S1913403

16 – 3 - 2001

Report: Algorithm

The Traveling Salesman Problem (TSP) is a very famous combinatorial optimisation problem. It may appear simple, finding the shortest tour to visit a set of nodes, but it can actually get pretty complex. Humans, by estimation in a visual map, can manage to get decent solutions for small data (~10% of the optimum) but will rarely find the optimum solution. On the other hand, computers will struggle to find a more or less decent solution with a general algorithm, due to the data space of search's factorial growth.

The TSP has gotten quite famous in the scientific community, and many people have come up with different approaches to solve it. Apart from the given algorithms given in the worksheet, there are some more complex ones:

- Efficient heuristic procedures, e.g. Lin-Kernighan-Helsgaun; metaheuristics
- A bit more costly heuristics e.g. Ant Colony optimisation
- Exact solution algorithms, e.g. Branch-and-Cut
- And quite more

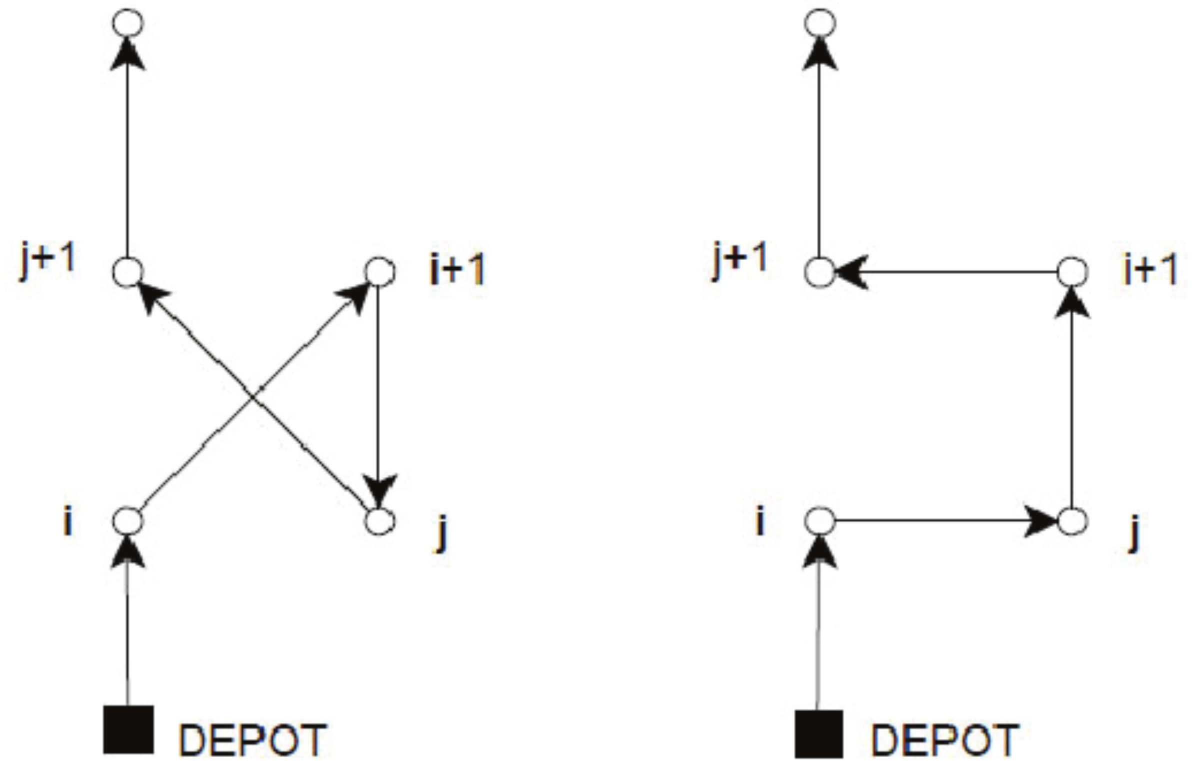
They algorithm that I chose to implement is the 3-opt. I chose it because it's one of the simplest most efficient algorithms in terms of quality of answer obtained and time spent. But let me tell you why:

I'll start explaining the generality of the λ-opt TSP operator. This operator works following the concept that, by swapping λ connections between cities, you can generate a reduction of the tour distance and thus improving the TSP path.

The good thing about this method is that for small amounts of nodes (‘n’), a very high λ can be chosen in order to achieve an almost optimal solution.

