CS 598CSC: Approximation Algorithms	Lecture date: March 2, 2011
Instructor: Chandra Chekuri	Scribe: CC

Local search is a powerful and widely used heuristic method (with various extensions). In this lecture we introduce this technique in the context of approximation algorithms. The basic outline of local search is as follows. For an instance I of a given problem let $\mathcal{S}(I)$ denote the set of feasible solutions for I. For a solution S we use the term *(local) neighborhood* of S to be the set of all solution S' such that S' can be obtained from S via some *local moves*. We let $N(S)$ denote the neighborhood of S.

```
LocalSearch:
Find a "good" initial solution S_0 ∈ S(I)
S ← S_0
repeat
  If (∃S' ∈ N(S) such that val(S') is strictly better than val(S))
      S ← S'
  Else
      S is a local optimum
      return S
  EndIf
Until (True)
```

For minimization problems S' is strictly better than S if $\text{val}(S') < \text{val}(S)$ whereas for maximization problems it is the case if $\text{val}(S') > \text{val}(S)$.

The running time of the generic local search algorithm depends on several factors. First, we need an algorithm that given a solution S either declares that S is a local optimum or finds a solution $S' \in N(S)$ such that $\text{val}(S')$ is strictly better thatn $\text{val}(S)$. A standard and easy approach for this is to ensure that the local moves are defined in such a way that $|N(S)|$ is polynomial in the input size $|I|$ and $N(S)$ can be enumerated efficiently; thus one can check each $S' \in N(S)$ to see if any of them is an improvement over S. However, in some more advanced settings, $N(S)$ may be exponential in the input size but one may be able to find a solution in $S' \in N(S)$ that improves on S in polynomial time. Second, the running time of the algorithm depends also on the number of iterations it takes to go from S_0 to a local optimum. In the worst case the number of iterations could be $|\text{OPT} - \text{val}(S_0)|$ which need not be strongly polynomial in the input size. We will see that one can often use a standard scaling trick to overcome this issue; basically we stop the algorithm unless the improvement obtained over the current S is a significant fraction of $\text{val}(S)$. Finally, the quality of the initial solution S_0 also factors into the running time.

1 Local Search for Max Cut

We illustrate local search for the well-known Max Cut problem. In Max Cut we are given an undirected graph $G = (V, E)$ and the goal is to partition V into $(S, V \setminus S)$ so as to maximize the number of edges crossing S, that is, $|\delta_G(S)|$. In the weighted version each edge e has a non-negative weight $w(e)$ the goal is to maximize the weight of the edges crossing S, that is, $w(\delta_G(S))$ where $w(A) = \sum_{e \in A} w(e)$.

We consider a simple local search algorithm for MAX CUT that starts with an arbitrary set $S \subseteq V$ and in each iteration either adds a vertex to S or removes a vertex from S as long as it improves the cut capacity.

```
LocalSearch for Max Cut:
S ← ∅
repeat
  If (∃v ∈ V \ S such that w(δ(S + v)) > w(δ(S)))
      S ← S + v
  Else If (∃v ∈ S such that w(δ(S − v)) > w(δ(S)))
      S ← S − v
  Else
      S is a local optimum
      return S
  EndIf
Until (True)
```

We will first focus on the quality of solution output by the local search algorithm.

Lemma 1 *Let S be a local optimum outbut the local search algorithm. Then for each vertex v, $w(\delta(S) \cap \delta(v)) \geq w(\delta(v))/2$.*

Proof: Let $\alpha_v = w(\delta(S) \cap \delta(v))$ be the weight of edges among those incident to v ($\delta(v)$) that cross the cut S. Let $\beta_v = w(\delta(v)) - \alpha_v$.

