Approximation Algorithms

1 Intro to Approximation Algorithms

Given an optimization problem and not enough resources (time, space, etc.) to solve it optimally, approximation algorithms come in handy. They are especially useful and interesting when the problem is **NP**-hard. The problem must be an optimization version rather than a decision version because it's not clear what it means to approximate a yes or no answer.

Formally, we define inputs I, solutions S, a solution cost or reward (depending on minimization or maximization) $C(S)$. We also have an algorithm A which outputs $A(I)$ with $OPT(I)$ an optimal solution for input I.

Cost minimization: A is an α-approximation if for all inputs I, $C(A(I)) \leq \alpha C(OPT(I))$ with $\alpha \geq 1$.

Reward maximization: A is an α-approximation if for all inputs I, $C(A(I)) \geq \alpha C(OPT(I))$ with $\alpha \leq 1$.

Unless otherwise specified, we only care about polytime approximation algorithms.

Note: usually α is multiplicative because additive α is unit-dependent, but in rare cases additive α is appropriate (e.g., graph colorings).

The biggest problem in the analysis is that we don't know the value of $OPT(I)$ – if we did, we would know the optimal solution. To overcome this, we find relatively easy to compute upper or lower bounds (for maximization and minimization respectively) on $OPT(I)$, then prove that we are within an α-factor of those bounds, so we must be within an α-factor of $OPT(I)$.

2 Vertex Cover Approximation

VERTEX COVER: Given $\mathcal{G} = (V, E)$, find the smallest $S \subseteq V$ such that $e \in E$ contains ≥ 1 $v \in S$.

For any matching M, we have $|M| \leq OPT$ because each $v \in OPT$ can cover at most one $e \in M$. If M is any maximal matching and S is the set of all endpoints of $e \in M$, then S is a vertex cover of $\mathcal{G}$. Because each $e \notin M$ must have ≥ 1 vertex in common with some $e' \in M$, that vertex must be in S. Thus:

$$|S| \leq 2|M| \leq 2|OPT|$$

So this is a 2-approximation.

Algorithm 1 Vertex Cover Approximation Algorithm

```
1: while there is an uncovered edge $e = (u, v) \in E$ do
2:     Add both $u$ and $v$ to $S$
3: end while
```

The approximation hardness of VERTEX COVER is $\frac{7}{6}$. In other words, there exists no α-approximation for this problem with $\alpha < \frac{7}{6}$.

3 Traveling Salesman Approximation

UNDIRECTED TRAVELING SALESMAN (TSP): Given n cities with metric distances $d_{i,j}$, find the optimal tour T^* which includes all n cities and minimizes $\sum_{e=(u,v)\in T^*} d_{u,v}$.

We know that $C(MST) \leq C(OPT)$ because OPT with one edge removed is a spanning tree. To get a TSP tour T from a spanning tree, we do a DFS traversal and shortcut repeat visits. Each edge is traversed twice before shortcuts, so:

$$C(T) \leq 2C(MST) \leq 2C(OPT)$$

So this is a 2-approximation.

There also exists an easy 1.5-approximation called Christofide's Algorithm, and an even better guarantee has been recently obtained if $d_{i,j}$ is the shortest path distance in an unweighted graph with all edges 1. It is open for general metrics if there exists an α-approximation with $\alpha \leq 1.5$.

4 Steiner Tree Approximation

UNDIRECTED STEINER TREE: Given $\mathcal{G} = (V, E)$, $V = T \cup X$, find the subset $V' \subseteq V$ containing all nodes in T such that the spanning tree on V' minimizes total metric cost. In other words, the nodes in T must be included, and the nodes in X may be included if it makes the tree cheaper).

The approximation algorithm for this problem is simply to return $MST(T)$. Consider an optimal subset V' – then, the solution is the MST of V'. The TSP tour R of that MST has $C(R) \leq 2OPT$. Each node (in particular, each node from X) has degree ≤ 2. Thus, we can remove all $x \in X$ from R without cost increase. So, the optimal $C(R)$ on $T \leq 2MST(V')$.

$$C(MST(T)) \leq C(OPT) - C(R \text{ on } T) \leq 2C(MST(V')) = 2C(OPT)$$

So this is a 2-approximation.

5 Maximum Cut Approximation

MAXIMUM CUT: Given $\mathcal{G} = (V, E)$, find the partition $V = S \cup \bar{S}$ that maximizes the number of edges between S and $\bar{S}$.

Algorithm 2 Local Search Approximation for Maximum Cut

```
1: Start with an arbitrary $S$
2: while there is a node $v$ such that moving $v$ from $S$ to $\bar{S}$ or vice versa increases the number of edges
   across the cut do
3:     Move $v$
4: end while
```

At termination, each node v has at least as many edges to the other side as it does within its side (or else we would have moved it). So, $\geq \frac{1}{2}$ of each v's edges are cut, which means that $\geq \frac{1}{2}$ of all edges are cut. Since $|OPT| \leq |V|$, this is a $\frac{1}{2}$-approximation.

The approximation hardness of MAXIMUM CUT is 0.878 based on a technique called semi-definite programming (SDP) rounding.