Travelling Thief Problem

Evolutionary Computation

Prepared by William Reid, Matthew Hart, Samantha Peachey & Alec Bellati

School of Computer Science,
The University of Adelaide

October 5, 2014

Exercise 3

This exercise focusses on the modification of the tour that our thief takes, based upon a given set of items that he wants to steal. These items are supplied in a text file that is read in. Since this problem revolves around solving the traditional travelling salesman problem (with a few restrictions), evolutionary approaches should yield good results within the required time-frame.

1 Algorithm

This algorithm performs four fundamental steps:

1. Seed the knapsack with items using various methods
2. Seed a random tour, taking into account the packing plan
3. Evaluate solution
4. Mutate tour
    - A subset of the tour is selected to be scrambled.
    - This subset is randomly shuffed around, checking to ensure that the order specified by the packing plan is not disrupted.

Steps 3 and 4 are then repeated, generating new individuals and selecting the fittest individual to continue generating. Scrambling was chosen as the mutator of choice as it was found to be quite powerful during assignment 1.

1.1 Profit Evaluation

To evaluate the profit of the solution at each step, the evaluate function from the TTPInstance class was used. This ensures that the calculated profit uses the correct formulas and also reduces the code base. Research was done into faster evaluation methods and while it is possible to create a simpler (but less accurate) evaluation technique, the gains were not significant enough for the implementation of a new method.

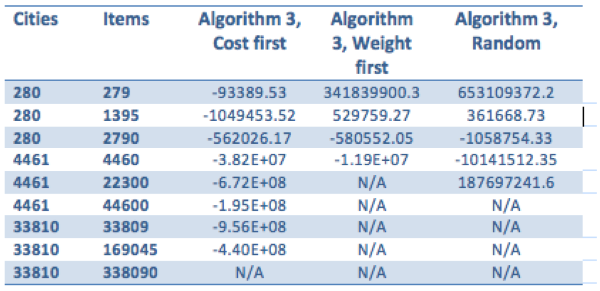
1.2 Useability on Problem Types

The primary bonus in the choice of this algorithm is it ensures it will behave in similar ways, no matter how the problem is presented. Whether it be strongly bounded, uncorrelated, single item per city, multiple items per city or thousands of cities - the algorithm will be consistent.

2 Testing, Additional Methods and Results

Some different methods were considered to try and better the result of the solution. This included different methods of seeding the original knapsack, and additional methods for modifying the TSP (such as swapping or inversion of cities within the tour). While these methods served as useful tests and concepts, they did not prove profitable within the algorithm.

The results of this algorithm were not as ideal as other approaches considered. This problem can be reduced to finding the shortest path through a graph, with required checkpoints at given stages. Given the limitations in regards to the items that need to be collected, this simply becomes a matter of minimising the rent paid and there are not many other variables that can influence this.



Cities	Items	Algorithm 3, Cost first	Algorithm 3, Weight first	Algorithm 3, Random
280	279	-93389.53	341839900.3	653109372.2
280	1395	-1049453.52	529759.27	361668.73
280	2790	-562026.17	-580552.05	-1058754.33
4461	4460	-3.82E+07	-1.19E+07	-10141512.35
4461	22300	-6.72E+08	N/A	187697241.6
4461	44600	-1.95E+08	N/A	N/A
33810	33809	-9.56E+08	N/A	N/A
33810	169045	-4.40E+08	N/A	N/A
33810	338090	N/A	N/A	N/A

Using the cost first approach to generating the knapsack, negative values were consistently produced. This approach was run using n = 10,000 generations (this was found to be the optimal number of generations in assignment 1), and the improvement over such a large number of generations was minimal.

The same can be said of the random knapsack generation, however over significantly smaller generation sizes (n = 10, n = 100), this approach saw a significantly better improvement from the initial starting position. The random knapsack generation approach also works much better than the cost first approach for both the bounded strong correlated and uncorrelated but similar weights problems, yet for the uncorrelated, dissimilar weights problems, the random knapsack seed was consistently worse.

It can be seen that the initial knapsack packing has a significant impact upon the performance of the travelling thief, and if this is suboptimal, there is not much that can be done to rectify the situation. It is also quite possible that there is a problem with my implementation, given the extremely negative figures, or in some instances, extremely high figures.