Traveling Salesperson (21)

1 TRAVELING SALESPERSON PROBLEM

1.1 Drilling Holes

Let’s say you are manufacturing printed-circuit boards. Each board needs to have a set of holes drilled in it that the components will later be inserted into.

Your automated driller drills a hole, then moves to the next location and repeats. Once all holes are drilled, the circuit board continues down the assembly line and the next one appears.

Time spent drilling is a constant. Want to minimize the time spent in transit between holes! Example...

1.2 Traveling Salesperson Problem

This is an example of the classic traveling salesperson problem.

Formally, we are given a set of \( n \) nodes \( V \) (points, vertices, cities, holes), and a distance function \( D : V \times V \to \mathbb{R} \) giving travel time between any given pair of nodes.

We want to find a tour (Hamiltonian circuit, permutation) of the nodes \( v_1, \ldots, v_n \) such that \( \sum_{i=1}^{n-1} D(v_i, v_{i+1}) + D(v_n, v_1) \) is minimized. It’s the shortest trip that visits every city once and only once.

We’ll assume the distance function \( D \) is symmetric: \( D(u,v) = D(v,u) \) for all \( u \) and \( v \) in \( V \).

In the metric TSP, we additionally assume that distances obey the triangle inequality: for all \( u \) and \( v \) and \( w \) in \( V \), \( D(u,v) \leq D(u,w) + D(w,v) \) (shortcut is never worse).

1.3 Hamiltonian Circuit

Before we go after the full-blown TSP, let’s look at a related simpler problem.

Given a graph \( G = (V,E) \), a Hamiltonian circuit (vertex tour) is a non-repeating, all-inclusive sequence of nodes in the graph that form a path.

The problem HAM is the decision problem: Does \( G \) have a Hamiltonian circuit?

Is HAM in NP?

1.4 Hardness of HAM

HAM is NP-complete. First, we’ll show that deciding if a directed graph has a Hamiltonian circuit is NP-hard. We’ll reduce 3-CNF-SAT to it.
Idea: We have a widget for each clause and an assignment-selection gadget that makes us choose Boolean values for each of the variables one at a time. We have the option of satisfying a clause and returning to variable selection. We must satisfy all clauses and then complete the loop.

Picture...

\[
x \quad v \text{ with loop}
\]

\[
\begin{array}{c}
\circ \longrightarrow \circ \\
\circ \rightarrow x \rightarrow x \rightarrow y \rightarrow y \rightarrow y \rightarrow y \\
\circ \rightarrow \circ \\
\circ \\
\text{not } x
\end{array}
\]

Now, we can turn any undirected instance into a directed instance by expanding each node into a trio of nodes: for incoming edges, outgoing edges, and a middle node to force us to go from in to out. Why do we need the middle node?

Picture...

### 1.5 Linking HAM and TSP

Take the decision problem: Given a TSP instance, is the optimal tour less than some given size \(C\)? It is in NP...

We can reduce HAM to this. Given an \(n\)-node graph \(G = (V, E)\), we generate a TSP instance by using the same set of nodes and setting \(D(u, v) = 1\) if \((u, v) \in E\) and \(D(u, v) = 2\) otherwise.

This TSP has a tour of size \(n\) if and only if \(G\) has a Hamiltonian circuit (otherwise it is at least \(n + 1\)). So, if we could solve TSP in polynomial time, we could solve HAM in polynomial time, and P would equal NP.

### 1.6 Dealing with NP-Completeness

What do we do now? TSP is NP-hard. So, we’d be real surprised if we could actually solve this both optimally and efficiently in the worst case.

But, we still need to solve the problem. People are depending on us! What can we do? Lower our standards!

- Find an approximation quickly.
- Find an optimal solution as quickly as you can (given that we’ll get worst-case exponential-time performance).
- Try to find a solution quickly that isn’t too awful (“empirical miracle”).

2
2 APPROXIMATION

2.1 Approximation Idea

We want an efficient algorithm, just like we’ve had all along. Ok, such an algorithm won’t be able to find the best tour, but maybe we can guarantee ourselves to find something that’s close to being optimal. Idea: Let’s say the cost of the truly optimal tour is OPT, but we find a tour of length C. The ratio $C/OPT \geq 1$ tells us how close we are to being optimal (the closer to 1, the better). We’ll give a fast algorithm that produces tours for metric TSP with a worst-case ratio of 2. So, we’re never further than a factor of 2 away from optimal!