We claim that $\alpha_v \geq beta_v$ for each v. If $v \in V \setminus S$ and $\alpha_v < \beta_v$ then moving v to S will strictly increase $w(\delta(S))$ and S cannot be a local optimum. Similarly if $v \in S$ and $\alpha_v < \beta_v$ $w(\delta(S - v)) > w(\delta(S))$ and S is not a local optimum. □

Corollary 2 *If S is a local optimum then $w(\delta(S)) \geq w(E)/2 \geq \mathrm{OPT}/2$.*

Proof: Since each edge is incident to exactly two vertices we have $w(\delta(S)) = \frac{1}{2}\sum_{v \in V} w(\delta(S) \cap \delta(v))$. Apply the above lemma,

$$\begin{aligned} w(\delta(S)) &= \frac{1}{2}\sum_{v \in V} w(\delta(S) \cap \delta(v)) \\ &\geq \frac{1}{2}\sum_{v \in V} w(\delta(v))/2 \\ &\geq \frac{1}{2} w(E) \\ &\geq \frac{1}{2}\mathrm{OPT}, \end{aligned}$$

since $\mathrm{OPT} \leq w(E)$. □

The running time of the local search algorithm depends on the number of local improvement iterations; checking whether there is a local move that results in an improvement can be done by trying all possible vertices. If the graph is unweighted then the algorithm terminates in at most $|E|$ iterations. However, in the weighted case, it is known that the algorithm can take an exponential

time in $|V|$ when the weights are large. Many local search algorithms can be modified slightly to terminate with an approximate local optimum such that (i) the running time of the modified algorithm is strongly polynomial in the input size and (ii) the quality of the solution is very similar to that given by the original local search. We illustrate these ideas for MAX CUT. Consider the following algorithm where $\epsilon > 0$ is a parameter that can be chosen. Let n be the number of nodes in G.

```
MODIFIED LOCALSEARCH FOR MAX CUT(ε):
S ← {v*} where v* = arg max_{v∈V} w(δ(v))
repeat
  If (∃v ∈ V \ S such that w(δ(S + v)) > (1 + ε/n)w(δ(S)))
      S ← S + v
  Else If (∃v ∈ S such that w(δ(S − v)) > (1 + ε/n)w(δ(S)))
      S ← S − v
  Else
      return S
  EndIf
Until (True)
```

The above algorithm terminates unless the improvement is a relative factor of $(1 + \frac{\epsilon}{n})$ over the current solution's value. Thus the final output S is an *approximate* local optimum.

Lemma 3 *Let S be the output of the modified local search algorithm for* MAX CUT. *Then $w(\delta(S)) \geq \frac{1}{2(1+\epsilon/4)} w(E)$.*

Proof: As before let $\alpha_v = w(\delta(S) \cap \delta(v))$ and $\beta_v = w(\delta(v)) - \alpha_v$. Since S is an approximately local optimum we claim that for each v

$$\beta_v - \alpha_v \leq \frac{\epsilon}{n} w(\delta(S)).$$

Otherwise a local move using v would improve S by more than $(1 + \epsilon/n)$ factor. (The formal proof is left as an exercise to the reader).

We have,

$$\begin{aligned}
w(\delta(S)) &= \frac{1}{2} \sum_{v \in V} \alpha_v \\
&= \frac{1}{2} \sum_{v \in V} ((\alpha_v + \beta_v) - (\beta_v - \alpha_v))/2 \\
&\geq \frac{1}{4} \sum_{v \in V} (w(\delta(v)) - \frac{\epsilon}{n} w(S)) \\
&\geq \frac{1}{2} w(E) - \frac{1}{4} \sum_{v \in V} \frac{\epsilon}{n} w(S) \\
&\geq \frac{1}{2} w(E) - \frac{1}{4} \epsilon \cdot w(S).
\end{aligned}$$

Therefore $w(S)(1 + \epsilon/4) \geq w(E)/2$ and the lemma follows. □

Now we argue about the number of iterations of the algorithm.

Lemma 4 *The modified local search algorithm terminates in $O(\frac{1}{\epsilon} n \log n)$ iterations of the improvement step.*

Proof: We observe that $w(S_0) = w(\delta(v^*)) \geq \frac{1}{2n} w(E)$ (why?). Each local improvement iteration improves $w(\delta(S))$ by a multiplicative factor of $(1 + \epsilon/n)$. Therefore if k is the number of iterations that the algorithm runs for then $(1 + \epsilon/n)^k w(S_0) \leq w(\delta(S_)$ where S is the final output. However, $w(\delta(S)) \leq w(E)$. Hence

$$(1 + \epsilon/n)^k w(E)/2n \leq w(E)$$

which implies that $k = O(\frac{1}{\epsilon} n \log n)$. □

A tight example for local optimum: Does the local search algorithm do better than 1/2? Here we show that a local optimum is no better than a 1/2-approximation. Consider a complete bipartite graph $K_{2n,2n}$ with $2n$ vertices in each part. If L and R are the parts a set S where $|S \cap L| = n = |S \cap R|$ is a local optimum with $|\delta(S)| = |E|/2$. The optimum solution for this instance is $|E|$.

