CS880: Approximations Algorithms

Scribe: Michael Kowalczyk **Lecturer:** Shuchi Chawla
Topic: Intro, Vertex Cover, TSP, Steiner Tree **Date:** 1/23/2007

Today we discuss the background and motivation behind studying approximation algorithms and give a few examples of approximations for classic problems.

1.1 Introduction

Approximation algorithms make up a broad topic and are applicable to many areas. In this course a variety of different techniques for crafting approximation algorithms will be covered. Since there is no fixed procedure that will always work when attempting to solve new problems, the aim of this course is to achieve a solid understanding of the most relevant techniques and to get a feel for when they are likely to apply. For those with a particular interest in theory, another goal of this course is to provide enough background in the area so that current research papers on approximation algorithms can be read with relative ease.

1.2 Motivation

In practice, many important problems are NP-hard. If we want to have a good chance at solving them, we need to modify our goal in some way. We could use heuristics, but these are approaches that make no guarantees as to their effectiveness. In some applications, we may only need to solve a special case of the NP-hard problem, and this loss of generality may admit tractibilty. Alternately, average case analysis may be useful; since NP-hardness is a theory of worst case behavior, a problem can be NP-hard and yet very easy to solve on most instances. Finally, if the NP-hard problem deals with optimizing some parameter then we can try to design an approximation algorithm that efficiently produces a sub-optimal solution. It turns out that we can often design these algorithms in such a way that the quality of the output is guaranteed to be, say, within a constant factor of an optimal solution. This is the approach that we will investigate throughout the course.

1.3 Terminology

In order for approximations to make sense, we need to have some quality measure for the solutions. We will use OPT to denote the quality associated with an optimal solution for the problem at hand, and ALG to denote the (worst case) quality produced by the approximation algorithm under consideration. We would like to guarantee $ALG(I) \geq \frac{1}{\alpha} OPT(I)$ on any instance I for maximization problems, where α is known as the approximation factor. Note that $\alpha \geq 1$, and we allow α to be a function of some property of the instance, such as its size. For minimization problems, we write $ALG(I) \leq \alpha OPT(I)$ instead. In other words, we usually state approximations in such a way that $\alpha \geq 1$, although this standard is not universal.

Different problems exhibit different approximation factors. One class of problems have the property

that given any positive constant ϵ, we can give a $(1+\epsilon)$-approximation algorithm for it (although the runtime of the approximation algorithm is efficient for fixed ϵ, it may get much worse as ϵ becomes small). Problems with this property are said to have a polynomial time aproximation scheme, or $PTAS$. This is one of the best scenarios for approximation algorithms. At the other extreme, we can prove in some cases that it is NP-hard to give an approximation algorithm with α any better than, say n^ϵ. We can then categorize optimization problems in terms of how well they can be approximated by efficient algorithms.

Approximation factor (α)	Approximability
$1+\epsilon$ ($PTAS$)	Very approximable
1.1	
2	
constant c	
$\log n$	Somewhat approximable
$\log^2 n$	
$\sqrt{n}$	
n^ϵ	
n	Not very approximable

Figure 1.3.1: Some various degrees to which a problem can be approximated

We will also study the limits of approximability. Hardness of approximation results are proofs that no approximation better than a certain factor exists for a particular problem, under some hardness asumption. For example, set cover cannot be approximated to within an $o(\log n)$ factor, where n is the number of nodes, unless $NP = P$ (in which case we can solve it exactly). For some problems like set cover, we can get tight results, i.e. any improvement over the currently best known factor of approximation would imply $P = NP$ or refute another such complexity assumption. Other problems have a huge gap between the upper and lower approximability bounds currently known, so there is much variation from problem to problem.

1.4 Vertex Cover

One important step in finding approximation algorithms is figuring out what OPT is. But having a method for calculating OPT exactly is often enough to get a full solution for the problem. Vertex cover has this property.

