CMSC 451: Lecture 22
Approximation Algorithms: Vertex Cover and TSP

Tuesday, Dec 5, 2017

Reading: Section 9.2 of DPV. Section 11.3 of KT presents a different approximation algorithm for Vertex Cover.

Coping with NP-completeness: With NP-completeness we have seen that there are many important optimization problems that are likely to be quite hard to solve exactly. Since these are important problems, we cannot simply give up at this point, since people do need solutions to these problems. How do we cope with NP-completeness:

Brute-force search: This is usually only a viable option for small input sizes (e.g., $n \leq 20$).

Heuristics: This is a strategy for producing a valid solution, but may be there no guarantee on how close it is to optimal.

General Search Algorithms: There are a number of very powerful techniques for solving general combinatorial optimization problems. These go under various names such as *branch-and-bound*, *Metropolis-Hastings*, *simulated annealing*, and *genetic algorithms*. The performance of these approaches varies considerably from one problem to problem and instance to instance. But in some cases they can perform quite well.

Approximation Algorithms: This is an algorithm that runs in polynomial time (ideally), and produces a solution that is guaranteed to be within some factor of the optimum solution.

Approximation Bounds: Most NP-complete problems have been stated as decision problems for theoretical reasons. However underlying most of these problems is a natural optimization problem. For example, find the vertex cover of *minimum* size, the independent set of *maximum* size, color a graph with the *minimum* number of colors, or find the traveling salesman tour with the *shortest* cost. An approximation algorithm is one that returns a valid (or feasible) answer, but not necessarily one of the optimal size/weight/cost.

How do we measure how good an approximation algorithm is? We define the *approximation ratio* of an approximation algorithm as the worst-case ratio between the answer produced by the approximation algorithm and the optimum solution. For minimization problems (where the approximate solution will usually be larger than the optimum) this is expressed as approx/opt, and for maximization problems (where the approximate solution will be smaller than the optimum) this is expressed as opt/approx. Thus, in either case, the ratio will be at least 1, and the smaller the better.

Although NP-complete problems are equivalent with respect to whether they can be solved exactly in polynomial time in the worst case, their approximability varies considerably. Here are some possibilities:

- Some NP-complete problems are *inapproximable* in the sense no polynomial time algorithm achieves a ratio bound smaller than ∞ unless P = NP. (For example, there is no bound on the approximability of either independent set or graph coloring, unless P = NP.)

- Some NP-complete problems can be approximated, but the ratio bound is a *function of n*. (For example, the set cover problem, can be approximated to within a factor of $O(\log n)$, and this is believed to be the best possible.)
- Some NP-complete problems can be approximated and the ratio bound is a *constant*. (For example, we will see below that Vertex Cover can be approximated to within a factor of 2.)
- Some NP-complete problems can be approximated *arbitrarily well*. In particular, the user provides a parameter $\varepsilon > 0$ and the algorithm achieves a ratio bound of $(1 + \varepsilon)$. Of course, as ε approaches 0 the algorithm's running time gets worse. If such an algorithm runs in polynomial time for any fixed ε, it is called a *polynomial time approximation scheme*, or PTAS. (For example, there is a PTAS for the Euclidean traveling salesman problem.)

Vertex Cover: We begin by showing that there is an approximation algorithm for vertex cover with a ratio bound of 2, that is, this algorithm will be guaranteed to find a vertex cover whose size is at most twice that of the optimum. Recall that a vertex cover is a subset of vertices such that every edge in the graph is incident to at least one of these vertices. The *vertex cover optimization problem* is to find a vertex cover of minimum size (See Fig. 1).

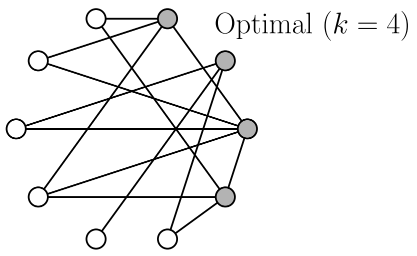

Fig. 1: Vertex cover (optimal solution).

How does one go about finding an approximation algorithm. The first approach is to try something that seems like a "reasonably" good strategy, a *heuristic*. It turns out that many simple heuristics, when not optimal, can often be proved to be close to optimal.