Max Directed Cut: A problem related to Max Cut is Max Directed Cut in which we are given a directed edge-weighted graph $G = (V, E)$ and the goal is to find a set $S \subseteq V$ that maximizes $w(\delta^+_G(S))$; that is, the weight of the directed edges leaving S. One can apply a similar local search as the one for Max Cut. However, the following example shows that the output S can be arbitrarily bad. Let $G = (V, E)$ be a directed in-star with center v and arcs connecting each of $v_1, \ldots, v_n$ to v. Then $S = \{v\}$ is a local optimum with $\delta^+(S) = \emptyset$ while OPT $= n$. However, a minor tweak to the algorithm gives a 1/3-approximation! Instead of returning the local optimum S return the better of S and $V \setminus S$. This step is needed because the directed cuts are not symmetric.

2 Local Search for Submodular Function Maximization

In this section we consider the utility of local search for maximizing non-negative submodular functions. Let $f : 2^V \to \mathbb{R}_+$ be a *non-negative* submodular set function on a ground set V. Recall that f is submodular if $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$ for all $A, B \subseteq V$. Equivalently f is submodular if $f(A + v) - f(A) \geq f(B + v) - f(B)$ for all $A \subset B$ and $v \notin B$. f is *monotone* if $f(A) \leq f(B)$ for all $A \subseteq B$. f is *symmetric* if $f(A) = f(V \setminus A)$ for all $A \subseteq V$. Submodular functions arise in a number of settings in combinatorial optimization. Two important examples are the following.

Example: Coverage in set systems. Let $S_1, S_2, \ldots, S_n$ be subsets of a set $\mathcal{U}$. Let $V = \{1, 2, \ldots, n\}$ and define $f : 2^V \to \mathbb{R}_+$ where $f(A) = |\cup_{i \in A} S_i|$. f is a monotone submodular function. One can also associate weights to elements of $\mathcal{U}$ via a function $w : \mathcal{U} \to \mathbb{R}_+$; the function f defined as $f(A) = w(\cup_{i \in A} S_i)$ is also monotone submodular.

Example: Cut functions in graphs. Let $G = (V, E)$ be an undirected graph with non-negative edge weights $w : E \to \mathbb{R}_+$. The cut function $f : 2^V \to \mathbb{R}_+$ defined as $f(S) = \sum_{e \in \delta_G(S)} w(e)$ is a symmetric submodular function; it is not monotone unless the graph is trivial. If G is directed and we define f as $f(S) = \sum_{e \in \delta^+_G(S)} w(e)$ then f is submodular but is not necessarily symmetric.

The following problem generalizes MAX CUT and MAX DIRECTED CUT that we have already seen.

Problem: MAX SUBMOD FUNC. Given a non-negative submodular set function f on a ground set V via a *value oracle*[1] find $\max_{S \subseteq V} f(S)$.

Note that if f is monotone then the problem is trivial since V is the optimum solution. Therefore, the problem is interesting (and NP-Hard) only when f is not necessarily monotone. We consider a simple local search algorithm for MAX SUBMOD FUNC and show that it gives a 1/3-approximation and a 1/2-approximation when f is symmetric. This was shown in [2].

LOCALSEARCH FOR MAX SUBMOD FUNC:
$S \leftarrow \emptyset$
repeat
  If ($\exists v \in V \setminus S$ such that $f(S+v) > f(S)$)
    $S \leftarrow S + v$
  Else If ($\exists v \in S$ such that $f(S-v) > f(S)$)
    $S \leftarrow S - v$
  Else
    S is a *local optimum*
    return the better of S and $V \setminus S$
  EndIf
Until (True)

We start the analysis of the algorithm with a basic lemma on submodularity.

