CSC 2421 – Approximation Algorithms
Lecture #6: 11 February 2003

Lecturer: A. Borodin
Scribes: L.C. Lau and R. Krueger

1 Outline

- Proof of Steiner forest approximation
- Local search
  - k-median problem
  - facility location problem

2 Steiner Forest as a hitting set problem

We recall the Steiner Forest problem. Given an undirected graph $G = (V, E)$, a cost function on edges $c : E \to \mathbb{Q}^+$, and a collection of disjoint subsets of V, $S_1, \ldots, S_k$, find a minimum cost subgraph in which each pair of vertices belonging to the same set S_i is connected.

We can restate this problem by defining a *connectivity requirement function* r mapping unordered pairs of vertices to $\{0, 1\}$ by

$$r(u,v) = \begin{cases} 1 \text{ if } u \text{ and } v \text{ belong to the same set } S_i \\ 0 \text{ otherwise} \end{cases}$$

Our problem is to find a minimum cost subgraph F that contains a u-v path for each pair (u, v) with $r(u, v) = 1$. Notice that such a subgraph will be a forest.

Recall our LP/IP formulation of the Steiner Forest problem rephrased as a hitting set problem. We define $\delta(S)$ as the set of edges crossing the cut $(S, \overline{S})$. The minimum number of edges that must cross each cut is

$$f(S) = \begin{cases} 1 \text{ if there exists } u \in S \text{ and } v \in \overline{S} \text{ with } r(u,v) = 1 \\ 0 \text{ otherwise} \end{cases}$$

The primal LP-relaxation is:

$$
\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{e \in E} c_e x_e & \\
\text{subject to} & \displaystyle\sum_{e: e \in \delta(S)} x_e \geq f(S) & \forall S \subseteq V \\
& x_e \geq 0 & \forall e \in E
\end{array}
$$

The dual is:

$$
\begin{array}{lll}
\text{maximize} & \displaystyle\sum_{S \subseteq V} f(S) \cdot y_S & \\
\text{subject to} & \displaystyle\sum_{S: e \in \delta(S)} y_S \leq c_e & \forall e \in E \\
& y_S \geq 0 & \forall S \subseteq V
\end{array}
$$

We have the notion of active sets S, which correspond to active cuts $(S, \overline{S})$. Each set S corresponds to one of the connected components we have formed so far. Our algorithm only considers adding edges leaving a connected component, so we know we will always have a forest. The algorithm may add edges which are not needed, so our pruning step will go back and prune them out of our forest.

Note that when some edge becomes tight, we throw in edges one at a time. This may result in some active sets deactivating, meaning some edge that became tight might not be added to the forest (for example, if we add all tight edges we may form a cycle, but adding edges one at a time means the last edge's set will become deactivated, preventing the cycle from forming).

Algorithm 22.3 from Vazirani (Steiner forest)

1. **(Initialization)** $F \leftarrow \emptyset$; **for each** $S \subseteq V$, $y_S \leftarrow 0$
2. **(Edge augmentation)**
   while there exists an unsatisfied set **do**:
   simultaneously raise y_S for each active set S, until some edge e goes tight;
   $F \leftarrow F \cup \{e\}$
3. **(Pruning) return** $F' = \{e \in F \mid F - \{e\} \text{ is primal infeasible}\}$

We define F' as the output of algorithm 22.3, and let F be the current forest at the end of any iteration.

Definition 2.1. $\deg_{F'}(C)$ *is the number of edges that leave* C *in forest* F'.

The following Lemmas are useful in showing that Algorithm 22.3 is a 2-approximation.

Lemma 2.2. *If* C *is a component in any iteration and* $f(C) = 0$*, then* $\deg_{F'}(C) \neq 1$*; that is,* $\deg_{F'}(C) = 0$ *or* $\deg_{F'}(C) \geq 2$.

Proof. Idea: This situation will not happen since redundant edges are pruned in step 3 of the algorithm.

