CS880: Approximations Algorithms	
Scribe: Chi Man Liu	**Lecturer:** Shuchi Chawla
Topic: Local Search: Max-Cut, Facility Location	**Date:** 2/13/2007

In previous lectures we saw how dynamic programming could be used to obtain PTAS for certain NP-hard problems. In this lecture we will discuss local search and look at approximation algorithms for two problems — Max-Cut and Facility Location.

7.1 Local Search

Local search is a heuristic technique for solving hard optimization problems and is widely used in practice. Suppose that a certain optimization problem can be formulated as finding a solution maximizing an objective function in the solution space. A local search algorithm for this problem would start from an arbitrary or carefully chosen solution, then iteratively move to other solutions by making small, local changes to the solution, each time increasing the objective function value, until a local optimum is found.

We view the solution space as a graph where vertices are the candidate solutions. An edge exists between two solutions if we can move from one to the other by making some small changes. A solution is a global optimum if its objective function value is maximized (or minimized, depending on the context) over all candidate solutions. A solution is a local optimum if none of its neighbor solutions has a greater (or smaller) objective function value.

When we apply local search to solving optimization problems we require that the solution graph has low (polynomially bounded) vertex degrees, so that at each vertex only polynomial time is needed to find a "good" neighbor. In order to show that the algorithm runs in polynomial time, we need to show that starting from any solution we can reach a local optimum in a polynomial number of steps. It doesn't suffice to show that the diameter of the solution graph is polynomial. We will see a few different arguments including a potential function based argument. And lastly, to show that the algorithm is indeed a good approximation algorithm, we must prove that any local optimum is nearly as good as a global optimum.

In the following sections we will see how local search can be used as approximation algorithms for two problems. The first problem we consider is the *Max-Cut* problem. The second problem is the *Facility Location* problem for which the analysis is a bit more involved. We will defer the last part of the analysis to the next lecture.

7.2 Max-Cut

7.2.1 The Problem

We first review the notion of a *cut* in graph theory.

Definition 7.2.1 *Let $G = \{V, E\}$ be a weighted graph where each edge $e \in E$ has a weight w_e. A*

cut *is a partition of* V *into two subsets* S *and* $S' = V \backslash S$*. We simply denote a cut by either one of its subsets. The value of a cut* S *is*

$$c(S) = \sum_{\substack{(s,s') \in E \\ s \in S, s' \in S'}} w_{s,s'}.$$

Now we can define the Max-Cut problem on weighted graphs.

Definition 7.2.2 (The Max-Cut Problem) *Given a weighted graph* $G = \{V, E\}$ *where each edge* $e \in E$ *has a positive integral weight* w_e*, find a cut in* G *with maximum cut value.*

While the Min-Cut problem is polynomial-time solvable by reducing it to Maximum Flow, the Max-Cut problem is known to be NP-hard. It has been shown that approximating Max-Cut to a factor of 17/16 is still NP-hard. In the following we present a simple 2-approximation local search algorithm for Max-Cut.

7.2.2 The Algorithm

The algorithm we are going to describe is a straightforward application of local search. The only thing we need to specify is the small changes we are allowed to make in each step. Since each candidate solution is a just a partition of the vertex set V, the following local step seems natural: Two partitions S_1 and S_2 are joined by an edge if and only if $|S_1 \backslash S_2| \leq 1$ and $|S_2 \backslash S_1| \leq 1$. Intuitively this means moving one vertex from one subset to the other. Thus we obtained the following algorithm:

1. Start with an arbitrary partition (for example, $\emptyset$).
2. Pick a vertex $v \in V$ such that moving it across the partition would yield a greater cut value.
3. Repeat step 2 until no such v exists.

