**6.891 Approximation Algorithms** March 10, 2000

# Problem Set 1 Solutions

1. The following problem arises in telecommunications networks, and is known as the *SONET ring loading problem.* The network consists of a cycle on $n$ nodes, numbered 0 through $n-1$ clockwise around the cycle. Some set $C$ of calls is given; each call is a pair $(i,j)$ originating at node $i$ and destined to node $j$. The call can be routed either clockwise or counterclockwise around the ring. The objective is to route the calls so as to minimize the total *load* on the network. The load $L_i$ on link $(i, i+1 (\text{mod } n))$ is the number of calls routed through link $(i, i+1(\text{mod } n))$, and the total load is $\max_{1 \le i \le n} L_i$.

   Give a 2-approximation algorithm for the SONET ring loading problem.

   **Solution**: We first formulate the problem as an integer programming problem. For the $k$th call, we create two 0-1 variables $x_0^k$ and $x_1^k$, corresponding to the two possibilities of how to route the call (0 for clockwise and 1 for counterclockwise). Obviously we must have $x_0^k + x_1^k = 1$. Note that for any link $(i, i+1(\text{mod } n))$, one of the two routings of call $k$ will use this link; let $b(k,i) = 0$ if the clockwise routing uses this link and let $b(k,i) = 1$ if the counterclockwise routing uses this link. Then we can formulate the problem as follows:

$$
\begin{array}{rll}
\text{Min} & L & \\
\text{subject to:} & x_0^k + x_1^k = 1 & \forall k \in C \\
& \displaystyle\sum_{k \in C} x_{b(k,i)}^k \le L & i = 0, \ldots, n-1 \\
& x_0^k \in \{0,1\} & \forall k \in C \\
& x_1^k \in \{0,1\} & \forall k \in C \\
& L \ge 0. &
\end{array}
$$

   We relax the integer program to a linear program by changing to the integrality constraints to $x_0^k \ge 0$ and $x_1^k \ge 0$. Now consider the following algorithm:

**SONET**

Solve LP; get opt. soln. $\bar{x}$, $\bar{L}$
For $k \in C$
  If $\bar{x}_0^k \geq \frac{1}{2}$
    $\tilde{x}_0^k \leftarrow 1$
    $\tilde{x}_1^k \leftarrow 0$
  else
    $\tilde{x}_0^k \leftarrow 0$
    $\tilde{x}_1^k \leftarrow 1$
Return $\tilde{x}$

Now for any $i \in \{0, \ldots, n-1\}$,

$$\sum_{k \in C} \tilde{x}^k_{b(k,i)} \leq 2 \sum_{k \in C} \bar{x}^k_{b(k,i)} \leq 2\bar{L} \leq 2OPT,$$

so that the algorithm is a 2-approximation algorithm.

2. Consider the weighted vertex cover problem.

(a) Prove that any extreme point of the linear program

$$\text{Min} \quad \sum_{i \in V} w_i x_i$$

subject to:

$$\begin{aligned} x_i + x_j &\geq 1 & \forall (i,j) \in E \\ x_i &\geq 0 & i \in V \end{aligned}$$

has the property that $x_i \in \{0, \frac{1}{2}, 1\}$ for all $i$ (Recall that an extreme point $x$ is a feasible solution that cannot be expressed as $\lambda x^1 + (1-\lambda)x^2$ for $0 < \lambda < 1$ and feasible solutions $x^1$ and $x^2$ distinct from $x$.)

(b) Give a $\frac{3}{2}$-approximation algorithm for the weighted vertex cover problem when the input graph is planar. You may use the fact that polynomial-time LP solvers return extreme points, and that there is a polynomial-time algorithm to 4-color any planar graph (i.e. the algorithm assigns each vertex one of four colors such that for any edge $(i,j) \in E$, vertices $i$ and $j$ have been assigned different colors).