Suppose $deg_{F'}(C) = 1$, and let e be the unique edge of F' crossing the cut $(C, \overline{C})$. Since e is nonredundant (every edge in F' is nonredundant), there is a pair of vertices, say u, v, such that $r(u, v) = 1$ and e lies on the unique u-v path in F'. Since this path crosses the cut $(C, \overline{C})$ exactly once, one of these vertices must lie in C and the others in $\overline{C}$. Now, since $r(u, v) = 1$, we get that $f(C) = 1$; a contradiction. $\square$

Lemma 2.3. $\sum_{e \in F'} c_e \leq 2 \sum_{S \subseteq V} y_S$.

Proof. Since every picked edge is tight,

$$\sum_{e \in F'} c_e = \sum_{e \in F'} \left(\sum_{S: e \in \delta(S)} y_S \right).$$

Changing the order of summation we get:

$$\sum_{e \in F'} c_e = \sum_{S \subseteq V} \left(\sum_{e \in \delta(S) \cap F'} y_S \right) = \sum_{S \subseteq V} deg_{F'}(S) \cdot y_S.$$

Thus, we need to show that

$$\sum_{S \subseteq V} deg_{F'}(S) \cdot y_S \leq 2 \sum_{S \subseteq V} y_S. \tag{1}$$

We will prove that in each iteration, the increase in the left-hand side of inequality (1) is bounded by the increase in the right-hand side. Consider an iteration; let Δ be the extent to which active sets were raised in this iteration. Then, we need to show:

$$\Delta \times \left(\sum_{S \text{ active}} deg_{F'}(S) \right) \leq 2\Delta \times (\# \text{ of active sets}).$$

Notice that the degree with respect to F' of any active set S is due to edges that will be picked *during or after* the current iteration. Let us rewrite this inequality as follows:

$$\frac{\sum_{S \text{ active}} deg_{F'}(S)}{\#\text{ of active sets}} \leq 2 \tag{2}$$

So, we need to show that in this iteration, the average degree of active sets with respect to F' is at most 2. The argument is based on the fact that the average degree of a forest is at most 2.

Let H be a graph on vertex set and edge set F'. Consider the set of connected components with respect to F at the beginning of the current iteration. In H, contract the set of vertices of each of these components to a single node to obtain graph H' (for clarity, we call the vertices of H' nodes). Notice that in going from H to H', all edges picked in F before the current iteration have been contracted. Clearly, the degree of a node in H' is equal to the degree of the corresponding set in H. We say that any node of H' corresponding to an active component is an *active node*; otherwise *inactive*. Each active node of H' has nonzero degree (since there must be an edge incident to it to satisfy the requirement), and H' is a forest. Now, remove all isolated nodes from H'. The remaining graph is a forest with average degree at most 2. By Lemma 2.2 the degree of each inactive node in this graph is at least 2. Hence, the average degree of active nodes is at most 2. □

Theorem 2.4. *Algorithm 22.3 is a 2-approximation.*

Proof. We claim that at the end of this algorithm, F' and y are primal and dual feasible solutions. It is clear that there is no cycle created; so it is a forest and thus primal feasible.

When an edge goes tight, the current iteration ends and active sets are redefined. Therefore, no edge is overtightened. Hence, y is dual feasible.

Notice that in our algorithm, if $f(S) = 0$, then $y_S = 0$. So the right hand side of the inequality in Lemma 2.3 is equal to the optimal value of the dual objective function (and thus the optimal value of the primal objective function) and the left hand side of the inequality in Lemma 2.3 is the cost of the Steiner forest constructed by the algorithm. So, by Lemma 2.3, the theorem follows. □

The tight example given for the metric Steiner tree problem is also a tight example for this algorithm. Furthermore, a lower bound of 2 on the integrality gap of this LP-relaxation can be shown (see page 207 of [1]) even if restricted to the minimum spanning tree problem.

3 Local Search

We now turn our attention to a different approximation technique, local search. This general technique is the underlying idea for many well known methods such as hill climbing, steepest descent, simulated annealing, and many others, and is traditionally known as a heuristic rather than as a worst case approximation technique.

