James Fitzwater
CS 325.400
June 9, 2017

Project #3: Travelling Salesman Problem (TSP)

Researching TSP, I sought out educational texts that would allow me understanding rather than imitation when solving the project's problem-set, and, hopefully, help me limit my time translating a recommended or model approach from an unfamiliar language into one that I can more handily compose. My for academic resources prioritized those two components: primarily, presenting one or more approaches in clear-enough—limited technically—language and, secondarily, demonstrating this approach in Python-formatted instruction. Skimming a collection of algorithmic & data structures textbooks and Google Scholar, I turned again to a primary source I've learned to rely upon several times, *Machine Learning: An Algorithmic Perspective* by Stephen Marsland. From among the multiple solutions to the TSP offered, I selected the author's "Hill Climbing" solution to explore for this project.

Hill Climbing – Description

"Hill Climbing" constructs what one might consider an "optimization approximation" approach, which, while lacking the completeness of a Brute Force method, provides a reasonably good result in reasonable time. The structure begins with an arbitrary solution and randomized travel ordering of cities. More likely than not, any hill-climb run will start as a comparatively poor performer; but, over cycles, its randomized approach will predictably find improved solution results. Explained: with incrementation, a small element of the solution is altered, and if it the solution is better, it is kept. In the context of TSP, this may manifest itself by generating a random solution and then swapping pairs of cities in the tour at random, keeping the solution if the total length of the tour decreases. With enough loops, it is likely that a relatively short tour path will be obtained.

Similar in outcome to the Greedy methodology, the random nature of "Hill Climbing" does not guarantee in any way that an acceptably-optimal solution will result. Firstly, climbing expressly caps the number of sequential "runs" (or tests), meaning that the instruction set will not allow itself additional cycles if a sub-optimal result remains the best result (such an instruction can certainly be inserted, but that does not absolve the hands-off nature presented by the climbing methodology). Secondly, "Hill Climbing" risks trapping itself within a local min-max ceiling that is not approximate to optimal or desirous min-max.

Nevertheless, while this solution did not teeter as close to excessive cost as a more complete approach, such as Brute Force, my research did uncover a more promising method that I did not have the opportunity to play-around with more completely (and, presumably, implement). That method's name is "Simulated Annealing;" and it takes the structure of "Hill Climbing" but adds a constraint method called "cooling down," which allows for fine-tuning the instruction set to better fit the characteristics of the data input.

Hill Climbing – Pseudocode

```
function climb( listOne ):
      repeat increment thru numCity's len
            distOne += dist(listOne[x], listOne[y])
      assign listTwo, copy of listOne
      reorder listTwo's array elements

      repeat increment thru numTests' len
            do,
                  assign cCity random listTwo element
                  assign dCity random listTwo element
            while compare cCity != dCity
            reassign, swap cCity, dCity w/ adjacent elements
            repeat increment thru listTwo's len
```

```
                distTwo += dist(listTwo[x], listTwo[y])
            compare if distTwo < distOne
                reassign distOne's val w/ distTwo's
                reassign listOne's entries w/ listTwo's

    return listOne & distOne   ## most-efficient thru numTests
```

Hill Climbing – Results, Run-times

Example Files:

tsp_example_1
- Dist: 152423
- Time: 1.54158616066

tsp_example_2
- Dist: 3181
- Time: 4.74730706215

tsp_example_3
- Dist: 7147976
- Time: 94.6820340157

Competition Files:

test-input-1
- Dist: 28089
- Time: 0.991125106812

test-input-2
- Dist: 48488
- Time: 1.9988451004

test-input-3
- Dist: 4250
- Time: 1.96098899841

test-input-4
- Dist: 172500
- Time: 3.96804094315

test-input-5
- Dist: 69000
- Time: 7.77233982086

test-input-6
- Dist: 350000
- Time: 15.9133279324

test-input-7
- Dist: 2530000
- Time: 39.508261919