CMPSCI 575/MATH 513

Combinatorics and Graph Theory

Lecture #9: The Traveling Salesperson Problem
(Tucker Section 3.3)
David Mix Barrington
26 September 2016

The Traveling Salesperson Problem

- The Problem and its Variants
- NP-Completeness of TSP
- The Branch and Bound Algorithm
- An Example
- Approximating TSP
- Tucker's Approximation Algorithm
- Proof that it Approximates

TSP and its Variants

- A **weighted graph** is a graph with a non-negative number assigned to each edge, its **cost**. A **weighted directed graph** has a cost for each directed edge.
- The cost of a path is the sum of the costs of its edges. Costs might represent distance, time, money, or anything we might want less of.
- The **traveling salesperson problem** is to find the *least-cost Hamilton circuit* in a weighted graph.

Representing Weighted Graphs

- We can represent a weighted graph by a matrix, where the (i, j) entry is the cost of the edge from node i to node j.
- If there is no edge, the entry is ∞. More typically, all the edges exist, though we still put ∞ on the diagonal entries to forbid loops.
- In a weighted ordinary graph, the matrix is symmetric.

Variants of TSP

- Along with directed and undirected versions, we sometimes require that the costs satisfy the **triangle inequality**, that $d(i, j) \leq d(i, k) + d(k, j)$ for any three nodes i, j, and k.
- The **Euclidean TSP problem** has nodes that are points on the plane, and costs that are Euclidean distance.
- We might also insist the graph be planar.

NP-Completeness of TSP

- TSP is one of the most famous NP-complete problems. As we discussed earlier, this means that there is probably no way to solve it in time polynomial in n, the number of nodes.
- The Hamilton circuit problem is NP-complete (see Sipser's book for an elegant proof). It's easy to reduce Hamilton circuit to TSP, by converting the input graph to a weighted graph such that the Hamilton circuit in the former is optimal in the latter if it exists.

The Branch & Bound Algorithm

- Tucker presents an algorithm to find an optimal TSP in any weighted directed graph. Of course it doesn't scale polynomially, but it works reasonably well on small instances, better than an exhaustive search.
- It's our first example of a general idea, the **branch and bound** algorithm.
- We have a tree of possible partial solutions to the problem, and we're going to search that tree in an intelligent way.

Branch and Bound for TSP

- With TSP, a partial solution is a set of edges we commit to use, and a set we commit not to use, in our tour. Call this a **scenario**.
- In each scenario, we will have a modified version of the cost matrix, and a **lower bound** on the cost of any tour in that scenario.
- Once we have this, we choose an edge and **branch** into two new scenarios, based on whether that new edge is forced in or out of the tour.

Row and Column Bounds

- Consider the n by n cost matrix. Suppose one row has all positive entries, the smallest being x. Since any tour will use exactly one edge out of that node, and hence one entry in the row, x is a lower bound on the tour cost.
- If we subtract x from every entry in the row, we get a new matrix. The optimal tour for the new weighted graph is the same as for the old one, with the cost reduced by x.
- The same trick works for columns.

More Branch and Bound

- We can make a tree of our scenarios, and expand the tree at each step by looking at a node with the smallest available bound.
- Eventually our modified matrix has a cost-0 set of edges, meaning that the original graph has a tour with cost equal to its bound.
- We save time in our search by not considering nodes whose bound is higher than the cost of the actual optimal tour.

Branch & Bound Example

- Starting with the top matrix, we subtract 3, 3, 5, and 4 from the four rows to get the second matrix, finding a lower bound of 15.
- Then we can subtract 1 from the last column, to the the third matrix which has a 0 in each row and each column. The lower bound is now 16. Now we need to branch.

```
∞ 3 9 7
3 ∞ 6 5
5 6 ∞ 6
9 7 4 ∞

∞ 0 6 4
0 ∞ 3 2
0 1 ∞ 1
5 3 0 ∞

∞ 0 6 3
0 ∞ 3 1
0 1 ∞ 0
5 3 0 ∞
```

Branch & Bound Example

- We pick a zero entry, say the second entry in the first row, corresponding to edge (1,2).
- If we don't use that edge, we get the second matrix, from which we can improve the lower bound to 20 by subtracting 3 from the first row and 1 from the second column.
- We can achieve this bound with tour 1-4-3-2, but this won't matter.

```
∞ 0 6 3
0 ∞ 3 1
0 1 ∞ 0
5 3 0 ∞

∞ ∞ 6 3
0 ∞ 3 1
0 1 ∞ 0
5 3 0 ∞

∞ ∞ 3 0
0 ∞ 3 1
0 0 ∞ 0
5 2 0 ∞
```

Branch & Bound Example

- If we instead rule edge (1,2) *into* our tour, we delete the first row and second column to get a new matrix with rows for 2, 3, 4 and columns for 1, 3, 4.
- We make the (2,1) entry ∞ to avoid the cycle 1-2-1, and subtract 1 from the new first row to make the lower bound 17.
- We can achieve this bound with tour 2-4-3-1, and this is the optimal tour.

$$\begin{matrix} \infty & 0 & 6 & 3 \\ 0 & \infty & 3 & 1 \\ 0 & 1 & \infty & 0 \\ 5 & 3 & 0 & \infty \end{matrix}$$

$$\begin{matrix} 0 & 3 & 1 \\ 0 & \infty & 0 \\ 5 & 0 & \infty \end{matrix}$$

$$\begin{matrix} \infty & 2 & 0 \\ 0 & \infty & 0 \\ 5 & 0 & \infty \end{matrix}$$

Approximating TSP

- Rather than spend exponential time to get the best tour, we can spend polynomial time to get a *pretty good* tour. This is one of the most widely studied approaches to NP-complete problems, as seen in CS 311.
- Tucker presents a fast algorithm that will always achieve a tour with cost at most twice the optimum, in the case of a symmetric matrix that satisfies the triangle inequality.

Approximating TSP

- Other algorithms approximate the optimum very well for Euclidean TSP, getting within a few percent of the optimum for graphs with millions of points.
- Many "novel approaches" to NP-complete problems fail to compete well with these.
- We'll revisit this algorithm soon when we discuss minimum spanning trees.

The Book's Algorithm

- The algorithm is simple. We create a "tour" of a single node at any vertex. Then we successively add new vertices to the tour until it is a Hamilton tour.
- At each step we have a cycle C_n on some of the nodes. We find the cheapest edge from any vertex y_n in C_n to a vertex z_n not in it.
- We then replace the edge out of y_n in C_n with a two-edge path through z_n.

Proof that it Approximates

- The challenge will be to prove the the tour we form in this way has at most twice the cost of the optimal tour.
- Let C^* be the optimal tour, and let S_1 be the Hamilton path formed by deleting the most costly edge from C^*.
- We'll make successive sets of edges $S_2, \ldots, S_n$ by deleting one edge each time, so $S_n = \emptyset$.
- When we delete an edge of cost c from S_i, we'll ensure $\text{cost}(C_{i+1}) \leq \text{cost}(C_i) + 2c$.

Proof that it Approximates

- The total cost of the edges in S_1 is less than the cost of C^*, so the cost of our tour C_n must be less than twice that cost.
- Consider what happens when we alter C_i to C_{i+1} by adding a cheap edge e into node z_n, adding an edge f of unknown cost out of z_n, and removing an edge g formerly in the cycle.
- By the triangle inequality, cost(f) $\leq$ cost(e) + cost(g), so what we add is at most $2 \cdot$cost(e).
- And e was chosen to be cheapest of a set that includes the edge we removed from S_i.