Here is an very simple algorithm, that guarantees an approximation within a factor of 2 for the vertex cover problem. First, we compute a maximal matching for the graph. Recall that a *matching* for a graph $G = (V, E)$ is a subset of edges $M \subseteq E$ such that each vertex is incident to at most one edge of M. A matching is *maximal* (not necessarily *maximum*) if it is not possible to add any more edges to the matching.

To compute a maximal matching, repeatedly find an unmarked edge (u, v), add it to the matching, and then mark all the edges incident to either u or v. Clearly, the result is a matching, and it cannot be extended to a larger matching, so it is maximal. To obtain a vertex cover, observe that for every edge (u, v) an *any* matching, either u or v must be included in *any* vertex cover. Thus, a simple (and rather dumb) solution is to add *both* of them to the vertex cover. (The algorithm shown in the following code fragment, and it is illustrated in Fig. 2.)

VC approximation via Maximal Matching

```
VC-approx(V,E) {
  C = emptyset
  mark all edges as uncovered
  while (there exists an unmarked edge (u,v)) do {  // (u,v) is in a maximal matching
    add both u and v to C                           // add u and v to the cover
    mark all the edges incident to u and to v       // mark all incident edges
  }
  return C as the approximate vertex cover
}
```

Claim: Above algorithm achieves an approximation ratio of 2.

Proof: Let M be the set of edges (u, v) identified by the algorithm, and let C be the set of endpoints of M. Let $k = |M|$. Since M is a matching, $|C| = 2k$. Because M is maximal, every edge in G is incident to some endpoint in C, implying that C is a vertex cover of size $2k$. By definition, any vertex cover must contain at least one endpoint from each of M's edges. Since no two edges of M share the same endpoint, any vertex cover has size at least $|M| = k$. Thus, C is within a factor of 2 of the optimum size.

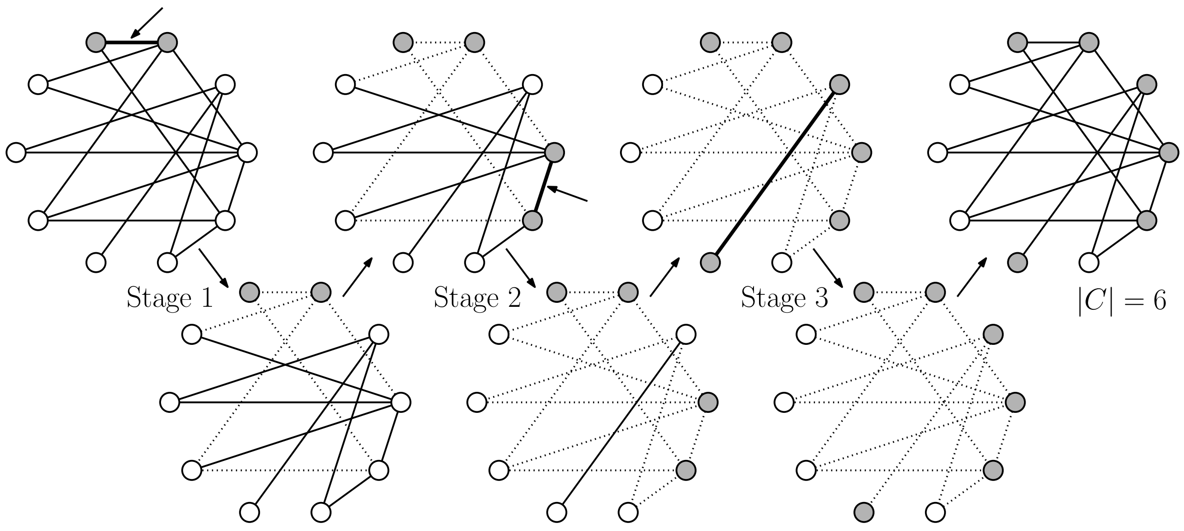

Fig. 2: The matching-based heuristic for vertex cover.

This proof illustrates one of the main features of the analysis of any approximation algorithm. Namely, that we need some way of finding a bound on the optimal solution. (For minimization problems we want a lower bound, for maximization problems an upper bound.) The bound should be related to something that we can compute in polynomial time. In this case, the bound is related to the set of edges A, which form a maximal independent set of edges.