2.2 Hardness Results

<table>
<thead>
<tr>
<th>ratio of 1</th>
<th>TSP</th>
<th>metric TSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>ratio of 2</td>
<td>NP-hard</td>
<td>polynomial</td>
</tr>
</tbody>
</table>

Proved the hardness results for exact metric TSP already (since $2 \leq 1 + 1$). The hardness for a ratio of 2 come from replacing the distance 2 edges with distance 2n edges (violating the “metric” property). Then, if there is a Hamiltonian circuit, the approximation algorithm would have to give us a tour of length no more than 2n. Such a tour could only consist of the length 1 edges and, therefore, would indicate the existence of a Hamiltonian circuit in the original graph.

2.3 Approximation Algorithm

Here’s what we do. Given a metric TSP instance with distance function $D$:

- 1. Take $D$ and treat it as the weight function for a weighted graph. Compute a minimum spanning tree (MST) of the graph. Let’s say its cost is $M$.
- 2. Pick a node in the MST and begin walking along the edges in an inorder traversal. Make sure to skip over nodes we’ve already visited.
- 3. Return the list of visited nodes as the tour.

2.4 Running Time

We compute an MST on $n^2$ edges: $O(n^2 \log n)$. We then walk the tree, avoiding duplicates: $O(n)$. Total: $O(n^2 \log n)$ (not too bad).
2.5 Analysis of Solution Quality

This is the interesting part. How does the tour we find relate to the optimal tour? Recall OPT is the length of the best tour, $M$ is the size of the MST we found, and $C$ is the length of the tour we return. Facts:

- $M \leq \text{OPT}$. Tour spans nodes. So, $2M \leq 2\text{OPT}$
- $C \leq 2M$. Touring the tree is $2M$, shortcuts only make it smaller.
- Therefore, $C \leq 2\text{OPT}$! (Ratio of 2.)

So, even though we don’t know OPT, we know it can’t be too far away!

2.6 Other Algorithms

Other approximation algorithms have been proposed.

- One starts with an MST and then uses a matching algorithm to add some edges to the tree from which we can build a good tour. Ratio is $3/2$ and good tours are found in practice.
- For TSP in 2-d, there is a dynamic-programming algorithm that exploits the sequence structure of the tour to find approximations with arbitrarily good ratios (the ratio shows up in the exponent in the running time).

3 OPTIMAL APPROACHES

3.1 Idea

If we really need the absolute best tour, approximation algorithms are not the way to go. Instead, we will need to, in essence, check all tours. We can do this in enumerating all permutations of the nodes. Running time?

$$n!.$$ 

More clever schemes can actually get this down to $O(2^n)$.

3.2 Partial-Tour Graph

Let’s say a partial tour is a graph with no degree-three nodes and no “premature” loops. That is, it is a subgraph of a tour. We can view the space of all partial tours as a big DAG where the edges connect a partial tour to another partial tour that includes one additional edge. The weight of a partial-tour-graph edge is the weights of the added edge in the graph. The empty graph is a partial tour with no incoming partial-tour-graph edges. What is the out degree of a partial-tour-graph node corresponding to a complete tour? What is the length of any path from the empty graph to a complete tour?
Complete tour has zero outdegree. All paths from the empty graph to a complete tour have \( n \) steps.

3.3 Shortest Path

Now, the problem has become one of doing a single-source shortest path search, which terminates when a complete tour is reached.

We can’t actually construct the entire graph, but we can generate it on an as-needed basis. Dijkstra’s algorithm will only visit as many partial tours as have weight less than or equal to OPT. But, that’s still pretty inefficient.

3.4 Heuristic Search

Recall that A* search gives us a more guided way to search the graph looking for a shortest path.

Needs an admissible heuristic to get going... that’s something that gives us a lower bound on the distance to the target node.

Here, a target node is any complete tour.
What gives us a lower bound on how the partial tour can be extended to form a complete tour? MST!
I actually have no idea if this would be useful... I did see it mentioned in an AI textbook, though.

4 HEURISTIC APPROACHES

4.1 Motivation

We’ve talked about getting a nearly optimal tour fast and getting a truly optimal tour slowly. What people actually do in practice is get a not-entirely-stupid tour in not-too-long time. Heuristics.

Let your imagination run wild! Ants, rubber bands, neurons, DNA computers, you name it!