Before we move on to prove that this algorithm is 2-approximate, let us first analyze its running time. Each step involves examining at most $|V|$ vertices and selecting one that increases the cut value when moved. This process takes $O(|V|^2)$ time. Since we assume that edges have positive integral weights, the cut value is increased by at least 1 after each iteration. The maximum possible cut value is $\sum_{e \in E} w_e$, hence there are at most such number of iterations. The overall time complexity of this algorithm is thus $O(|V|^2 \sum_{e \in E} w_e)$. For the special case of unweighted graphs (all weights equal to 1), the time complexity becomes $O(|V|^4)$, which is strongly polynomial in the input size.

Note. There exist modifications to the algorithm that give a strongly polynomial running time, with an additional ϵ term in the approximation coefficient. An example of such modifications will be shown when we discuss the Facility Location problem.

We now proceed to prove that the above algorithm is 2-approximate.

Theorem 7.2.3 *The local search algorithm described above gives a 2-approximation to the Max-Cut problem.*

Proof: First of all, we observe that the maximum cut value cannot be larger than the sum of all edge weights, thus giving

$$\sum_{e \in E} w_e \geq OPT.$$

We say that an edge *contributes* to a cut if its endpoints lie in different subsets of the cut. Let S be a cut produced by our algorithm. Let v be a vertex in S. Consider the set E_v of edges incident to v. If we move v from S to $S' = V \backslash S$, edges in E_v that contributed to S become non-contributing, and vice versa. Edges not in E_v are not affected. Since S is a local optimum, moving v to S' does not increase the cut value. Therefore we have

$$\begin{aligned}
\sum_{\substack{u \in S' \\ (u,v) \in E}} w_{u,v} &\geq \sum_{\substack{u \in S \\ (u,v) \in E}} w_{u,v} \\
2 \sum_{\substack{u \in S' \\ (u,v) \in E}} w_{u,v} &\geq \sum_{\substack{u \in S \\ (u,v) \in E}} w_{u,v} + \sum_{\substack{u \in S' \\ (u,v) \in E}} w_{u,v} \\
\sum_{\substack{u \in S' \\ (u,v) \in E}} w_{u,v} &\geq \frac{1}{2} \sum_{u:(u,v) \in E} w_{u,v}.
\end{aligned}$$

Similarly, for any $v' \in S'$ we get

$$\sum_{\substack{u \in S \\ (u,v') \in E}} w_{u,v'} \geq \frac{1}{2} \sum_{u:(u,v') \in E} w_{u,v}.$$

Summing over all vertices, we obtain the desired result:

$$2c(S) = 2 \sum_{\substack{u \in S, v \in S' \\ (u,v) \in E}} w_{u,v} \geq \sum_{e \in E} w_e \geq OPT.$$

■

There are many other 2-approximate algorithms for Max-Cut. One example is the following simple randomized algorithm: Construct the cut by uniformly assigning each vertex to one of of two subsets. It is trivial to show by linearity of expectation that this algorithm produces a cut with an expected value of $\frac{1}{2} \sum_{e \in E} w_e \geq \frac{1}{2} OPT$. This algorithm can be made deterministic by applying derandomization.

As a sidenote, the best known approximation for Max-Cut is 1.134-approximate [1]. This algorithm is based on semidefinite programming. The approximation factor (1.134) arises from geometric arguments and is suspected to be the best possible. (We will learn more about this in a subsequent lecture.) It is not likely that this approximation result can be improved unless P = NP [2].

7.3 Facility Location

7.3.1 The Problem

Suppose there is a set of customers and a set of facilities serving these customers. Our goal is to open some facilities and assign each customer to the nearest opened facility. Each facility has an opening cost. Each customer has a routing cost, which is proportional to the distance (see the next paragraph) traveled. The Facilitiy Location problem asks for a subset of facilities to be opened such that the total cost (opening costs plus routing costs) is minimized. While the aforementioned "distance" could mean geographic distance, in general any metric would do. Recall the definition of a metric:

Definition 7.3.1 *Let M be a set. A* metric *on M is a function $d : M \times M \to \mathbb{R}$ that satisfies the following three properties:*

1. *$d(x, y) \geq 0$; and $d(x, y) = 0$ iff $x = y$ (non-negativity)*
2. *$d(x, y) = d(y, x)$ (symmetry)*
3. *$d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality)*

The formal definition of the the Facility Location problem is given below.