**Solution:**

(a) To prove this, for any $x$ such that there exists $x_i \notin \{0, \frac{1}{2}, 1\}$, we give a $\lambda$ and feasible solutions $x^1, x^2$ such that $x = \lambda x^1 + (1-\lambda)x^2$. Let

$I = \{i : x_i \notin \{0, \frac{1}{2}, 1\}$, and let $\epsilon = \min_{i \in I} \min(x_i, |\frac{1}{2} - x_i|, |1 - x_i|)$. Clearly $\epsilon > 0$. Set $\lambda = \frac{1}{2}$ and

$$x_i^1 = \begin{cases} x_i & \text{if } i \notin I \\ x_i - \epsilon & \text{if } i \in I, x_i < \frac{1}{2} \\ x_i + \epsilon & \text{if } i \in I, x_i > \frac{1}{2} \end{cases}$$

and

$$x_i^2 = \begin{cases} x_i & \text{if } i \notin I \\ x_i + \epsilon & \text{if } i \in I, x_i < \frac{1}{2} \\ x_i - \epsilon & \text{if } i \in I, x_i > \frac{1}{2} \end{cases}$$

Clearly $x = \frac{1}{2}x^1 + \frac{1}{2}x^2$. Now we must show that $x^1$ and $x^2$ are feasible solutions. By the definition of $\epsilon$, $x^1 \geq 0$ and $x^2 \geq 0$. Pick any edge $(i,j) \in E$. We know that $x_i + x_j \geq 1$, and must show that $x_i^1 + x_j^1 \geq 1$ and $x_i^2 + x_j^2 \geq 1$. In the case of $x^1$, clearly there only problems if either both $i, j \in I$ and $x_i$ and $x_j$ are both less than $1/2$, or if one of $i, j \in I$ (say $i$), $x_i < 1/2$, and $x_j = 1/2$. But in both cases, this implies $x_i + x_j < 1$. In the case of $x^2$, there are only problems if both $i, j \in I$, and $x_i$ and $x_j$ are both greater than $1/2$, or if one of $i, j \in I$ (say $i$), $x_i > 1/2$, and $x_j = 0$ or $x_j = 1/2$. In the first case we have

$$x_i^2 + x_j^2 = x_i + x_j - 2\epsilon \geq x_i + x_j - (x_i - \frac{1}{2}) - (x_j - \frac{1}{2}) \geq 1,$$

by the definition of $\epsilon$. In the second case, with $x_j = 1/2$, we have

$$x_i^2 + x_j^2 = x_i - \epsilon + \frac{1}{2} \geq x_i - (x_i - \frac{1}{2}) + \frac{1}{2} \geq 1.$$

The case with $x_j = 0$ is similar.

(b) Consider the following algorithm.

Solve the linear programming relaxation to obtain a solution $x^*$ such that $x_i^* \in \{0, \frac{1}{2}, 1\}$ for all $i \in V$.
4-color the graph $G$; Let $S_k$ be the set of vertices of color $k$.
$k \leftarrow \arg\max_{k=1,2,3,4} \sum_{i \in S_k : x_i^* = 1/2} w_i$
For each $i \in V$
  If $x_i^* = 1$ or ($x_i^* = \frac{1}{2}$ and $i \notin S_k$)
    $C \leftarrow C \cup \{i\}$

**Lemma 1** *$C$ is a vertex cover.*

**Proof:** Pick any edge $(i,j) \in E$. Since $x_i^*, x_j^* \in \{0, \frac{1}{2}, 1\}$ and $x_i^* + x_j^* \geq 1$, it must be the case that either one of $x_i^*, x_j^*$ is 1, or $x_i^* = x_j^* = \frac{1}{2}$. Suppose we are in the first case, and $x_i^* = 1$. Then by construction $i \in C$. Suppose we are in the second case. Since we have a 4-coloring of the graph, it cannot be the case that both $i \in S_k$ and $j \in S_k$. So suppose $i \notin S_k$. Then by construction $i \in C$. $\Box$