4.2 Nearest Neighbor

Idea: Start at a node. Find closest unvisited node. Visit it. Repeat until finished, then return to start.
Runs fast.
Worst-case ratio of about \( 1/2 \log n \) (not so good).
Not too bad in practice.
Running time?

At each step, we need to find the nearest unvisited node. Keep a table of which nodes have been visited. Loop over the unvisited nodes and keep the min in \( O(n) \).
Move and update the table. Total is \( O(n^2) \).
4.3 Greedy

Kind of like MST algorithm, but works with partial tours.
Start with empty partial tour. Add smallest edge that results in a valid partial tour. Repeat
until complete tour reached.
Worst-case ratio no worse than $1/2 \log n$, maybe better.
In practice, performs better than MST-based approximation scheme.
Running time?

Can sort all edges in $O(n^2 \log n)$. How test quickly for valid partial tour? Can keep
the standard Kruskal data structure to say which nodes are linked to which others.
We also keep track of the number of edges coming out of each node. We're ok as
long as we don't connect two nodes that already have a path between them (except
for the final edge) and we don't put an edge on a vertex that already has two. Like
Kruskal's algorithm, this second phase takes $O(n \log n)$, so $O(n^2 \log n)$ is the total.

4.4 Local Search

A more popular approach is local search: we start off with a valid tour and then use local
moves to improve the tour.
Basic hill-climbing algorithm:

- Pick a tour (perhaps using the MST approximation).
- While there is some local move that improves the tour, make it.

Terminates at a “local minimum.” Kind of like a “minimal” tour in that the only way to
improve it is to do something drastic.

4.5 Some Local Moves

The $k$-OPT class of moves takes the tour and removes $k$ edges from it, then reglues the
pieces in some other way.

- 2-OPT: flips a half tour. Pretty fast, can get badly stuck.
- 3-OPT: shuffles thirds. Slower, less prone to sticking.

Picture...
Theoretically, even in metric TSPs, ratio between $1/4 \sqrt{n}$ and $4\sqrt{n}$ in the worst case.
Furthermore, it can take $O(2^{n/2})$ moves to get there!
Good performance in practice, though.

6
4.6 Other Approaches

In practice, more sophisticated search strategies are used.

- Kernigan-Lin was the champ from 1973 to about 1989. It is a variation of 2-OPT that uses a technique to move away from local minima to get unstuck.
- Many “neural net” approaches have been advocated.
- Some of the best techniques now are based on genetic search.

Lots of cool variations.
5 CONCLUSION

5.1 Today’s Lecture

Fitting last class, draws on much of what we’ve learned:

- Analysis: Critical for understanding why the problem is so difficult in the first place!
- Sorting: Used in Kruskal’s algorithm.
- Statistics: Useful for comparing algorithms empirically.
- Heap: Used in Dijkstra’s algorithm.
- Binary Search Trees: splay trees provide efficient implementation of local-search heuristics.
- Hashing: can map node labels to identifiers.
- Matching: Important approximation method.
- Matrix Algorithms: linear programming (which builds on solving systems of linear equations) used extensively in generating lower bounds.
- Graphs: Special type of graph problem, tree walk is a DFS.
- Minimum Spanning Tree: Simple approximation method.
- Shortest Paths: Search the graph of partial tours.
- DP: New approximation schemes for 2-d TSP.
- Complexity: NP-complete.

5.2 The Course

Take Home Lessons:

- Algorithms matter
- Some classic algorithms and their analyses
- How to analyze algorithms
- How to invent new algorithms
- Empirical evaluation of algorithms
5.3 Thanks!

I had fun and learned stuff. I hope you did, too!
Thanks to the UTAs (Mike, Matt, Jimmy, Spence, Iris, Geoff) and TA (Cheng) for helping makes things run smoothly.

6 PRACTICE PROBLEMS

- Explain how to use a solution to the bipartite matching decision problem to find a maximum bipartite matching of a graph. Hint: First use a binary search to find the size of the maximum matching, then use self-reducibility to find the set of edges that attains this value.

- CLR Exercise 36.4-7 (pg. 946).

- CLR Exercise 36.5-2 (pg. 960).

- Why are those middle nodes needed in the reduction from undirected HAM to directed HAM? What would happen if they were omitted?

- Argue that, in non-metric TSPs, for every ratio $r > 1$, there is no algorithm that can efficiently produce a tour within a ratio bound of $r$ unless P=NP.