Traveling Salesman with Triangle Inequality: In the Traveling Salesperson Problem (TSP) we are given a complete undirected graph with nonnegative edge weights, and we want to find a cycle that visits all vertices and is of minimum cost. Let $w(u, v)$ denote the weight on edge (u, v). Given a set of edges A forming a tour we define $W(A)$ to be the sum of edge weights in A.

For many of the applications of TSP, the edge weights satisfy a natural property called the *triangle inequality.* Intuitively, this says that the direct path from u to x, is never longer than an indirect path. More formally, for all $u, v, x \in V$

$$w(u, v) \;\le\; w(u, x) + w(x, v).$$

There are many examples of graphs that satisfy the triangle inequality. For example, given any weighted graph, if we define $w(u, v)$ to be the shortest path length between u and v, then it will satisfy the triangle inequality. Another example is if we are given a set of points in the plane, and define a complete graph on these points, where $w(u, v)$ is defined to be the Euclidean distance between these points, then the triangle inequality is also satisfied.

When the underlying cost function satisfies the triangle inequality there is an approximation algorithm for TSP with a ratio-bound of 2. (In fact, there is a slightly more complex version of this algorithm that has a ratio bound of 1.5, but we will not discuss it.) Thus, although this algorithm does not produce an optimal tour, the tour that it produces cannot be worse than twice the cost of the optimal tour.

The key insight is to observe that a TSP with one edge removed is just a spanning tree. However it is not necessarily a minimum spanning tree. Therefore, the cost of the minimum TSP tour is at least as large as the cost of the MST. We can compute MST's efficiently, using, for example, Kruskal's algorithm. If we can find some way to convert the MST into a TSP tour while increasing its cost by at most a constant factor, then we will have an approximation for TSP. We shall see that if the edge weights satisfy the triangle inequality, then this is possible.

Here is how the algorithm works. Given any free tree there is a tour of the tree called a *twice around tour* that traverses the edges of the tree twice, once in each direction (see Fig. 3).

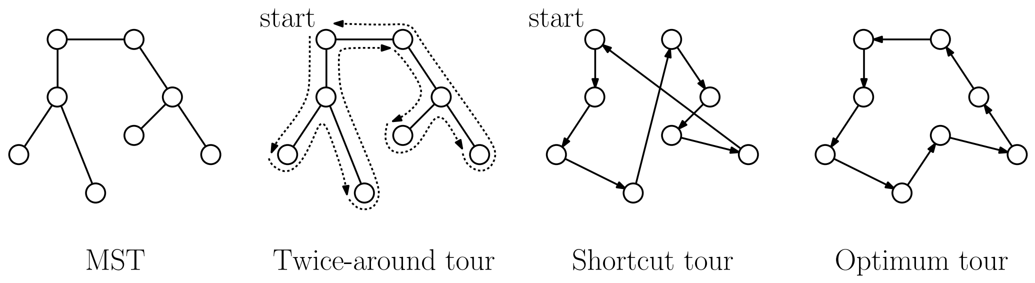

Fig. 3: TSP Approximation.

This path is not simple because it revisits vertices, but we can make it simple by *short-cutting*, that is, we skip over previously visited vertices. Notice that the final order in which vertices are visited using the short-cuts is exactly the same as a preorder traversal of the MST. (In fact, any subsequence of the twice-around tour which visits each vertex exactly once will suffice.) The triangle inequality assures us that the path length will not increase when we take short-cuts.

Claim: Approx-TSP achieves a approximation ratio of 2.

TSP Approximation

```
approx-TSP(G=(V,E)) {
    T = minimum spanning tree for G
    r = any vertex
    H = list of vertices visited by a preorder walk of T starting at r
    return L
}
```

Proof: Let H denote the tour produced by this algorithm and let H^* be the optimum tour. Let T be the minimum spanning tree. As we said before, since we can remove any edge of H^* resulting in a spanning tree, and since T is the minimum cost spanning tree we have

$$W(T) \;\leq\; W(H^*).$$

Now observe that the twice-around tour of T has cost $2 \cdot W(T)$, since every edge in T is hit twice. By the triangle inequality, whenever we short-cut an edge of T to form H we do not increase the cost of the tour, and so we have

$$W(H) \;\leq\; 2 \cdot W(T).$$

Combining these we have

$$W(H) \;\leq\; 2 \cdot W(T) \;\leq\; 2 \cdot W(H^*) \quad \Rightarrow \quad \frac{W(H)}{W(H^*)} \;\leq\; 2,$$

as desired.