Definition 1.4.1 (Vertex cover) *Given an undirected graph $G = (V, E)$, find a smallest subset $S \subseteq V$ such that every edge in E is incident on at least one of the vertices in S.*

Now assume that we have an algorithm that can tell us how many vertices are in an optimal vertex cover for any given graph. Then we can use this procedure to efficiently find a solution that is optimal. The idea is to observe how removing a vertex effects the size of an optimal vertex cover. If the number of vertices needed to make a vertex cover in the induced subgraph is reduced, then we know that we need to include the removed vertex in S. If the count remains the same, then we know that there is no harm in leaving that vertex out of S. We continue in this way to determine

which vertices to include in the vertex cover. Once our algorithm outputs zero for the remaining induced subgraph, S is an optimal vertex cover. This property of being able to reduce an instance to a smaller instance of the same problem is called *self-reducibility*.

It would be ideal to prove that an algorithm always outputs a solution that is within an α factor of OPT, even if we don't know what OPT is. This is where lower bounds come into play. Given a problem instance I, we can find some property Π such that $\Pi(I) \leq OPT(I)$. Then we get around calculating OPT by showing that $ALG(I) \leq \alpha \cdot \Pi(I) \leq \alpha \cdot OPT(I)$.

For vertex cover, we can look for a collection of edges such that no two of these edges share a vertex. Since any vertex cover would have to choose at least one of the vertices from each of these edegs, this shows that the size of any vertex cover for G is at least the size of any matching for G.

Lemma 1.4.2 *The size of any matching in G is a lower bound on the size of an optimal vertex cover in G.*

Proof: See above. ■

Since we want $\Pi(I)$ to be as large as possible, we choose our lower bound to be the size of a maximal matching for G. Given a maximal matching, a natural approximation to an optimal solution is just to include all vertices incident on the edges of our maximal matching. It is easy to see that this is an admissible solution since the existence of an edge that isn't incident on one of these vertices contradicts the maximality of the matching.

```
Approximate Vertex Cover(G = (V, E) - an undirected graph)
    Let M = any maximal matching on G
    Let S = all vertices incident on M
    return S
```

Theorem 1.4.3 *The above algorithm is a 2-approximation to vertex cover.*

Proof: Let $\Pi(I)$ be the size of the matching found by the above algorithm. Then $ALG(I) \leq 2 \cdot \Pi(I)$ and by lemma 1.4.2, $\Pi(I) \leq OPT(I)$, so $ALG(I) \leq 2 \cdot \Pi(I) \leq 2 \cdot OPT(I)$. ■

This is the best known approximation for vertex cover to date. In future lectures we will see other ways of obtaining the same approximation factor.

1.5 Metric TSP

Now we will consider the traveling salesperson problem.

Definition 1.5.1 (Traveling salesperson problem (TSP)) *Given a graph with weights assigned to the edges, find a minimum weight tour that visits every vertex exactly once.*

It's a good exercise to show that any reasonable approximation for TSP yields a solution for the Hamiltonian cycle problem (even when the graph is required to be the complete graph). Therefore, we will consider a less general version of the problem where we relax the ban on revisiting vertices.

Definition 1.5.2 (Metric TSP) *Given a graph with weights assigned to the edges, find a mini-*

mum weight tour that visits each vertex at least once.

This is called metric TSP because it is equivalent to solving the original TSP (without revisiting vertices) on the "metric completion" of the graph. By metric-completion, we mean that we put an edge between every pair of nodes in the graph with length equal to the length of the shortest path between them. The shortest path function on a connected graph forms a metric. A metric is an assignment of length to every pair of nodes such that the following hold for all u, v, and w.

$$\begin{aligned} d(u,v) &\geq 0 \\ d(u,v) &= 0 \quad \text{if and only if} \quad u = v \\ d(u,v) &= d(v,u) \\ d(u,v) &\leq d(u,w) + d(w,v) \qquad \text{(the triangle inequality)} \end{aligned}$$

This is a minimization problem, so now we look for lower bounds on OPT. Some possibilities include minimum spanning tree, the diameter of the graph, and TSP-path (i.e. a Hamiltonian path of minimum weight).

