, \pi_{i-1}, j$, we use the following algorithm: We divide the triples $(q, r, s)$ of $T$ into three categories:

- $|\{q,r,s\} \cap \{\pi_1, \pi_2, \ldots, \pi_{i-1}, j\}| \geq 2$:
  In this case the probability that the triple is satisfied is either 0 or 1, because the status of the triple is completely determined and does not depend on future choices. Let $m_1$ be the number of triples in this category which are satisfied.
- $|\{q,r,s\} \cap \{\pi_1, \pi_2, \ldots, \pi_{i-1}, j\}| = 1$:
  In this case, if $\{q,r,s\} \cap \{\pi_1, \pi_2, \ldots, \pi_{i-1}, j\} = \{r\}$, the probability that the triple is satisfied is 0, otherwise, this probability is $\frac{1}{2}$. Let $m_2$ be the number of triples $(q,r,s)$ in this category for which $\{q,r,s\} \cap \{\pi_1, \pi_2, \ldots, \pi_{i-1}, j\} \neq \{r\}$.
- $|\{q,r,s\} \cap \{\pi_1, \pi_2, \ldots, \pi_{i-1}, j\}| = 0$:
  For every triple in this category, the probability that it is satisfied is exactly $\frac{1}{3}$. Let $m_3$ be the number of triples in this category.

By the linearity of expectation, the conditional expected value of the number of satisfied triples is exactly $m_1 + \frac{m_2}{2} + \frac{m_3}{3}$. Furthermore, for a given sequence $\pi_1, \pi_2, \ldots, \pi_{i-1}, j$, one can easily compute the values of $m_1, m_2, m_3$. Therefore, the above algorithm can be implemented efficiently.

(c) **Assume there is an ordering that satisfies all the triples in $T$. Prove that there are vectors $v_1, \ldots, v_n \in \mathbb{R}^n$ such that**

$$\begin{array}{rcll} ||v_i - v_j|| & \geq & 1 & \textbf{for all } i \neq j, \\ (v_i - v_j)(v_k - v_j) & \leq & 0 & \textbf{for all } (i,j,k) \in T \end{array} \tag{1}$$

Consider an ordering $\pi_1, \pi_2, \ldots, \pi_n$ that satisfies all the triples. Therefore if $\sigma_i$ denotes the position of $i$ in this ordering, then for every triple $(i,j,k) \in T$, $(\sigma_i - \sigma_j)(\sigma_k - \sigma_j) < 0$. Let the vector $v_i$ be the vector that has $\sigma_i$ in its first coordinate and 0 elsewhere. Therefore, $(v_i - v_j)(v_k - v_j) = (\sigma_i - \sigma_j)(\sigma_k - \sigma_j) < 0$. Also, for every $i \neq j$, $||v_i - v_j|| = |\sigma_i - \sigma_j| \geq 1$. Therefore, $v_i$'s constitute a feasible solution for the program (1).

**Show how we can find such $v_1, \ldots, v_n$ using semidefinite programming.**

Let $Y$ be an $n \times n$ matrix defined by

$$y_{ij} := v_i . v_j. \tag{2}$$

We know that such a matrix is positive semidefinite. Conversely, for every positive semidefinite matrix $Y$, we know how to find $v_i$'s satisfying (2). The constraints $||v_i - v_j|| \geq 1$ and $(v_i - v_j)(v_k - v_j) \leq 0$ can be written in terms of $Y$ as $y_{ii} + y_{jj} - 2y_{ij} \geq 1$ and $y_{ik} + y_{jj} - y_{ij} - y_{jk} \leq 0$. Therefore, program (1) is equivalent to the following semidefinite program. (Here we only need a feasible solution, so we can take an arbitrary function as the objective function).

$$\begin{array}{rcll} y_{ii} + y_{jj} - 2y_{ij} & \geq & 1 & \text{for all } i \neq j, \\ y_{ik} + y_{jj} - y_{ij} - y_{jk} & \leq & 0 & \text{for all } (i,j,k) \in T \\ Y & \succeq & 0 & \end{array} \tag{3}$$

(d) **Give an example where the program (1) is satisfiable, but there is no ordering that satisfies all the triples in $T$.**

Let $n = 4$ and $T = \{(1,2,3),(2,3,4),(3,4,1)\}$. Assume there is an ordering of $\{1,2,3,4\}$ satisfying the triples in $T$. We may assume, without loss of generality, that 1 comes before 2 in this ordering. Therefore, since the triple $(1,2,3)$ is satisfied, 3 must come after 2, and since the triple $(2,3,4)$ is satisfied, 4 must come after 3. Therefore, the ordering does not satisfy the triple $(3,4,1)$. This shows that the above instance is not satisfiable.

Now, let $v_i$'s be defined as follows:

$$v_1 := \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad v_2 := \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad v_3 := \begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad v_4 := \begin{bmatrix} 0 \\ -1 \\ 0 \\ 0 \end{bmatrix} \tag{4}$$

It is easy to verify that $v_1, v_2, v_3, v_4$ is a feasible solution to the program (1).