3.1 The Basic Local Search Algorithm

Let Π be an optimization problem and let I be an instance. Following the notation from our first lecture, we call the solution space $S_\Pi(I)$, and for any $s \in S_\Pi(I)$, we say $c_\Pi(s, I)$ is the cost of solution s to instance I.

We define a function $n_\Pi(s, I)$ as the neighborhood of s (which may contain other solutions to I). For example, if solutions are just 0-1 vectors, the neighborhood could be defined by Hamming distance.

Basic Local Search Algorithm

1. Choose some naïve feasible solution $s = s_0$.
2. Search in $n_\Pi(s, I)$ for an $s' \in S_\Pi(I)$ with

$$c_\Pi(s', I) < c_\Pi(s, I) \quad \text{(for minimization problem)}$$
$$c_\Pi(s', I) > c_\Pi(s, I) \quad \text{(for maximization problem)}$$

   If such an s' exists, set $s \leftarrow s'$ and repeat step 2.

3. Terminate and output "local optimum" s.

Definition 3.1. *The locality gap for a given neighborhood function n_Π is* $\max_I \frac{\textit{local-OPT}(I)}{\textit{OPT}(I)}$.

In local search, we choose a neighborhood function with respect to which it is efficient to search for a better solution. Usually the neighborhood of any solution is polynomial size but this is not necessary. Then there are two big issues to resolve for obtaining a good approximation in polynomial time:

1. number of iterations before hitting the local optimum is polynomial
2. locality gap is small

There are many variations of the basic local search paradigm.

1′. Choose a "random" initial feasible solution.

1″. Use some efficient method to find a "good" initial solution. Note that we do not expect to prove lower bounds for such algorithms since we can not rule out computing an optimal solution as the initial solution.

2′. Find a best $s' \in n_{\Pi}(s, I)$.

2″. Find an s' such that $c'(s', I) \leq c'(s, I)$ for some $c' \neq c_{\Pi}$.

When we use the given objective function for searching the local neighborhood, Khanna *et al* [2] call this an "oblivious" local search but I prefer calling this a greedy local search. Consider the following greedy local search strategy for MAX 2-SAT:

Greedy Local Search Algorithm for MAX 2-SAT

1. $S \leftarrow$ an arbitrary feasible solution.
2. **while** there exists a swap operation op that flips the truth of some variables with

$$\text{cost}(\text{op}(S)) \leq \text{cost}(S),$$

where $\text{cost}(S)$ is evaluated as the number of clauses in S being satisfied,

do $S \leftarrow \text{op}(S)$.

3. **return** S.

In [2] they show that if we use this algorithm, even if we consider a swap of Hamming distance δ (i.e. for any $\delta \in o(n)$), the locality ratio cannot be better than $3/2$. Furthermore, they show that this ratio is still bounded by $5/4$ when δ may take any value less than $n/2$ (notice that the complexity could be exponential). On the other hand, if we use a cost function

$$\text{cost}(S) = \frac{3}{2}W(S_1) + 2W(S_2)$$

where $W(S_1)$ is the number of clauses with exactly 1 true literal and $W(S_2)$ is the number of clauses with exactly 2 true literals, then using the above algorithm with Hamming distance 1 ensures a performance ratio of $4/3$. Notice that this cost function may rule out an optimal solution (an assignment with all clauses being satisfied may not be a local optimal solution), but such a problem is not very important from an approximation point of view. On the other hand, in order to prove a lower bound on local search algorithms, we will need to restrict the class of allowable cost functions.

$2'''$. Quit when we can't find an s' such that $c_\Pi(s', I) \leq \left(1 - \frac{1}{p(n)}\right) \cdot c(s, I)$, where $p(n)$ is some polynomial.
But we might reach near-flat neighborhoods and get stuck at a solution whose cost/profit is far from that of a local optimum, an optimum that we may reach if we take one tiny improvement step.