**Lemma 2** *The algorithm is a $\frac{3}{2}$-approximation algorithm.*

**Proof:**

$$
\begin{aligned}
(1)\quad \sum_{i \in C} w_i &\leq \sum_{i \in V: x_i^* = 1} w_i + \sum_{i \in V: x_i^* = 1/2} w_i - \sum_{i \in S_k: x_i^* = 1/2} w_i \\
&= \sum_{i \in V: x_i^* = 1} w_i x_i^* + 2 \sum_{i \in V: x_i^* = 1/2} w_i x_i^* - 2 \sum_{i \in S_k: x_i^* = 1/2} w_i x_i^* \\
&\leq \sum_{i \in V: x_i^* = 1} w_i x_i^* + 2 \sum_{i \in V: x_i^* = 1/2} w_i x_i^* - 2(\frac{1}{4} \sum_{i \in V: x_i^* = 1/2} w_i x_i^*) \\
(2)\quad &\leq \sum_{i \in V: x_i^* = 1} w_i x_i^* + \frac{3}{2} \sum_{i \in V: x_i^* = 1/2} w_i x_i^* \\
&\leq \frac{3}{2} \sum_{i \in V} w_i x_i^* \\
&\leq \frac{3}{2} OPT.
\end{aligned}
$$

Inequality 1 follows from the construction of $C$. Inequality 2 follows since we chose $k$ to maximize $\sum_{i \in S_k: x_i^* = 1/2} w_i x_i^*$; thus it must be the case that $\sum_{i \in S_k: x_i^* = 1/2} w_i x_i^* \geq \frac{1}{4} \sum_{i \in V: x_i^* = 1/2} w_i x_i^*$. $\Box$

3. In the weighted Set Cover problem, one finds a collection $I$ of sets that minimizes $\sum_{j \in I} w_j$ such that

$$|\bigcup_{j \in I} S_j| = |T|.$$

Consider the partial cover problem, in which one finds a collection $I$ of sets that minimizes $\sum_{j \in I} w_j$ such that

$$|\bigcup_{j \in I} S_j| \geq p|T|,$$

where $0 < p < 1$ is some constant.

(a) Give a polynomial-time algorithm to find a solution to the partial cover problem in which the value is no more than $c(p) \cdot OPT$, where $c(p)$ is a constant that depends on $p$, and $OPT$ is the value of the optimal solution to the set cover problem.

(b) (Bonus problem) Give an $f(p)$-approximation algorithm for the partial cover problem, such that $f$ is nondecreasing in $p$ and $f(1) \leq H_{|T|}$.

**Solution:**

(a) First assume that no single set forms a set cover. Now consider the greedy algorithm:

**Greedy**

$I \leftarrow \emptyset$
$i \leftarrow 1$
$n_i = |T|$
$\tilde{S}_j \leftarrow S_j \;\; \forall j$
while $|\bigcup_{j \in I} S_j| < p|T|$
 $l \leftarrow \arg\min_{j:\tilde{S}_j \neq \emptyset} \frac{w_j}{|\tilde{S}_j|}$
 $I \leftarrow I \cup \{l\}$
 $\tilde{S}_j \leftarrow \tilde{S}_j - S_l \;\; \forall j.$
 $i \leftarrow i + 1$
 $n_i = |T - \bigcup_{j \in I} S_j|.$

Note that $n_i$ denotes the number of elements not covered at the beginning of the $i$th iteration. Let $c_i$ denote the cost of the set chosen in the $i$th iteration, and let $k_i$ denote the number of additional elements covered in the $i$th iteration. Let $\ell$ be the number of iterations needed before at least $p|T|$ elements are covered, so that $n_\ell > |T|(1-p)$, but $n_{\ell+1} \leq |T|(1-p)$. Since in the $i$th iteration we can cover the remaining $n_i$ uncovered elements at the price of $OPT$, the set chosen in the $i$th iteration must have a cost-to-items-covered ratio no more than this; that is,

$$\frac{c_i}{k_i} \leq \frac{OPT}{n_i}.$$

Obviously $n_{i+1} = n_i - k_i$. Then

$$n_{i+1} = n_i - k_i = n_i \left(1 - \frac{k_i}{n_i}\right) \leq n_i \left(1 - \frac{c_i}{OPT}\right).$$

Thus

$$\ln \frac{n_i}{n_{i+1}} \geq -\ln\left(1 - \frac{c_i}{OPT}\right) \geq \frac{c_i}{OPT},$$

using the fact that $-\ln(1-x) \geq x$ for $0 \leq x < 1$. Thus $c_i \leq \ln \frac{n_i}{n_{i+1}} \cdot OPT$. Since no single set forms a set cover, it is also the case that $c_i \leq OPT$.