(e) **Assume that $v_1, \ldots, v_n \in \mathbb{R}^n$ is a solution of the program (1). Choose $r$ uniformly at random from $\{p \in \mathbb{R}^n : ||p|| = 1\}$, and consider the ordering obtained by sorting the elements of $\{1, \ldots, n\}$ with respect to their $r^T v_i$ value. Show that in expectation this ordering satisfies at least half the constraints in $T$.**

We prove that in the ordering that is obtained by sorting $r^T v_i$'s, the probability that any fixed triple in $T$ is satisfied is at least $1/2$. This, by the linearity of expectation implies that the expected number of satisfied triples is at least $\frac{1}{2}T$. Therefore, what we need to prove is that for every triple $(i,j,k) \in T$, if we pick $r$ at random, then with probability at least $1/2$, we either have $r.v_i < r.v_j < r.v_k$ or $r.v_k < r.v_j < r.v_i$. In other words, we need to prove that with probability at least $1/2$, $r.v_i - r.v_j = r.(v_i - v_j)$ and $r.v_k - r.v_j = r.(v_k - v_j)$ have different signs. Let $x := v_i - v_j$ and $y := v_k - v_j$. In other words, we would like to compute the probability that the hyperplane normal to $r$ separates $x$ from $y$. In class, we have seen that this probability is equal to the angle between $x$ and $y$ divided by $2\pi$. Since this angle is at least $\pi/2$ because of the program (1), we are done.

2. **Consider the following scheduling problem. We are given $n$ jobs that are all available at time 0 and that can be processed on any of $m$ machines. Each job has a processing time $p_j$ which represents the amount of time a machine (any one of them) needs to process it (without interruption). A machine can only process one job at a time. This scheduling problem is to assign each job to a machine and schedule the jobs so as to minimize $\sum_j p_j C_j$ where $C_j$ represents the time at which the processing of job $j$ completes. (For example, if we have 5 jobs of unit processing time and 3 machines, there are many ways of obtaining an objective function value of $1+1+1+2+2=7$.)**

   (a) **Show that the problem is equivalent to minimizing $\sum_{i=1}^{m} M_i^2$ where $M_i$ is the total amount of processing time assigned to machine $i$.**

   Consider a solution $SOL$, and let $a_{i1}, a_{i2}, \ldots, a_{il_i}$ be the list of jobs that are scheduled on the $i$'th machine in this solution. We have

$$C_{a_{ij}} = \sum_{k=1}^{j} p_{a_{ik}}.$$

Therefore,

$$
\begin{aligned}
\sum_j p_j C_j &= \sum_{i=1}^{m} \sum_{j=1}^{l_i} p_{a_{ij}} C_{a_{ij}} \\
&= \sum_{i=1}^{m} \sum_{j=1}^{l_i} \sum_{k=1}^{j} p_{a_{ij}} p_{a_{ik}} \\
&= \frac{1}{2} \sum_{i=1}^{m} \left( \left( \sum_{j=1}^{l_i} p_{a_{ij}} \right)^2 + \sum_{j=1}^{l_i} p_{a_{ij}}^2 \right) \\
&= \frac{1}{2} \sum_{i=1}^{m} \left( M_i^2 + \sum_{j=1}^{l_i} p_{a_{ij}}^2 \right) \\
&= \frac{1}{2} \sum_{i=1}^{m} M_i^2 + \frac{1}{2} \sum_j p_j^2
\end{aligned}
$$

Therefore, since $\frac{1}{2}\sum_j p_j^2$ does not depend on $SOL$, minimizing $\sum_j p_j C_j$ is equivalent to minimizing $\sum_{i=1}^m M_i^2$.

(b) **Let $L = \frac{1}{m}\sum_j p_j$ be the average load of any machine. Show that any optimum solution for $\sum_{i=1}^n M_i^2$ will be such that each machine $i$ either satisfy $M_i \leq 2L$ or processes a single job $j$ with $p_j > 2L$.**

Consider an optimum solution $SOL$ and assume there is a machine $i$ with $M_i > 2L$ that processes more than one job. Let $j$ be the shortest job running on this machine. By the definition of $L$, there is a machine $k$ with $M_k \leq L$. Now, consider the solution $SOL'$ that is obtained from $SOL$ by running job $j$ on the machine $k$ instead of the machine $i$. If $M_i'$ denotes the total amount of processing time assigned to machine $i$ in $SOL'$, we have

$$M_i' = M_i - p_j, \qquad M_k' = M_k + p_j, \qquad M_\ell' = M_\ell \ \ \forall \ell \neq i, k.$$

Therefore,

$$
\begin{aligned}
\sum M_\ell'^2 &= \sum M_\ell^2 + (M_i - p_j)^2 - M_i^2 + (M_k + p_j)^2 - M_k^2 \\
&= \sum M_\ell^2 + 2p_j(p_j - M_i + M_k) \qquad (5)
\end{aligned}
$$

But since $j$ is the shortest job on machine $i$, we have $p_j \leq M_i/2$, and therefore, $p_j - M_i + M_k \leq -M_i/2 + M_k < -2L/2 + L = 0$. Thus, $\sum M_\ell'^2$ is smaller than $\sum M_\ell^2$, which is a contradiction with the assumption that $SOL$ is optimal.

