NPTSP Project

Name: Max Lam | Section: 109
Partners: Maya Reddy, Sneha Sankavaram, Natasha Sandy
Team Name:LMMNRSSS | Login-Submitted-With:abcd

May 4, 2015

Contents

1 Main Idea

Two stages

- Matlab integer linear programming to outright solve 85% of the problems
- Simulated annealing to approximate the ones that couldn't be solved

2 Matlab integer linear programming

Matlab provides a function called 'intlinprog' which uses the branch&cut algorithm. The branch&cut algorithm works by using branch and bound, and linear programming to compute upper and lower bounds to the solution, which prunes nodes of the search tree. The Matlab guide for TSP was adopted for our project. The link is here: `http://www.mathworks.com/help/optim/ug/travelling-salesman-problem.html` The idea is that TSP can be formulated as a integer linear program such that every node has the constraint that it has exactly 2 edges coming out of it. Then we want binary integer indicators representing whether each edge should be included in the tour, for a minimum path. We modified the 'tour' by using a dummy node which has a distance of 0 to every other node. At the end, we remove the dummy node, thus creating a minimum path, instead of minimum tour. The way the tutorial handles subtours is it repeatedly adds constraints whenever it detects multiple subtours, then re-runs the integer linear program again and again. We adopt this method to detect invalid paths that traverse more than 3 cities with the same color consecutively. Thus, whenever we detect that the path violates the 'non-partisan' property, we re-run the integer linear program with the added constraints. So the overall process is as follows:

1. Formulate NPTSP as integer linear program
2. Run linear integer program until only 1 subtour, adding constraints if not satisfied
3. Check if satisfies 'non-partisan'. If so, exit with optimal path, else go to 2.

3 Simmulated Annealing

Very bare-bones approach. Uses the widely known simulated annealing approach. We have a temperature variable which indicates how likely we are to take a bad solution. So, generate a random valid partisan path, keep randomly modifying it, and take a bad solution sometimes to avoid local minima. Keep track of the best path so far, then at the end output it. So the process is as follows:

1. Generate random path
2. Loop with temperature
3. If temperature not the end temperature
   (a) Make random changes to the path
   (b) If new path is better than the old, replace it
   (c) If new path is worse than the old, replace with $e^{\text{-diffscore/temperature}}$ probability
   (d) Decrease temperature by factor c
   (e) Keep track of best path
   (f) Goto 3
4. Output best path

4 Input File Generation

We followed the 2-Opt algorithm and tried to generate cases that would be hard to approximate with it. This means clumping together nodes of the same color. Drew small cases by hand, then extended the cases to be 50x50 by adding random weights for the remaining edges

5 Running the Code

The code is separated into several files, some were experimental and don't quite fully work. The ones that don't quite work include the DP solution (which sometimes gives a partisan path) and the python approximation one. Therefore, we will mainly be focusing on the matlab code, and the C/C++ Simulated Annealing code, which was the only code we used to solve the inputs.

6 Matlab code

Run matlab, and within the matlab console cd into projbase/code/matlab.
Within the matlab console run TSP(inputfile,1,60).

- First argument is the input file
- Second argument is whether to debugoutput or not
- Third argument is time limit in seconds

7 Simulated annealing code

cd into projbase/code/SA.
run make
run ./a.out < inputfile

8 Resources

- Matlab & libraries
- Matlab TSP tutorial: `http://www.mathworks.com/help/optim/ug/travelling-salesman-problem.html`
- C/C++ stdlibs