Thus the cost of the set cover is

$$\begin{aligned}\sum_{i=1}^{\ell} c_i &\leq OPT \cdot \left(1 + \sum_{i=1}^{\ell-1} \ln \frac{n_i}{n_{i+1}}\right) \\ &\leq OPT \cdot \left(1 + \ln \frac{n_1}{n_\ell}\right) \\ &\leq OPT \cdot \left(1 + \ln \frac{|T|}{|T|(1-p)}\right) \\ &\leq OPT \cdot \left(1 + \ln \frac{1}{1-p}\right).\end{aligned}$$

(b) See the attached pages. They are from Petr Slavík, "Approximation Algorithms for Set Cover and Related Problems," Ph.D. thesis, SUNY Buffalo Dept. of Mathematics, April 1998.

4. In the Karmarkar-Karp algorithm, we used *geometric grouping* in each iteration of the algorithm to create an instance $I'$ from $I$. Consider the following grouping scheme: for $i = 0, \ldots, \lceil \log_2 SIZE(I) \rceil$, create a group $G_i$ such that all pieces from $I$ of size $(2^{-(i+1)}, 2^{-i}]$ are placed in group $G_i$. In each group $G_i$, create subgroups $G_{i,1}, G_{i,2}, \ldots, G_{i,k_i}$ by arranging the pieces in $G_i$ from largest to smallest and putting the $4 \cdot 2^i$ largest in the first subgroup, the next $4 \cdot 2^i$ in the next subgroup, and so on. Now create instance $I'$ from $I$ in the following manner: for each $i = 0, \ldots, \lceil \log_2 SIZE(I) \rceil$, discard subgroups $G_{i,1}$ and $G_{i,k_i}$ (i.e., the first and last subgroups of $G_i$) and for $j = 2, \ldots, k_i - 1$ round each piece of subgroup $G_{i,j}$ to the size of the largest piece in $G_{i,j}$.

   Prove that by using this grouping scheme within the Karmarkar-Karp algorithm, we obtain an algorithm $KK'$ such that $KK'(I) \leq OPT(I) + O(\log^2(OPT(I)))$ for all instances $I$.

   **Solution:** We only need prove that this grouping scheme has the same behavior as the grouping scheme for Karmarkar-Karp that we saw in class. Namely, we must show that for the instance $I'$ produced the following three claims hold: (1) $OPT(I') \leq OPT(I)$ and $LP(I') \leq LP(I)$; (2) the discards can be packed in $O(\log SIZE(I))$ bins; (3) the total number of piece sizes in $I'$ is no more than $SIZE(I)/2$. Claim (1) follows easily, as we can map each piece in $I'$ to a piece of size at least as large in $I$.

   Claim (2) is also not difficult: we discard $8 \cdot 2^i$ pieces from each group $G_i$, each of which is size at most $2^{-i}$. Thus the total size of the discards is at most

$$\sum_{i=0}^{\lceil \log_2 SIZE(I) \rceil} 8 \cdot 2^i \cdot 2^{-i} = 8(\lceil \log_2 SIZE(I) \rceil + 1),$$

   which we can pack in $O(\log SIZE(I))$ bins using First Fit.

Now we prove claim (3). Let $n_i$ be the number of pieces in $I$ of size $(2^{-(i+1)}, 2^{-i}]$, so that $SIZE(I) \geq \sum_{i=0}^{\lceil \log_2 SIZE(I) \rceil} n_i 2^{-(i+1)}$. Let $s_i$ be the number of piece sizes in $I'$ in the range $(2^{-(i+1)}, 2^{-i}]$, so that the total number of piece sizes in $I'$ is $\sum_{i=0}^{\lceil \log_2 SIZE(I) \rceil} s_i$. Note that $s_i \leq \frac{n_i}{4 \cdot 2^i}$, so that the total number of piece sizes in $I'$ is no more than

$$\begin{aligned} \sum_{i=0}^{\lceil \log_2 SIZE(I) \rceil} \frac{n_i}{4 \cdot 2^i} &= \sum_{i=0}^{\lceil \log_2 SIZE(I) \rceil} \frac{n_i 2^{-(i+1)}}{2} \\ &\leq \frac{1}{2} SIZE(I). \end{aligned}$$