3.2 Applying Local Search

Recall the facility location problem. We are given:

- a set of m facilities $\mathcal{F}$,
- a set of n cities $\mathcal{C}$,
- facility opening cost f_i for each facility $i = 1, \ldots, m$, and
- connection cost c_{ij} for connecting city i to facility j.

Two related problems are thus:

- **k-facility location problem** — you are only allowed to open at most k facilities.
- **k-median problem** — k-facility location problem when $f_i = 0$ for all facilities i. Clearly we only need to consider solutions where exactly k facilities are opened. We note that this problem is basically a clustering problem.

The *disjoint* case, when facilities and cities are distinct, is the traditional characterization for facility location problems.

The *complete* case, when $\mathcal{F} = \mathcal{C}$, is traditional for the k-median problem.

We note that approximation bounds obtained for the facility location problem are not necessarily meaningful for the k-median problem, and vice versa. But algorithms and approaches useful for one can frequently be modified to fit the other.

3.3 The Metric k-median Problem

Consider the metric case of the disjoint k-median problem (i.e. the connection costs respect the triangle inequality and $\mathcal{F} \cap \mathcal{C} = \emptyset$). As noted before, an α-approximation in the disjoint model implies an α-approximation in the complete model.

A solution to the k-median problem is a subset $S \subseteq \mathcal{F}$ with $|S| = k$. If we have n facilities, a solution is a vector in $\{0,1\}^n$ with exactly k ones.

Our neighborhood will be those vectors within Hamming distance 2; that is, those vectors that differ by a single swap. Formally, a swap $\langle s, s' \rangle$ is $S - \{s\} + \{s'\}$ where $s \in S, s' \notin S$.

The algorithm that we analyze is described in the following figure. Here $\varepsilon > 0$ is a constant and $p(n)$ is a polynomial in the number of cities.

k-median Local Search Algorithm

1. $S \leftarrow$ an arbitrary feasible solution.
2. **while** there exists a swap operation op with
$$\mathrm{cost}(\mathrm{op}(S)) \leq (1 - \frac{\varepsilon}{p(n)})\,\mathrm{cost}(S),$$
do $S \leftarrow \mathrm{op}(S)$.
3. **return** S.

We call an operation op *admissible* for S if $\mathrm{cost}(\mathrm{op}(S)) \leq (1 - \varepsilon/p(n)) \cdot \mathrm{cost}(S)$. At any execution of step 2 of the algorithm, there will be at most a polynomial number of operations to be checked for admissibility. During each admissible operation, the cost of the current solution decreases by a factor of at least $\varepsilon/p(n)$. If S^* denotes an optimum solution and S_0 denotes the initial solution, then the number of operations (swaps) that the algorithm does is at most

$$\log\left(\mathrm{cost}(S_0)/\,\mathrm{cost}(S^*)\right) / \log \frac{1}{1 - \varepsilon/p(n)}.$$

As $\log(\mathrm{cost}(S_0))$ is polynomial in the input size and performing each operation takes polynomial time, this algorithm terminates in polynomial time.

To simplify the exposition, we work with the assumption that the algorithm terminates when no operation improves the cost of the solution, that is, for every operation op,

$$\text{cost}(\text{op}(S)) \geq \text{cost}(S).$$

We use the following notation. Let S denote the output of the algorithm and S^* denote an optimum solution. The cost of serving a client j, or the *service cost* of j, is the distance between j and the facility that serves it. Let s_j and o_j denote the service costs of j in the solutions S and S^* respectively (notice that o is a facility in S^* while o_j is the service cost of j in S^*). Let $N_S(s)$ denote the set of clients in $\mathcal{C}$ that are served by a facility $s \in S$ in the solution S. Similarly, let $N_{S^*}(o)$ denote the set of clients in $\mathcal{C}$ that are served by a facility $o \in S^*$ in the solution S^*. For a subset $A \subseteq S$ and $B \subseteq S^*$, let $N_S(A) = \cup_{s \in A} N_S(s)$ and $N_{S^*}(B) = \cup_{o \in B} N_{S^*}(o)$.