The downside is as follows. We can clearly see that the running time of the method grows with respect to lambda the following way: 2-opt is an O(n2), 3-opt is an O(n3), etc. Thus, for big amounts of ‘n’, larger values of λ are unusable if you want a reasonable running time.

Now for the specific case of the 2-opt consists in finding a pair of nodes (i and j) and we replace: (i,i+1) with (i , j) and (j,j+1) with (i+1,j+1).



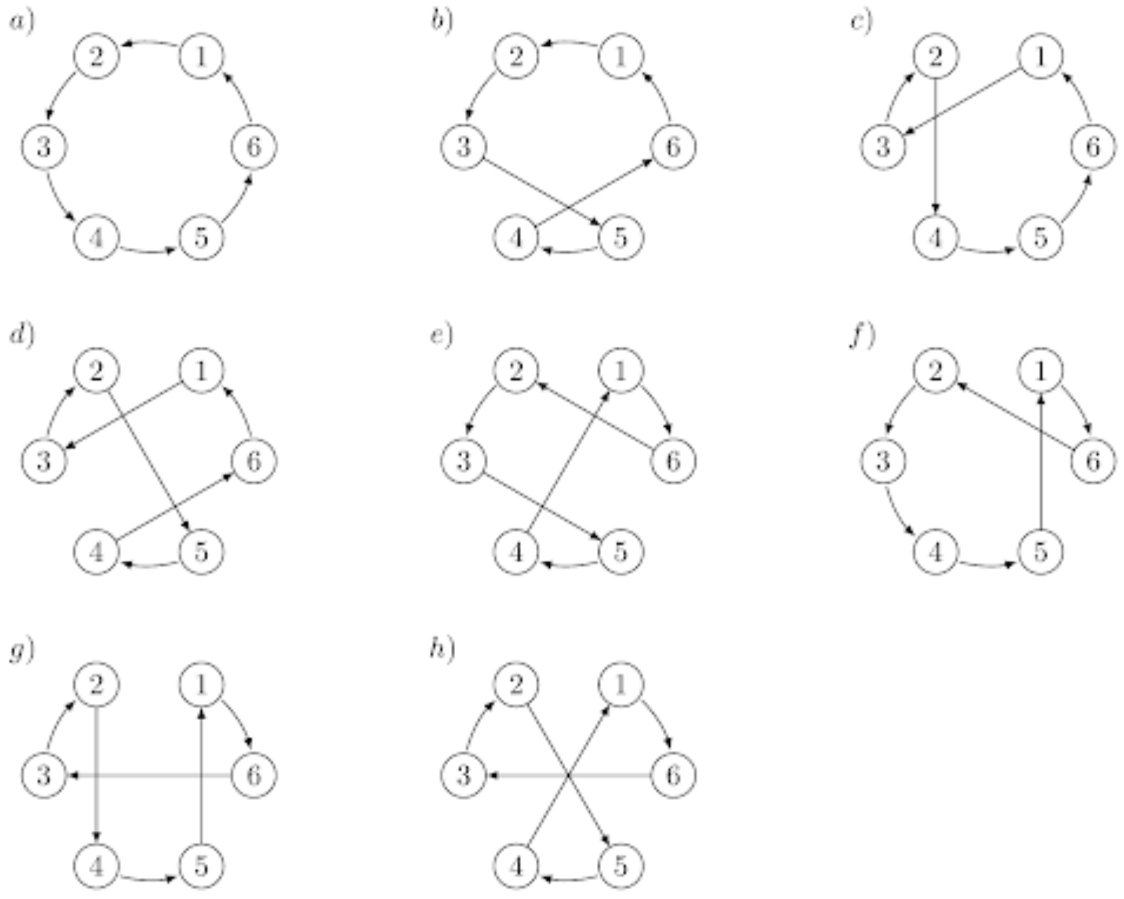
The gain offered by such a move is easily calculated as:

G = dists[i],[j] + dists[i+1][j+1] – dists[i][i+1] – dists[j][j+1]

If the gain is positive, we have an improving move.

Now 3-opt heuristic, is the logical next step of the 2-opt: it will relink three nodes, and this is better because some cases that cannot be optimised by the 2-opt algorithm can be optimised by the 3-opt

With three edges, there is more than one way we can re-arrange them while retaining a tour: we can rearrange them in eight different permutations (counting the base case):



The likeness of one of these seven moves improving the tour is quite high, and statistically speaking it will outperform 2-opt by up to 10% for node quantities bigger than 1000.