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 5

Each of the algorithms supplied in the previous 3 exercises all approach the problem in completely separate ways, ranging from simple linear searches to evolutionary algorithms base on complex heuristics. With additional time, these algorithms could be evolved to formulate even greater results - however the conclusions drawn from this assignment appear to point towards taking two separate approaches.

Using simple linear search methods and intelligent heuristics, we were able to find a "close to optimal" packing of the knapsack. Once found, evolutionary methods are used to find the best TSP for the knapsack provided. An attempt to do this in a similar manner by splitting the problem in half, and optimising one half for a TSP solution and the other for an optimal knapsack solution yielded poor results. Taking the entire solution in two separate stages may yield even greater results.

The best algorithm formulated was that of the Obsessive Packing. Its simple implementation finds the best packing of the knapsack, given a particular tour. Our methods also enable the algorithm to first quickly find an optimal tour using methods created in assignment one. This functionality has been turned off for the purposes of this exercise but can be added if desired.

1 Test Parameters

The following parameters were found to be close to optimal to achieve good results. Definitions for the categories are:

- **Knapsack Seed:** The algorithm allows for various methods to seed the knapsack. The "seed" is the starting packing of the knapsack and can greatly affect the final result. Seed 2 chooses items based on their profit-to-weight ratio, while seed 4 randomly picks items. (Explanations for a total of 6 seeds can be found in the *Will.java* file)

- **TSP Choice:** To account for extendibility, this algorithm not only considers just the packing of the knapsack, but may also consider changes to the TSP tour. Choice 1 simply uses the supplied linkern tour, where as choice 2 will use our best TSP algorithm from assignment 1.

- **Generations:** Number of times to perform the algorithm on the knapsack (knapsack packing is retained for the next generation, it is not re-seeded).

- **Iterations:** Number of times to replace a randomly chosen item in the knapsack with the current item. A value of "ALL" implies every item in the knapsack will be replaced to find the optimal position of the current item.

- **Items to be considered:** Number of items to be placed in the knapsack to evaluate. A value of "ALL" implies every item not currently contained within the knapsack will be considered.

Cities	Items	Knapsack Seed	TSP Choice	Generations	Iterations	Items to be considered
280	279	2	1	10	ALL	ALL
280	1395	2	1	10	ALL	ALL
280	2790	2	1	6	ALL	ALL
4461	4460	2	1	1	ALL	ALL
4461	22300	2	1	10	8	ALL
4461	44600	2	1	1	25	ALL
33810	33809	4	1	6	5	7500 Good 2500 Random
33810	169045	4	1	1	12	2000 Good 500 Random
33810	338090	4	1	1	50	2000 Good 500 Random

2 Results

2.1 Obsessive Packing

For each instance, the algorithms mentioned in the table below were run 20 times and the result was obtained. In addition the execution time for the entire 20 runs was recorded. The timer was independent of the Java classes and hence the minor fluctuations above the allocated 600 seconds (10 minutes) are due to the set-up and deallocation of the Java instances and do not imply a longer execution time of the algorithm.

Cities	Items	Hill Climber	Average Run-time (Seconds)	Obsessive Packing	Average Run-time (Seconds)	Max. Value
280	279	15 656.13	1.30	15 940.59	1.14	16 138.12
280	1395	104 363.98	3.10	104 365.19	139.32	104 365.73
280	2790	411 714.27	8.75	411 714.79	483.11	411 714.79
4461	4460	232 122.20	175.45	250 039.91	220.77	250 520.95
4461	22300	1 478 784.83	558.25	1 478 811.36	600.13	1 478 891.23
4461	44600	5 8423 56.39	600.15	6 027 685.27	558.77	6 030 747.53
33810	33809	1 541 800.98	599.90	1 575 701.95	600.23	1 591 115.89
33810	169045	-6 318 212.89	600.10	-3 235 085.31	600.30	-3 145 792.42
33810	338090	-48 261 363.83	600.40	2 830 880.21	600.64	3 735 362.06

2.2 Remaining Algorithms

Cities	Items	Matt	Sami	Alec
280	279	2 336.46		
280	1395	-260 949.38		
280	2790	-735 902.37		
4461	4460			
4461	22300			
4461	44600			
33810	33809			
33810	169045			
33810	338090			

3 Conclusion

The results shown in the previous section show that the Obsessive Packing algorithm can consistently obtain values equal to or higher then that of the *Hill Climber* algorithm. Due to its intrinsic design, smaller data sets take slightly longer (apart from a280_n279). However as the instances grow larger, the Obsessive Packing algorithm obtains a better result within the allocated time and if left running past the 10 minute limit, would also complete sooner than the *Hill Climber* algorithm.

The Obsessive Packing algorithm could be improved by considering changes to the TSP tour. Instances **a280_n1395** and **a280_n2790** appear to reach the optimal knapsack packing for the supplied tour, hence implying that modifying the TSP tour based on the knapsack may in-fact yield better results.