Definition 7.3.2 (The Facility Location Problem) *Let X be a set of facilities and Y be a set of customers. Let c be a metric defined on $X \cup Y$. For each $i \in X$, let f_i be the cost of opening facility i. Let S be a subset of X. For each $j \in Y$, the routing cost of customer j with respect to S is $r_j^S = \min_{i \in S} c(i, j)$. The total cost of S is $C(S) = C_f(S) + C_r(S)$, where $C_f(S) = \sum_{i \in S} f_i$ is the facility opening cost of S and $C_r(S) = \sum_{j \in Y} r_j^S$ is the routing cost of S. The* Facility Location *problem asks for a subset $S^* \subseteq X$ that minimizes $C(S^*)$ over all subsets of X.*

The requirement that c is a metric seems arbitrary at first sight. However, if we let c be any general weight function, we will be able to reduce Set Cover to this relaxed Facility Location problem, implying that it is unlikely to be approximable within a constant factor (see lecture 3). By imposing the metric constraints, we are able to get a constant factor approximation algorithm for Facility Location.

The Facility Location problem arises very often in operations research. There are many variants of the problem, such as Capacitated Facility Location, in which each facility can only serve a certain number of customers. Another variant is the k-Median problem, in which facilities have no opening costs but at most k of them can be opened.

We remark that the best known approximation algorithm for Facility Location has an approximation coeffcient of 1.52 [3]. On the other hand, it has been shown that approximating this problem within a factor of 1.46 is NP-hard unless NP $\subseteq$ DTIME($n^{\log \log n}$) [4].

7.3.2 The Algorithm

We present a local search algorithm for approximating Facility Location within a factor of $5 + \epsilon$ for any constant $\epsilon > 0$ [5]. In fact, it can be shown that this algorithm is a $(3 + \epsilon)$-approximation [6],

but the analysis is more involved and we will not go through that in class.

We start by specifying the solution graph. Each vertex in the graph represents a subset of facilities to be opened. At each step, we can add a facility to the subset, remove a facility from a subset, or swap a facility in the subset with one outside the subset. The algorithm is as follows:

1. Start with an arbitrary subset of facilities.
2. Carry out an operation (add, remove or swap) that leads to a decrease in the total cost by a factor of at least $(1 - \epsilon/p(n))$, where $p(n)$ is a polynomial to be determined later.
3. Repeat step 2 until such an operation does not exist.

We first bound the running time of this algorithm. Let $n = |X| + |Y|$. For easier analysis we assume that all distances and costs are integers. Observe that the cost of any solution is bounded from above by some polynomial in n, c and f. Let $q(n, c, f)$ be such a polynomial. The cost of the solution after t iterations is at most $(1 - \epsilon/p(n))^t q(n, c, f)$. After $p(n)/\epsilon$ iterations, the cost is at most $q(n, c, f)/e$ since $(1 - x)^{1/x} \approx 1/e$. Thus the algorithm halts after $O(\log q(n, c, f)p(n)/\epsilon)$ steps, which is polynomial in $1/\epsilon$ and the size of the instance (in binary).

Theorem 7.3.3 says that any local optimum in the solution graph is nearly as good as any global optimum. Note that in the analysis, we will assume that in step 2 of the algorithm we carry out an operation as long as there is some improvement, even if the improvement is not by a factor of at least $(1 - \epsilon/p(n))$. We will deal with this issue later on, and instead of a 5-approximation we will only get a $(5+\epsilon)$-approximation. This theorem follows directly from Lemma 7.3.4 and Lemma 7.3.5 which we will see in a second.

Theorem 7.3.3 *Let S^* be a global optimum and S be a local optimum. Then $C(S) \leq 5C(S^*)$.*

Remark. As mentioned at the beginning of this section, the bound on $C(S)$ in Theorem 7.3.3 can be improved to $3C(S^*)$. We will not go into details.

