Day 19 Notes

Zach Neveu

June 6, 2019

1 Agenda

- More intro to local search
- Neighborhoods
- Tabu search
- Reading Assigned
- Project due Monday
- Quiz Tuesday

2 Initial Solutions for Local Search

- Eulerian-cycle based algorithm (Christofides Algorithm)
- Gets pretty close to HKB

3 Neighborhood Functions for TSP

- A neighborhood function: select two edges to delete, then add edges to reconnect the cycle
- Maps from one solution to a similar solution
- Gets one neighbor. To get all neighbors, run this function on all pairs of edges.
- $\binom{e}{2}$ total neighbors. Neighborhood called **2-opt** neighborhood
- If current solution better than all 2-opt neighbors, it is a **2-optimal solution**

4 Local Search Algorithms

- Init algorithm
  - Random
  - Nearest Neighbor
  - Greedy

- Steepest descent: Look at 2-opt neighborhood, best neighbor becomes current solution until solution is 2-optimal
- Simple 2-opt helps answer so much!
- 3-Opt: select 3 edges to delete, then add edges to reconnect the cycle. $O(n^3)$ ways to disconnect, 6 ways to put a broken cycle back together. MUCH bigger neighborhood. $O(n^3)$ work for a single step of improvement.
- 2-opt greedy: 4.9%, 3-opt greedy: 3%. Incremental improvement, but MUCH slower to run.
- Local optima problem: These algorithms get permanently stuck in local optima.
- Simplest solution: Repeatedly select random initialization and optimize from each starting point with 2-opt steepest descent.
- Not a particularly a good use of time in practice.
- Revised steepest descent: go to best neighbor, even if it's worse than current solution
- This frequently gets caught in cycles. To fix, just don't go back to already visited solutions
- Tabu list can contain a list of previously visited solutions
- In Tabu search, we select a best neighbor that is not on the tabu list, even if it is worse than current solution. Keep track of champion solution. Stop based on time, time since champion found, improvement in sequential champions small etc.
- Tabu list can actually be quite short, as few as ≈ 10 entries.

Table 1: Initialization Performance

Algorithm	Initial Euclidean	2-Opt	Non-Euclidean Initial	2-Opt
Random	2150	7.9	34500	290
Nearest Neighbor	25	6.6	240	96
Greedy	17.6	4.9	170	70

5 Uniform Graph Partitioning

- Given a graph $G = (V, E)$ with an even number of nodes n, and weighted edges, divide the graph into two groups of nodes with equal size such that the weight of the edges crossing the boundary is minimized.
- Imagine there are two processors. Each node is a job, and the goal is to split the jobs onto the two processors such that each processor has the same amount of work and the amount of data sent between the processors is minimized.
- Neighborhood function: Select a node from each partition and swap them. Equivalent to 2-opt in TSP.
- What to put in the tabu list?

  - Complete solutions. Obvious, and eliminates all cycles with length < 10. Huge amount of into in the queue.
  - Nodes that were recently moved. Prevents going directly backwards. Moves that are made must be left for awhile. Problem with this is it's really restrictive, eliminating possibly valid and good moves. More space efficient than full solutions by far.
  - Store direct pairs of nodes on tabu list. Each node from pair is eligible to be moved again, just not with each other. This is space efficient, and less restrictive. Cycles possible using intermediate variables.

Extensions to Tabu search

- Aspiration-level conditions: ignore the tabu list if a tabu neighbor would be a new champion solution.
- **Intensification**: Focus on solutions that are very similar to previous good solutions. Say a lot of past champions have very similar characteristics. Try freezing these characteristics and focus on changing everything else.
- **Diversification:** Focus on producing unique solutions different from all past solutions.
- Solvers tend to go through phases of diversification then intensification and alternate

Effective Problem Modeling

- Graph Coloring - Minimize # Colors
- Neighborhood function: change the color of one node
- This neighborhood function can have illegal neighbors, and so can also run into the problem of having no legal neighbors.
- Almost all neighbors will have same objective value. We're in Kansas.
- Solution is to change the problem. Instead of minimizing # colors, minimize # conflicts for given number.