IADS Coursework 3: Heuristics for the Travelling Salesman Problem

March 26, 2020

C. Algorithm

Motivation

One of the simplest ways to construct an effective heuristic for TSP is to repeatedly apply small transformations to the current permutation and see how they affect the overall cost. Swap and 2-Opt heuristics are examples of this idea. My algorithm uses slightly more complicated transformations - it cuts the permutation in up to four places and considers combinations of putting the path back togetther. As only the ends of subpaths might get reconnected to different vertices, this transformation preserves most of the original solution.
Notes:
If improvement over a single iteration was less than 1%, we switch to more cuts so that we perform more complex transformations only if simpler ones fail. Also, in each iteration we return a solution immediately if it is significantly better.

Pseudocode

Data: perm
Result: best permutation found by the heuristic
$m_{opt} \leftarrow 1$;
for *iterations* $\leftarrow 0$ **to** $10n$ **do**
  improvement, perm $\leftarrow$ `MyHeuristicIteration(`m_{opt}`)`;
  if *improvement smaller than* 0.1% *of current solution cost* **then**
    increase m_{opt} by 1;
  else
    $m_{opt} \leftarrow 1$
end

Algorithm 1: Main method

Data: perm, m_{opt}
Result: best improvement found when cutting in m_{opt} places (one iteration)
best-tour $\leftarrow$ perm;
forall *indices* i, j, k, l *dividing perm* **do**
  intervals $\leftarrow$ *subpaths after cutting the solution in* i, j, k, l;
  forall reversed-subpaths $\leftarrow$ *possibilities of reversing some subpaths* **do**
    forall permuted-subpaths $\leftarrow$ *possibilities of permuting* reversed-subpaths **do**
      new-tour $\leftarrow$ `Concatenate(permuted-subpaths)`;
      if *improvement bigger than* 1% *of current solution cost* **then**
        return new-tour
      else if `TourValue(new-tour)` $<$ `TourValue(perm)` **then**
        best-tour $\leftarrow$ new-tour
    end
  end
  return new-tour
end

Algorithm 2: MyHeuristicIteration method

Time and memory complexity

Let n be the number of vertices in the graph.

Time complexity

First, we are doing $\Theta(n)$ iterations. Now, each iteration takes at most $O(n^5)$ time as for $m_{opt} = 4$, we need to iterate over all possibilites to cut the permutation in 4 places (there are $n * (n-1) * (n-2) * (n-3)/4!$ of them) and for all such cuts we do $\Theta(n)$ work (there is a constant number of permutations, constant number of reversing subpaths).
Therefore, the total time complexity is $O(n^6)$.

Memory complexity

The memory complexity is $O(n^2)$ as we need to maintain costs of edges and we do not need to maintain more than that at any time.

D. Experiments

Test	Swap	2-Opt	Greedy	My heuristic
30 points placed evenly on a circle of radius 1000, shuffled randomly	29025	6272	6272	6272
5 per circle with radii: 1k, 2k, 4k, 8k + 5 random, sorted by distance from the origin	104617	68729	75383	64982

1. Even though the points were shuffled, all but one heuristics managed to get the optimal solution (going along the circle). Therefore, this suggest that there are local minima which are significantly higher than the global minimum for swaps. The results for Swap and Two-Opt can be seen in Figure 1.

2. All other heuristics outperformed Swap again. From what I saw when I plotted the solution, it did not go stricly along the circles (and this state was a local minimum). My heuristic performed significantly better - I suspect this was due to the fact that it

3.