For the proofs of the two lemmas we need a few notations. We define, for any $j \in Y$,

- $\sigma^*(j)$ = facility that customer j is assigned to in S^*
- $\sigma(j)$ = facility that customer j is assigned to in S
- $r^*(j)$ = routing cost for customer j in S^*
- $r(j)$ = routing cost for customer j in S.

Lemma 7.3.4 *Let S^* be a global optimum and S be a local optimum. Then $C_r(S) \leq C(S^*)$.*

Proof: Let $i^* \in S^*$. Consider adding i^* to S. Since S is a local optimum, this operation does not decrease the total cost, and therefore

$$f_{i^*} + \sum_{j:\sigma^*(j)=i^*} (r^*(j) - r(j)) \geq 0.$$

Summing over all $i^* \in S^*$, we get

$$\begin{aligned} C_f(S^*) + C_r(S^*) - C_r(S) &\geq 0 \\ C(S^*) - C_r(S) &\geq 0. \end{aligned}$$

■

Lemma 7.3.5 *Let S^* be a global optimum and S be a local optimum. Then $C_f(S) \leq 2C(S^*) + 2C_r(S) \leq 4C(S^*)$.*

Proof: To bound $C_f(S)$, we consider swapping facility i in S for the nearest facility i^* in S^*. We reassign all customers of i to i^* in S^*. Since S is a local optimum, this operation does not increase the total cost, and hence we can bound the opening cost of i by the increase in routing cost plus the opening cost of i^*. However, the opening cost of i^* will be charged multiple times in our analysis if there is more than one facility in S whose nearest facility in S^* is i^*. To avoid this, we only charge the opening cost of i^* to the one nearest to i^* among those facilities. We call that facility *primary*, and the rest *secondary*. A formal argument is presented below.

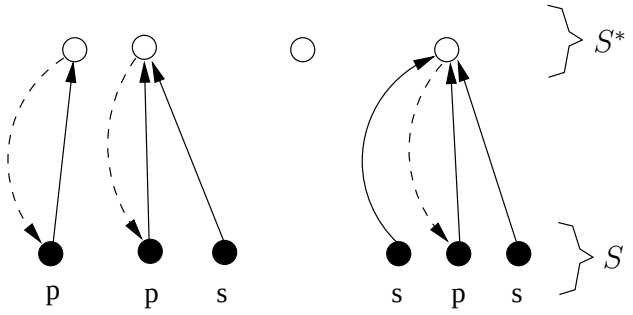

Figure 7.3.1: Primary and secondary facilities. Solid arrows represent the σ function; dotted arrows represent the π function; 'p' means primary and 's' means secondary.

For any $i \in S$, let $\sigma^*(i)$ be the facility in S^* which is nearest to i, i.e. $\sigma^*(i) = \arg\min_{i^* \in S^*} c(i, i^*)$. For any $i^* \in \{\sigma^*(i) \mid i \in S\} \subseteq S^*$, let $\pi(i^*)$ be the primary facility associated to i^*, i.e. $\pi(i^*) = \arg\min_{i \in S : \sigma^*(i) = i^*} c(i, i^*)$.

For any $i \in S$, let R_i be the total routing cost of all customers of i in S, i.e. $R_i = \sum_{j:\sigma(j)=i} r_j$; let R_i^* be the total routing cost of the same set of customers in S^*, i.e. $R_i^* = \sum_{j:\sigma(j)=i} r_j^*$. Note that R_i and R_i^* sum over the same set of customers — R_i^* does not sum over $\{j \mid \sigma^*(j) = i\}$.

