Day 18 Notes

Zach Neveu

June 5, 2019

1 Agenda

- Dynamic TSP Review
- Local Search

2 Dynamic TSP

- Review concept of Hamiltonian paths $c(\boldsymbol{S}, k)$
- To optimize $c(\boldsymbol{S}, k)$, try all possible last nodes on the path before k, and choose minimum.
- $c(\boldsymbol{S}, k) = argmin_k[c(s - \{k\}, m) + w[m, k]]$
- $c(\{2\}, 2 = w[1, 2]$ base case. Cheapest HP with one node is just weight of single path.
- $c(\{4, 2\}, 2) = c(\{4\}, 4) + w[4, 2]$
- $c(\{2, 4, 6\}, 2) = min_k(c(\{4, 6\}, 4) + w[4, 2], c(\{4, 6\}, 6) + w[6, 2])$
- Once all costs are generated from 1 to all other nodes, just add paths from last node back to 1 and minimize.
- Problem: $O(n_2^n)$ entries, each computation is $O(n)$, so complexity is $O(n^2 2^n)$
- Exhaustive algorithm requires $O(n!)$
- 2^n smaller than $n!$, so we actually have a good improvement!
- Difference is, exhaustive algorithm re-solves all subproblems every time.
- **quiz q idea**: write top down/bottom up algorithm to solve TSP this way

3 Local Search

- "Blunt tool" that works almost everywhere
- Usually a heuristic, not exact
- Really dang good results sometimes
- Key idea: Good solutions lie in similar areas. If you have a good solution, try tweaking it a little to see if it gets better.
- Consider optimization problems in form (F, c) where F is feasible set and c is cost function

- Given a **neighborhood function** which maps each element of F onto a subset $f \in F$
- Given F, $N(t)$, is set of feasible points "near" t
- **Improvement function**: `improve(t)` returns any neighbor with lower cost, or none if none exist.

```
# Generic Local Search Algorithm
t = rand(F) # random point in F
while improve(t) is not None:
    t = improve(t)
return t
```

- Details to Fill in:
  1. How to select initial solution?
  2. How is the neighborhood defined?
  3. How to select a neighbor?
  4. What to do when there are no better neighbors? (exploration vs. exploitation dilemma)
- No single right answer, but many good options

Initial Solutions (TSP as example)

- Assume TSP on fully connected graph
- Nearest Neighbor (NN): At every step, visit closest neighbor that has not already been visited.
  - Euclidean instances: 26% from HK bound - pretty good!
  - Non-euclidean instances: 130-410% from HK bound - not so good
- Greedy Algorithm (GD): TSP version of spanning tree. Keep adding cheapest edge, so long as it doesn't create a node of degree 3, or a cycle.
  - Euglidean: 14%-20% of HKB
  - Non-Euclidean: 100%-280% of HKB

Insertion Algorithms

- All start with small tour, and add nodes such that it is always expanding
- Nearest Addition
  - Given tour, find closest node not on tour(k) to a node on the tour (j)
  - Select a neighbor, i, of j.
  - Delete delete i,j, add j,k and k,i
- Cheapest Insertion

  - Similar to nearest addition, but minimize cost of j,k+k,i instead of just j,k
  - Select node that minimizes c[i,k]+c[k,j]-c[i,j]
  - Slower than nearest addition, but logically would be more accurate
- Farthest Insertion
  - Select node furthest from tour, then use cheapest insertion to find where to add it
  - Euclidean: 16% from HKB

Other 2 Algorithms (category name?)

- MST vs TSP
  - Deleting one edge from HC gives spanning tree!
  - Cheapest HC must be more expensive than MST.
  - MST is lower bound on HC cost
  - Possible to adjust edges of MST to create HC?
  - Traverse tree by always visiting an unvisited neighbor. Backtrack when hit a dead end.
  - When backtracking, don't keep track of nodes that have already been visited.
  - On example graph, go IHGEGHFDADFCFBFI, but don't record the backwards sections
- Eulerian Cycle Approach
  - Eulerian cycle visits each edge once.
  - Eulerian cycle exists if each node of the graph has even degree - trivial to compute
  - Given MST, find all nodes with odd degree
  - Find minimum weighted matching among these odd degree nodes and add edges that are found
  - Now an Eulerian cycle must exist!
  - Find this, and convert it to a HC.

Experimental Methodology

- How to compare algorithm IRL?
- TSPLIB - large, standard collection of hard TSP instances.
- Multiple reasons TSP can be hard
  - For instances from real maps, a->b always shorter than or equal to a->c->b **triangle inequality**

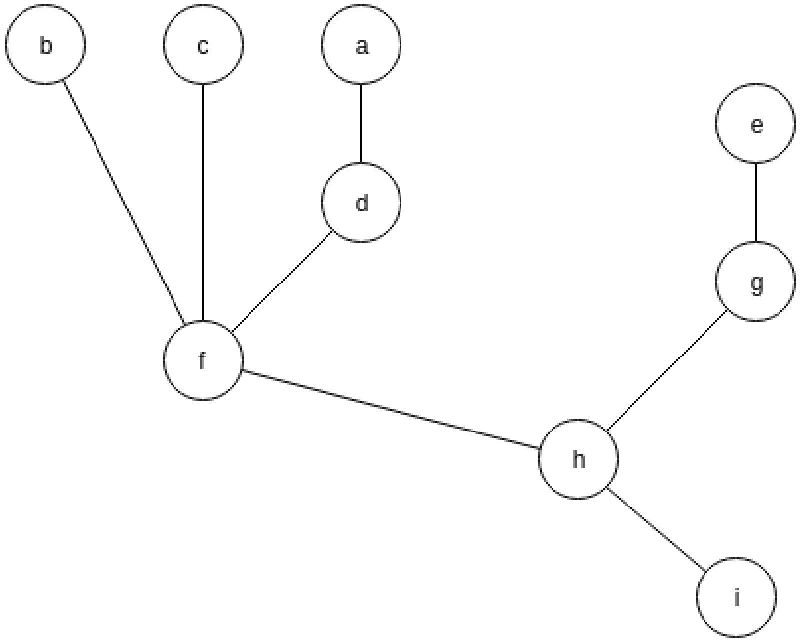

Figure 1: Example Graph

  - Random graphs will not generally satisfy this
  - Triangle inequality helps both heuristics and speed of optimal solutions

- TSPLIB separated into “euclidean"->triangle inequality holds and “non-euclidean"->triangle inequality doesn’t hold
- Using bounds, it is possible to get an upper limit on the difference between a heuristic value and the optimal solution.
- Express TSP in ILP, then use LP bound. Unfortunately, this is long, so approximates used -> Held-Karp bound.