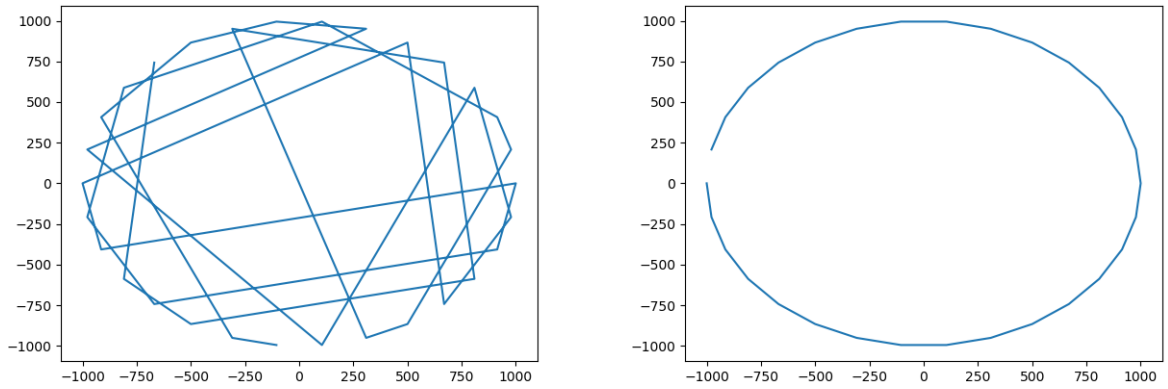

Figure 1: Swap and Two-Opt results for points placed evenly on a circle

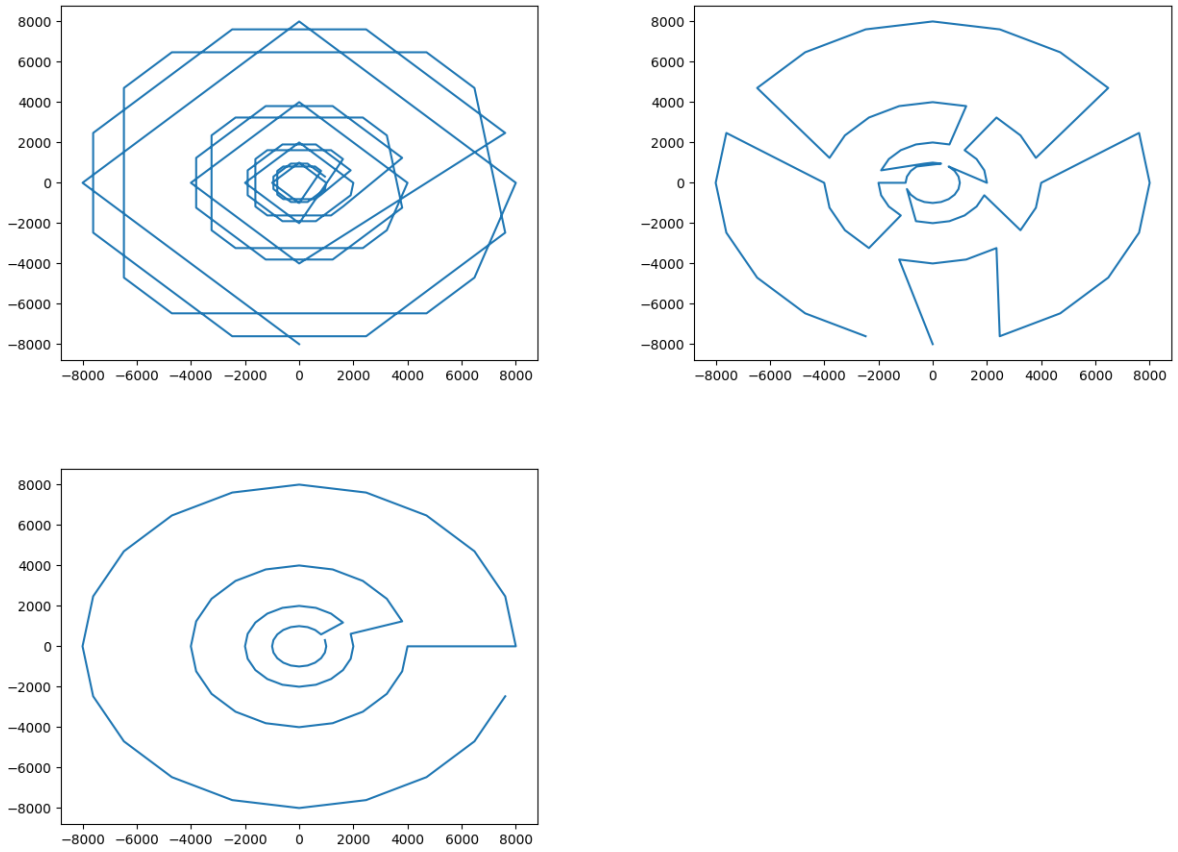

Figure 2: Results for points placed evenly on 4 circles with radii 1000, 2000, 4000, 8000