We now show the locality gap for this problem, and we will obtain a $(5+\varepsilon)$-approximation local search algorithm using the $2'''$ idea from above.

Theorem 3.2. *The locality gap for the* k**-median Local Search Algorithm** *is at most 5.*

Proof. This proof is from [3]. From the local optimality of S, we know that any swap $\langle s, o \rangle$ for $s \in S$ and $o \in S^*$,

$$\text{cost}(S - s + o) \geq \text{cost}(S) \quad \text{for all } s \in S, o \in S^* \tag{3}$$

Note that even if $S \cap S^* \neq \emptyset$, the above inequalities hold. In the remainder of this proof, we will combine k inequalities (each inequality corresponds to a swap which will be defined carefully later) to show that, $\text{cost}(S) \leq 5 \cdot \text{cost}(S^*)$.

Consider a facility $o \in S^*$; we partition $N_{S^*}(o)$ into subsets $p_s = N_{S^*}(o) \cap N_S(s)$ for $s \in S$. Consider a 1-1 and onto mapping $\pi : N_{S^*}(o) \rightarrow N_{S^*}(o)$ with the following property.

For all $s \in S$ such that, $|p_s| \leq \frac{1}{2}|N_{S^*}(o)|$, we have $\pi(p_s) \cap p_s = \emptyset$.

It is easy to see that such a mapping exists. For example, we can order the cities in $N_{S^*}(o)$ based on the partitions defined (i.e. $p_{s_1}, p_{s_2}, \ldots, p_{s_n}$) Notice that $|N_{S^*}(o)| \leq n$, some sets p_{s_i} may be empty. Suppose $|N_{S^*}(o)| = a$; then, we can define π by mapping the ith element in the order of $N_{S^*}(o)$ to the $(i + a/2)$-th element (addition over modulus a). It is not difficult to verify that π is a 1-1 onto mapping satisfying the property. Furthermore, observe that the set of $N_{S^*}(o)$ for $o \in S^*$ partition the set of cities S^*, π is also a 1-1 onto mapping on S^* (so it is also a 1-1 onto mapping on S). The property of this mapping will be used to reassign the clients when we do swapping and then to bound the total cost of the reassignments.

We say that a facility $o \in S^*$ is *captured* by a facility $s \in S$ if s serves more than half the clients served by o, that is, $|N_S(s) \cap N_{S^*}(o)| > \frac{1}{2}|N_{S^*}(o)|$. Note that a facility $o \in S^*$ is captured by at most one $s \in S$. We call a facility $s \in S$ *bad* if it captures some facility S^* and *good* otherwise.

We now consider k swaps (the k swaps that establish the approximation ratio of the algorithm), one for each facility in S^*. If some bad facility $s \in S$ captures exactly one facility $o \in S^*$ then we consider the swap $\langle s, o\rangle$. Suppose l facilities in S (and hence l facilities in S^*) are not considered in such swaps. These l facilities in S are either good or bad, and the bad facilities capture at least two facilities in S^*. Hence there are at least $l/2$ good facilities in S. Now, consider l swaps in which the remaining l facilities in S^* get swapped with the good facilities in S such that each good facility is swapped out at most twice (notice that there are l independent swaps).

It is easy to verify that the swaps considered above satisfy the following properties.

1. Each $o \in S^*$ is swapped-in exactly once.
2. Each $s \in S$ is swapped-out at most twice. This is because a facility in S that captures more than one facility in S^* is never swapped-out and a facility that captures exactly one facility in S^* is swapped only with the facility that it captures.
3. If a swap $\langle s, o\rangle$ is considered, the facility s does not capture any facility $o' \neq o$.

We now analyze these swaps by considering an arbitrary swap $\langle s, o\rangle$. We place an upper bound on the increase in cost due to this swap by reassigning the clients in $N_S(s) \cup N_{S^*}(o)$ to the facilities in $S - s + o$ as follows.