One desirable property for a lower bound is that it is as close to OPT as possible. Suppose we had a property Π such that there exist arbitrarily large instances I such that $OPT(I) = 100 \cdot \Pi(I)$. Then we can't hope to get α better than 100. In our case, if I is the complete graph with unit edge weights, and $\Pi(I)$ is the diameter of the graph then we see that $OPT(I) = n \cdot \Pi(I)$, so we won't be able to use graph diameter as a lower bound for metric TSP.

Another quality of a good lower bound is that the property is easier to understand or calculate than OPT itself. We may run into this trouble with TSP-path. However, minimal spanning tree can be computed efficiently, so we will try to use this as our lower bound (to see that it is a lower bound, note that an optimal tour with one edge removed is a tree).

If we use a depth first search tour of our tree as an approximation then we see that $ALG = 2 \cdot MST \leq 2 \cdot OPT$ so we have a 2-approximation of metric TSP.

Can this analysis be improved upon? The answer is no, because we can find a family of instances where $OPT(I) = 2 \cdot \Pi(I)$. The line graph suffices as an example of this. That is, let I be a graph on n vertices where each vertex connects only to the "next" and "previous" vertices (there will be a total of $n-1$ edges, each with edge weight 1). Then $OPT(I) = 2(n-1) = 2 \cdot \Pi(I)$. This example shows that we cannot get a better approximation for TSP using only the MST lower bound. But it doesn't preclude the possibility of a different algorithm using a different lower bound.

The crucial observation in improving this algorithm is to observe that our approach was really to convert the tree into an Eulerian graph and then use the fact there is an Eulerian tour. If we can find a more cost-effective way to transform the tree into an Eulerian graph, we could improve the approximation factor. An Eulerian tour exists if and only if every vertex has even degree (given that we want to start and end at the same vertex). Thus it suffices to find a matching between all nodes of odd degree in the minimal spanning tree and add these edges to the tree.

Lemma 1.5.3 *Given a weighted graph $G = (V, E)$ and a subset of vertices $S \subseteq V$ with even cardinality, a minimum cost perfect matching for S has weight at most half of a minimum cost metric TSP tour on G.*

Proof: Let S as in the statement of the lemma and consider a minimum cost TSP tour on S. Because of the metric property, this tour has a cost at most OPT. It also induces 2 matchings (take every other edge), and the smaller matching has cost at most $\frac{1}{2}OPT$. ■

APPROXIMATE METRIC TSP($G = (V, E)$ - a weighted undirected graph)
  Let M = minimal spanning tree in G
  Let U = set of vertices with odd degree in M
  Let P = minimum cost perfect matching on U in G
  return Eulerian tour on $M \cup P$

Theorem 1.5.4 *The above algorithm is a $\frac{3}{2}$-approximation to metric TSP.*

Proof: Since there must be an even number of vertices of odd degree in the minimal spanning tree M, we can use a minimum cost perfect matching beteween these vertices to complete an Eulerian graph. We know the minimum cost perfect matching has weight at most $\frac{1}{2}OPT$, thus $ALG \leq MST + MATCHING \leq \frac{3}{2}OPT$. We have a $\frac{3}{2}$-approximation for metric TSP. ■

Note that we used two lower bounds in our approximation. Each lower bound on its own isn't fantastic but combining them together produced a more impressive result. As an exercise, find a family of instances for which there is a big gap between the matching lower bound and the optimal TSP tour. This simple approximation algorithm has been known for over 30 years [1] and no improvements have been made since its discovery.

1.6 Steiner Tree

Next time we will talk about the Steiner tree problem.

Definition 1.6.1 (Steiner tree problem) *Given a weighted graph $G = (V, E)$ and a subset $T \subseteq V$ of nodes in a graph, find a least cost tree connecting all of the nodes in T (the "helper nodes" that are in the graph but not in in T are known as Steiner nodes).*

Applications of Steiner trees arise in computer networks. We will discuss this problem next lecture.

References

[1] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. In *Technical report no. 388, Graduate School of Industrial Administration, Carnegie-Mellon University*, 1976.