Claim 7.3.6 *If $i \in S$ is primary, then $f_i \leq R_i + R_i^* + f_{i^*}$.*

Proof: Consider swapping i for i^*, then assigning all customers of i to i^*. Since S is a local optimum, such an operation would not decrease the total cost, and therefore

$$f_{i^*} - f_i + \sum_{j:\sigma(j)=i} (c(j, i^*) - c(j, i)) \geq 0. \tag{7.3.1}$$

We are going to show that the term $c(j, i^*) - c(j, i)$ in the summation is small. By the triangle inequality, we have $c(j, i^*) - c(j, i) \leq c(i, i^*)$. Now since i^* is the facility in S^* that is closest to i,

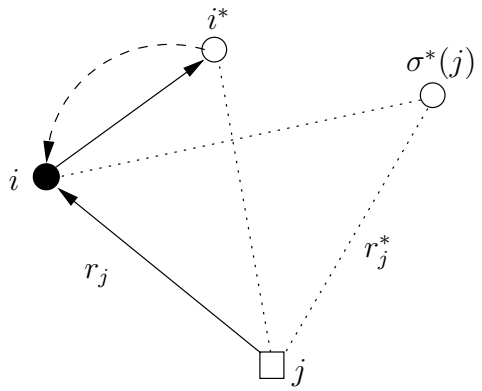

Figure 7.3.2: Entities in Claim 7.3.6.

and noting that $\sigma^*(j) \in S^*$, we get $c(i, i^*) \leq c(i, \sigma^*(j))$. By applying the triangle inequality again, $c(i, \sigma^*(j)) \leq c(j, i) + c(j, \sigma^*(j)) = r_j + r_j^*$. Combining the above, we get the following inequality:

$$c(j, i^*) - c(j, i) \leq r_j + r_j^*.$$

Now we can substitute it into Equation 7.3.1 to obtain the desired result:

$$\begin{aligned} 0 &\leq f_{i^*} - f_i + \sum_{j:\sigma(j)=i} (r_j + r_j^*) \\ &\leq f_{i^*} - f_i + R_i + R_i^*. \end{aligned}$$

■

For a secondary facility i in S, we remove it and assign its customers to a nearby facility in S. Specifically, we assign them to the primary facility associated to $\sigma^*(i)$.

Claim 7.3.7 *If $i \in S$ is secondary, then $f_i \leq 2(R_i + R_i^*)$.*

Proof: Let $i^* = \sigma^*(i)$ and $i' = \pi(i^*)$. Consider removing i from S and assigning its customers to i'. Let j be a customer assigned to i in S, and consider the increase in its routing cost:

$$\begin{aligned} c(i', j) - c(i, j) &\leq c(i, i^*) + c(i^*, i') \\ &\leq 2c(i, i^*) \\ &\leq 2c(i, \sigma^*(j)) \\ &\leq 2(r_j + r_j^*) \end{aligned}$$

The first and last inequalities follow from the triangle inequality; the second inequality follows from the fact that i' is the primary facility associated to i^*; the third inequality follows from the fact that i^* is the facility in S^* closest to i.

Summing over all customers of i, we have shown that the total increase in routing costs is no more than $2(R_i + R_i^*)$. Since the increase in the total cost is non-negative, we get

$$-f_i + 2(R_i + R_i^*) \geq 0.$$

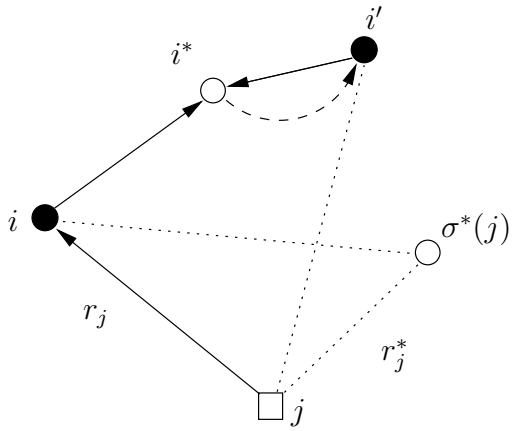

Figure 7.3.3: Entities in Claim 7.3.7.