5. Consider the following problem:

**Directed Steiner Tree**

- **Input:**
  - A directed graph $G = (V, E)$
  - Costs $c_{ij} \geq 0$ for arcs $(i, j) \in E$
  - A root vertex $r \in V$
  - A set of terminal $T \subseteq V$
- **Goal:** Find a minimum-cost tree such that for each $i \in T$ there exists a directed path from $r$ to $i$.

Prove that for some constant $c$ there can be no $c \log |T|$-approximation algorithm for the Directed Steiner Tree problem unless $P = NP$.

(Research problem: Give a $c \log^k |T|$-approximation algorithm for the Directed Steiner Tree problem for some constants $c$ and $k$).

**Proof:** We know from class that there exists no $c \log n$-approximation algorithm for the set cover problem for some constant $c$ unless $P = NP$. We show that if we could get a $c \log |T|$-approximation algorithm for the directed Steiner tree problem, then we could get a $c \log n$-approximation algorithm for the set cover problem, and that will imply the result.

To do this, suppose we have an instance of a set cover problem with ground set $E = \{e_1, \ldots, e_n\}$, subsets $S_j \subseteq E$ $(1 \leq j \leq m)$, and weights $w_j$ on the sets. We create the following directed graph $G$: we have a root vertex $r$, vertices $s_1, \ldots, s_m$ (one for each set), and vertices $e_1, \ldots, e_n$ (one for each $e_i \in E$). We have an edge $(r, s_j)$ for each vertex $s_j$ of cost $c(r, s_j) = w_j$, and for each $s_j$, we introduce edges $(s_j, e_i)$ for each $e_i \in S_j$ of cost $c(s_j, e_i) = 0$. We let $T = \{e_1, \ldots, e_n\}$.

We now show that there exists a set cover of cost $C$ if and only if there exists a directed Steiner tree in $G$ of cost $C$. Suppose $I$ is a set cover such that

$\sum_{j \in I} w_j = C$ and $\bigcup_{j \in I} S_j = E$. Then the set of edges $E' = \{(r, s_j) : j \in I\} \cup \{(s_j, e_i) : j \in I, e_i \in S_j\}$ has cost $\sum_{j \in I} c(r, s_j) = \sum_{j \in I} w_j = C$. Furthermore, since $\bigcup_{j \in I} S_j = E$, we know that for each vertex $e_i \in T$, there exists a path $(r, s_j), (s_j, e_i) \in E'$ from $r$ to $e_i$, since $e_i \in S_j$ for some $j \in I$. Similarly, suppose we have a set of edges $E'$ which is a directed Steiner tree of cost $C$. Let $I = \{j : (r, s_j) \in E'\}$. Then certainly $\sum_{j \in I} w_j = C$. Furthermore, since $E'$ is a directed Steiner tree, there must be a path from $r$ to $e_i$ in $E'$ for each $e_i$, which must have the form $(r, s_j), (s_j, e_i)$. Then since $(s_j, e_i)$ is an edge only if $e_i \in S_j$, it follows that $e_i \in \bigcup_{j \in I} S_j$ for all $i$, and $I$ is a set cover.

Let $OPT_{DS}$ be the optimum cost of a directed Steiner tree in $G$, and let $OPT_{SC}$ be the optimum cost of the set cover problem. The argument of the preceding paragraph implies $OPT_{DS} = OPT_{SC}$. Suppose we had a $c \log |T|$-approximation algorithm for the directed Steiner tree problem. We could then produce a directed Steiner tree of cost no more than $(c \log |T|) OPT_{DS}$. But this would give us a set cover of cost no more than $(c \log |T|) OPT_{DS} = (c \log n) OPT_{SC}$. Thus we would have a $(c \log n)$-approximation algorithm for the set cover problem. $\square$