Name: Finlay Boyle User-ID: chpf93

Algorithm A: Water-Flow Algorithm

Algorithm B: Ant Colony Optimisation

Description of enhancement of Algorithm A:

Water-flow uses 2-opt to alter the local solution of a flow to optimise it. For my enhancement I have decided to replace the 2-opt algorithm with the Lin-Kernighan heuristic. It is significantly more complicated, but it provides better solutions because it is a variable k-opt algorithm that builds on 2-opt. I also varied the parameters but ultimately the best results were obtained by those recommended in the paper (referenced within the code), so I have focussed on describing the Lin-Kernighan enhancement here.

When using the enhanced version, I was able to obtain better tours for the city sets 42, 48, 58, 175 and 180 with the same parameters for the water-flow algorithm. For city sets 12, 17, 21 and 26 following extensive applications of all my implemented algorithms I believe the near-optimal solutions were found already by the original water-flow implementation (and matched by the enhanced version). For the 535-city set the algorithm is quite slow, as such it was not able to improve the solution (only match what my original implementation had managed). think it is because the search space has become too large for my implementation and I would need to further direct the search (as suggested by Helsgaun's paper referenced in the code).

Overall, the enhancement was successful at finding improved tours for the majority of the city sets – even when the original waterflow and Lin-Kernighan on their own would stall.

Description of enhancement of Algorithm B:

My enhancement of the ACO was simpler than that of algorithm A. I implemented a Ranked Ant System. I chose to implement this because it is a well-known algorithm that gave me a solid benchmark for comparing all my other implemented solutions to. To implement this, I used the bisect module to essentially create a size-limit dictionary that is sorted by the "length" of the ant's tour as the ants complete their tour. This allowed for increased efficiency as I only stored a maximum of *w* edge lists and they were sorted as they were added to increase the speed of the implementation.

In terms of results (relative to the basic implementation) it tied on city sets 12, 17, 21 and 180 (I believe near-optimum solutions were already achieved as they agree with algorithm A). It improved on city sets 26, 42, 48, 58, 175 and 535.

I also varied the parameters to try and determine the best. Higher parameters appeared to work better for large city sets (as evidenced in the added_note in some of the submitted files).

Overall, this enhancement was successful at finding improved tours and performed significantly better on some sets than my basic ACO implementation with little to no change in running time.