■

From the above two claims we obtain

$$\begin{aligned} C_f(S) &\leq C_f(S^*) + 2\sum_{i \in S}(R_i^* + R_i) \\ &= C_f(S^*) + 2C_r(S^*) + 2C_r(S) \\ &\leq 2C(S^*) + 2C_r(S). \end{aligned}$$

Finally, by Lemma 7.3.4 we have

$$C_f(S) \leq 2C(S^*) + 2C_r(S) \leq 4C(S^*).$$

■

Finally, we will show that carrying out step 2 only when the cost decreases significantly increases the approximation factor by only a small amount. Our algorithm moves from one solution to another only if the total cost decreases by a factor of at least $(1 - \epsilon/p(n))$, so it may not halt at a local optimum. Nevertheless, we can still bound the cost of the solution by slightly modifying the above proofs. In Lemma 7.3.4 and Lemma 7.3.5, we can replace 0 with $(-\epsilon/p(n))C(S)$ when we set up inequalities for the increases in cost. There are less than n^2 inequalities. Summing them up we get $(1 - \epsilon n^2/p(n))C(S) \leq 5C(S^*)$. By choosing $p(n) = n^2$, we have shown that $C(S)$ is a $(5 + \epsilon)$-approximation for $C(S^*)$.

Recall that our analysis of the algorithm is not tight — the bound can be improved to $C(S) \leq 3C(S^*)$ by reassigning customers more carefully in Lemma 7.3.5. In the following, we present an example showing that this is in fact the best we can get from this local search algorithm.

Suppose that there are $k + 2$ facilities and $k + 1$ customers "connected" in the way shown in Figure 7.3.4. (The distance between any two entities is the shortest path distance between them. Note that shortest path distances on a graph always form a metric.) The opening cost of facility x_{k+2} is $2k$; other facilities have zero opening cost. The global optimum $S^* = \{i_1, \ldots, i_{k+1}\}$ has

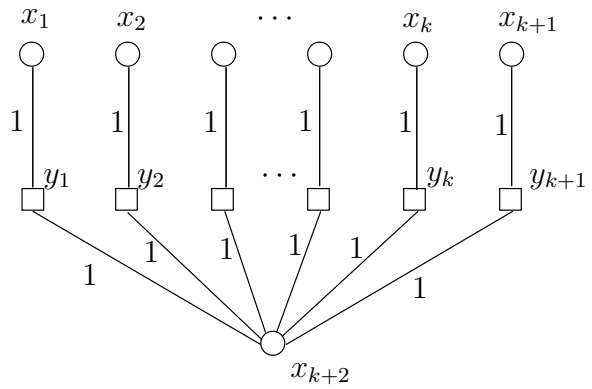

Figure 7.3.4: A worst case example for our local search algorithm.

total cost $k+1$. Let $S = \{x_{k+2}\}$. The total cost of S is $3k+1$. We claim that S is a local optimum: We cannot remove a facility from S since that would result in the empty set. Adding a facility does not help either. Swapping x_{k+2} for another facility, say x_m, does not decrease the total cost, since each customer except y_m is 3 units away from x_1. The total cost of S is roughly 3 times that the total cost of S^*, thus showing the tightness of the bound.

References

[1] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. In *JACM*, 42, 1115–1145, 1995.

[2] S. Khot, G. Kindler, E. Mossel and R O'Donnell. Optimal inapproximability results for max-cut and other 2-Variable CSPs? In *45th Annual IEEE Symposium on Foundations of Computer Science*, pp. 146–154, 2004.

[3] M. Mahdian, Y. Ye and J. Zhang. Improved approximation algorithms for metric facility location problems. In *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization*, pp. 229–242, September 17–21, 2002.

[4] S. Guha and S. Khuller. Greedy Strikes Back: Improved Facility Location Algorithms. In *Journal of Algorithms*, Volume 31, Issue 1, pp. 228–248, 1999.

[5] M. R. Korupolu, C. G. Plaxton and R. Rajaraman. Analysis of a local search heuristic for facility location problems. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1–10, 1998.

[6] V. Arya, N. Garg, R. Khandekar, K. Munagala and V. Pandit. Local search heuristics for k-median and facility location problems. In *Symposium on Theory of Computing*, pp. 21–29, 2001.