- The clients $j \in N_{S^*}(o)$ are assigned to o.
- The clients $j' \in N_S(s) - N_{S^*}(o)$ are assigned to the unique $s' \in S$ with $\pi(j') \in N_S(s')$.

Consider a client $j' \in N_S(s) - N_{S*}(o)$. It is served by a unique $o' \in S^*$. As s does not capture o', we have $|N_S(s) \cap N_{S^*}(o')| \leq \frac{1}{2}|N_{S^*}(o')|$ and hence, by the property of π, we have that $\pi(j') \notin N_S(s)$ and thus $s \neq s'$. Note that when we reassign client j' from s to s', the distance that the client j' travels to the nearest facility in $S - s + o$ is at most $c_{j's'}$. By triangle inequality, $c_{j's'} \leq c_{j'o} + c_{o\pi(j')} + c_{\pi(j')s'} = o_{j'} + o_{\pi(j')} + s_{\pi(j')}$. The remaining clients continue to be assigned to the old facilities. From inequality (3) we have,

$$\text{cost}(S - s + o) - cost(S) \geq 0 \tag{4}$$

Therefore,

$$\sum_{j \in N_{S^*}(o)} (o_j - s_j) + \sum_{j \in N_S(s),\ j \notin N_{S^*}(o)} (o_j + o_{\pi(j)} + s_{\pi(j)} - s_j) \geq 0 \qquad (5)$$

Now we sum the k swaps defined before. As each facility $o \in S^*$ is swapped-in exactly once, the first term of inequality (5) added over all the k swaps gives exactly $\text{cost}(S^*) - \text{cost}(S)$. For the second term, we use the fact that each s is swapped-out at most twice. Also for any $j \in \mathcal{C}$, as s_j is the shortest distance from j to the facility in S, we get, using triangle inequality, $o_j + o_{\pi(j)} + s_{\pi(j)} \geq s_j$. Thus the second term of inequality (5) added over all the k swaps is not greater than

$$2 \cdot \sum_{j \in \mathcal{C}} (o_j + o_{\pi(j)} + s_{\pi(j)} - s_j).$$

Since π is a 1-1 onto mapping, $\sum_{j \in \mathcal{C}} o_j = \sum_{j \in \mathcal{C}} o_{\pi(j)} = \text{cost}(S^*)$ and $\sum_{j \in \mathcal{C}} (s_{\pi(j)} - s_j) = 0$. Thus, $2 \sum_{j \in \mathcal{C}} (o_j + o_{\pi(j)} + s_{\pi(j)} - s_j) = 4 \cdot \text{cost}(S^*)$. Combining the two terms we get, $\text{cost}(S^*) - \text{cost}(S) + 4 \cdot \text{cost}(S^*) \geq 0$. This completes the proof of the theorem. □

If in our algorithm we use idea $2'''$ and stop early (when we are close to a local optimum), our cost in (4) is replaced with

$$\text{cost}(S - s + o) - \left(1 - \frac{\varepsilon}{k}\right) \text{cost}(S) \geq 0 \qquad (4')$$

We get a $(\frac{5}{1-\varepsilon})$-approximation algorithm by following the same analysis (notice that if we sum k inequalities of $(4')$, we have added $\varepsilon \cdot \text{cost}(S)$ in total). If we allow r swaps to define our neighborhood, this can be generalized to a $(3 + \frac{2}{r})$-approximation for r less than k. (The above analysis was the case $r = 1$.)

References

[1] Vijay V. Vazirani. *Approximation Algorithms.* Springer, 2001.

[2] Sanjeev Khanna, Rajeev Motwani, Madhu Sudan, and Umesh Vazirani. On syntactic versus computational views of approximability. *SIAM Journal on Computing*, 28(1):164–191, 1999. Preliminary version in Proc. of FOCS'94.

[3] Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala and Vinayaka Pandit. Local search heuristics for k-median and facility location problems. *STOC'01*, pages 21–29, July 2001.