Lemma 5 *Let $f : 2^V \rightarrow \mathbb{R}_+$ be a submodular set function on V. Let $A \subset B \subseteq V$. Then*

- *If $f(B) > f(A)$ then there is an element $v \in B \setminus A$ such that $f(A+v) - f(A) > 0$. More generally there is an element $v \in B \setminus A$ such that $f(A+v) - f(A) \geq \frac{1}{|B \setminus A|}(f(B) - f(A))$.*
- *If $f(A) > f(B)$ then there is an element $v \in B \setminus A$ such that $f(B-v) - f(B) > 0$. More generally there is an element $v \in B \setminus A$ such that $f(B-v) - f(B) \geq \frac{1}{|B \setminus A|}(f(A) - f(B))$.*

We obtain the following corollary.

Corollary 6 *Let S be a local optimum for the local search algorithm and let S^* be an optimum solution. Then $f(S) \geq f(S \cap S^*)$ and $f(S) \geq f(S \cup S^*)$.*

Theorem 7 *The local search algorithm is a 1/3-approximation and is a 1/2-approximation if f is symmetric.*

Proof: Let S be the local optimum and S^* be a global optimum for the given instance. From the previous corollary we have that $f(S) \geq f(S \cap S^*)$ and $f(S) \geq f(S \cup S^*)$. Note that the algorithm outputs the better of S and $V \setminus S$. By submodularity, we have,

$$f(V \setminus S) + f(S \cup S^*) \geq f(S^* \setminus S) + f(V) \geq f(S^* \setminus S)$$

[1] A value oracle for a set function $f : 2^V \rightarrow \mathbb{R}$ provides access to the function by giving the value $f(A)$ when presented with the set A.

where we used the non-negativity of f in the second inequality. Putting together the inequalities,

$$\begin{aligned} 2f(S) + f(V \setminus S) &= f(S) + f(S) + f(V \setminus S) \\ &\geq f(S \cap S^*) + f(S^* \setminus S) \\ &\geq f(S^*) + f(\emptyset) \\ &\geq f(S^*) = \text{OPT}. \end{aligned}$$

Thus $2f(S) + f(V \setminus S) \geq \text{OPT}$ and hence $\max\{f(S), f(V \setminus S)\} \geq \text{OPT}/3$.

If f is symmetric we argue as follows. Using Lemma 5 we claim that $f(S) \geq f(S \cap S^*)$ as before but also that $f(S) \geq f(S \cup \bar{S}^*)$ where $\bar{A}$ is shorthand notation for the the complement $V \setminus A$. Since f is symmetric $f(S \cup \bar{S}^*) = f(V \setminus (S \cup \bar{S}^*)) = f(\bar{S} \cap S^*) = f(S^* \setminus S)$. Thus,

$$\begin{aligned} 2f(S) &\geq f(S \cap S^*) + f(S \cup \bar{S}^*) \\ &\geq f(S \cap S^*) + f(S^* \setminus S) \\ &\geq f(S^*) + f(\emptyset) \\ &\geq f(S^*) = \text{OPT}. \end{aligned}$$

Therefore $f(S) \geq \text{OPT}/2$. □

The running time of the local search algorithm may not be polynomial but one can modify the algorithm as we did for Max Cut to obtain a strongly polynomial time algorithm that gives a $(1/3 - o(1))$-approximation ($(1/2 - o(1)$ for symmetric). See [2] for more details. There has been much work on submodular function maximization including work on variants with additional constraints. Local search has been a powerful tool in these algorithms. See references for some of these results and further pointers.

References

[1] C. Chekuri, J. Vondrák, and R. Zenklusen. Submodular function maximization via the multilinear relaxation and contention resolution schemes. *Proc. of ACM STOC*, 2001. To appear.

[2] U. Feige, V. Mirrokni and J. Vondrák. Maximizing a non-monotone submodular function. *Proc. of IEEE FOCS*, 461–471, 2007.

[3] J. Lee, V. Mirrokni, V. Nagarajan and M. Sviridenko. Maximizing nonmonotone submodular functions under matroid and knapsack constraints. *SIAM J. on Disc. Math.*, 23(4): 2053–2078, 2010. Preliminary version in *Proc. of ACM STOC*, 323–332, 2009.

[4] S. Oveis Gharan and J. Vondrák. Submodular maximization by simulated annealing. *Proc. of ACM-SIAM SODA*, 1098–1116, 2011.