(c) **Assume that $p_j \geq \alpha L$ for some constant $\alpha > 0$ for every job $j$, and assume that all $p_j$'s can only take $k$ different values, where $k$ is a fixed constant. Design a polynomial-time algorithm for this case.**

We use dynamic programming to solve this problem. Let $f_m(n_1, n_2, \ldots, n_k)$ denote the minimum value of $\sum M_i^2$ for scheduling $n_1$ jobs with processing time $p_1$, $n_2$ jobs with processing time $p_2$, …, and $n_k$ jobs with processing time $p_k$ on $m$ machines. The number of such subproblems is at most $mn^k$, which is a polynomial in $n$ and $m$. Now we only need to find a recurrence for computing the values of $f_m(n_1, n_2, \ldots, n_k)$.

Since $p_j$'s are at least $\alpha L$ and each machine $i$ either processes only one job, or processes more than one job with total processing time at most $2L$, therefore in any optimal solution, the number of jobs on each machine is at most $2/\alpha$. Assume that in an optimal solution machine $m$ processes $r_j$ jobs of processing time $p_j$, for $j = 1, \ldots, k$. By the above argument, $\sum_{j=1}^{k} r_j \leq 2/\alpha$. Also, by the definition of $f$, the value of the solution is $(\sum_{j=1}^{k} r_j p_j)^2 + f_{m-1}(n_1 - r_1, n_2 - r_2, \ldots, n_k - r_k)$. On the other hand, for every sequence $\vec{r} \in R = \{(r_1, \ldots, r_k) : \sum_{i=1}^{k} r_i \leq 2/\alpha\}$ with $r_i \leq n_i$ for every $i$, there is a solution of cost $(\sum_{j=1}^{k} r_j p_j)^2 + f_{m-1}(\vec{n} - \vec{r})$. Thus,

$$f_m(\vec{n}) = \min_{\vec{r} \in R, r_i \leq n_i} \left( (\sum_{j=1}^{k} r_j p_j)^2 + f_{m-1}(\vec{n} - \vec{r}) \right).$$

The size of $R$ is at most $(2/\alpha)^k$, which is a constant. Therefore, we can use the above recurrence to compute $f_m(\vec{n})$ in constant time given the values of $f_{m-1}(\vec{n} - \vec{r})$. For the base case, it is clear that $f_1(\vec{n}) = (\sum_j n_j p_j)^2$. Therefore, we can use dynamic programming to solve the problem in polynomial time.

(d) **Assume that $p_j \geq \alpha L$ for some constant $\alpha > 0$ for every job $j$. Design a polynomial-time approximation scheme for this case.**

Let $I$ denote the instance of the problem that is given as the input. First, for every job $j$ with a processing time greater than $2L$, we assign a machine to process this job (and no other job). Then we solve the problem recursively for the set of remaining jobs and remaining machines. By part (b), we know that assigning a job with processing time more than $2L$ to a machine that only processes this job does not increase the value of the optimum. Therefore, we are not losing any approximation factor here.

Now, we know that for every $j$, $\alpha L \leq p_j \leq 2L$. We define $p'_j$ as follows: $p'_j := \min\{(1+\varepsilon')^h : (1+\varepsilon')^h \geq p_j\}$, where $\varepsilon'$ is such that $(1+\varepsilon')^2 \leq (1+\varepsilon)$ and, say, greater than $1+\varepsilon/2$. In other words, $p'_j$ is the smallest power of $(1+\varepsilon')$ greater than $p_j$. Let $I'$ denote the instance of the problem with $p'_j$'s instead of $p_j$'s. It is clear form the definition that $p_j \leq p'_j \leq (1+\varepsilon')p_j$. Also, since all $p_j$'s are between $\alpha L$ and $2L$, $p'_j$'s can take at most $k := \log_{(1+\varepsilon')}(2/\alpha) + 1 = O(1)$ values. Therefore, using the algorithm in part (c), we can find the optimal solution $SOL'$ for $I'$ in polynomial time.

Now consider an optimal solution $SOL$ of cost $OPT$ for $I$, and evaluate it as a solution to $I'$. Since for each $j$, the new value of $C_j$ with respect to $p'_j$'s is at most $(1+\varepsilon')$ times its value with respect to $p_j$'s, therefore the cost of $SOL$ with respect to $p'_j$'s is at most $(1+\varepsilon')^2 OPT \leq (1+\varepsilon)OPT$. This shows that there is a solution of cost at most $(1+\varepsilon)OPT$ for $I'$. Therefore, the cost of $SOL$ with respect to $p'_j$'s is at most $(1+\varepsilon)OPT$. However, since $p'_j \geq p_j$ for every $j$, the cost of $SOL$ with respect to $p_j$'s is upper bounded by its cost with respect to $p'_j$'s, which is at most $(1+\varepsilon)OPT$.