Evolutionary Computation Lab IV

Piotr Kaszubski 148283

Monday, November 13, 2023

Contents

1 Problem description

We are given three columns of integers with a row for each node. The first two columns contain `x` and `y` coordinates of the node positions in a plane. The third column contains node costs.

1. Select exactly 50% of the nodes (if the number of nodes is odd we round the number of nodes to be selected up).
2. Form a Hamiltonian cycle (closed path) through this set of nodes such that the sum of the total length of the path plus the total cost of the selected nodes is minimized. The distances between nodes are calculated as Euclidean distances rounded mathematically to integer values.

The distance matrix should be calculated just after reading an instance and then only the distance matrix (no nodes coordinates) should be accessed by optimization methods to allow instances defined only by distance matrices.

2 Pseudocode

The method is mostly the same as in the previous report. However, there is an extra preprocessing step and a slight alteration to the runtime logic.

2.1 Before linear search

1. Initialize an `NxN` boolean matrix `M` to `false` values, where `N` is the number of nodes in the graph `G`.
2. For each node `n` in `G`, do:
   (a) Find 10 nodes with the smallest distance to `n` (excluding `n`), and mark them as `true` in `M`.

Once we have the boolean matrix of 10 closest nodes to each node, we use it to cache all acceptable moves (of which there are 3 types: inter-route node swap, intra-route node swap, intra-route edge swap).
Inside the linear search algorithm, we modify:

1. For index `i` in `size(G)`, do:
   (a) For index `j` in `size(G)`, do:

    i. Evaluate intra-route node swap between `i`th and `j`th node.
    ii. Evaluate intra-route edge swap between `i`th and `j`th node.

2. For index `i` in `size(G)`, do:

    (a) For index `j` in `size(G')` (where `G'` is the set of nodes outside of `G`), do:

        i. Evaluate inter-route node swap between `i`th node in `G` and `j`th node in `G'`.

...to instead be:

1. For index `i` in `size(G)`, do:

    (a) For index `j` in `size(G)`, do:

        i. **If intra-route node swap for `(i, j)` is cached as acceptable, do**:
            A. Evaluate intra-route node swap between `i`th and `j`th node.
        ii. **If intra-route edge swap for `(i, j)` is cached as acceptable, do**:
            A. Evaluate intra-route edge swap between `i`th and `j`th node.

2. For index `i` in `size(G)`, do:

    (a) For index `j` in `size(G')` (where `G'` is the set of nodes outside of `G`), do:

        i. **If inter-route node swap for `(i, j)` is cached as acceptable, do**:
            A. Evaluate inter-route node swap between `i`th node in `G` and `j`th node in `G'`.

3 Results

ALG.	TSPA TSPB	TSPC TSPD
ls-steepest-random	77,866 (75,315–81,017) 71,322 (68,623–76,002)	51,371 (49,257–53,785) 48,234 (45,351–51,534)
lsc-steepest-random	89,197 (83,559–99,633) 84,469 (74,393–96,069)	62,674 (56,990–74,215) 60,289 (53,205–67,824)

Table 1: Average, minimum and maximum scores of found solutions

ALG.	TSPA TSPB	TSPC TSPD
ls-steepest-random	82.528 (70.451–117.119) 84.666 (73.417–98.120)	82.132 (72.866–96.202) 83.055 (68.014–94.368)
lsc-steepest-preset	19.134 (14.749–25.742) 18.863 (14.981–25.721)	18.321 (13.988–24.015) 17.899 (14.340–22.888)

Table 2: Average, minimum, maximum running times per instance (ms)

4 Visualizations

4.1 TSPA.csv

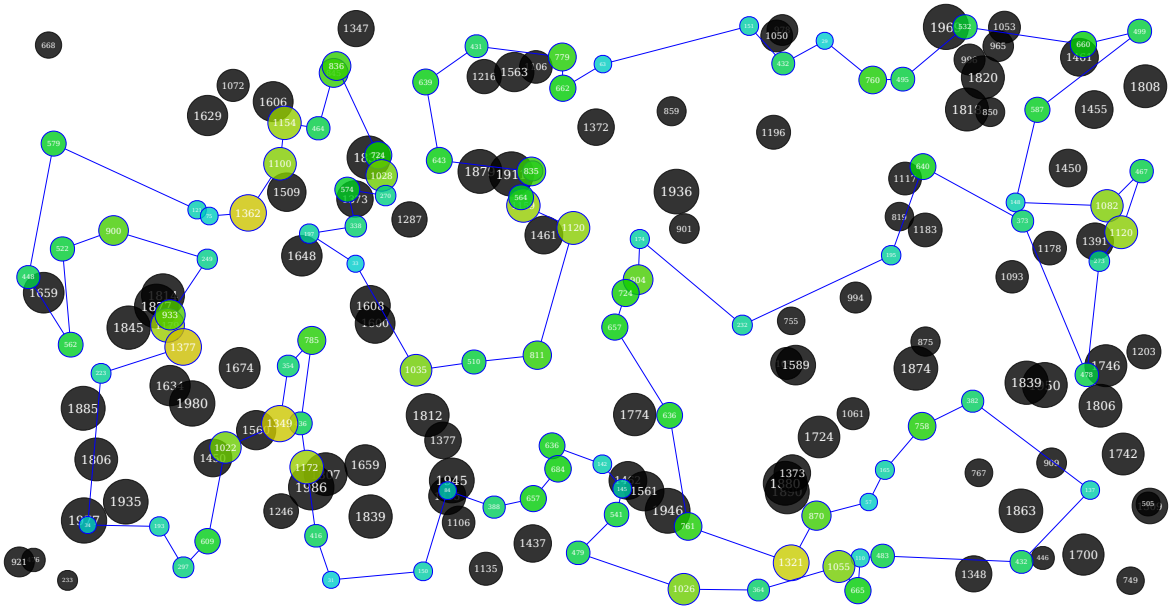

Figure 1: Best ls-steepest-random solution to TSPA (75,315)

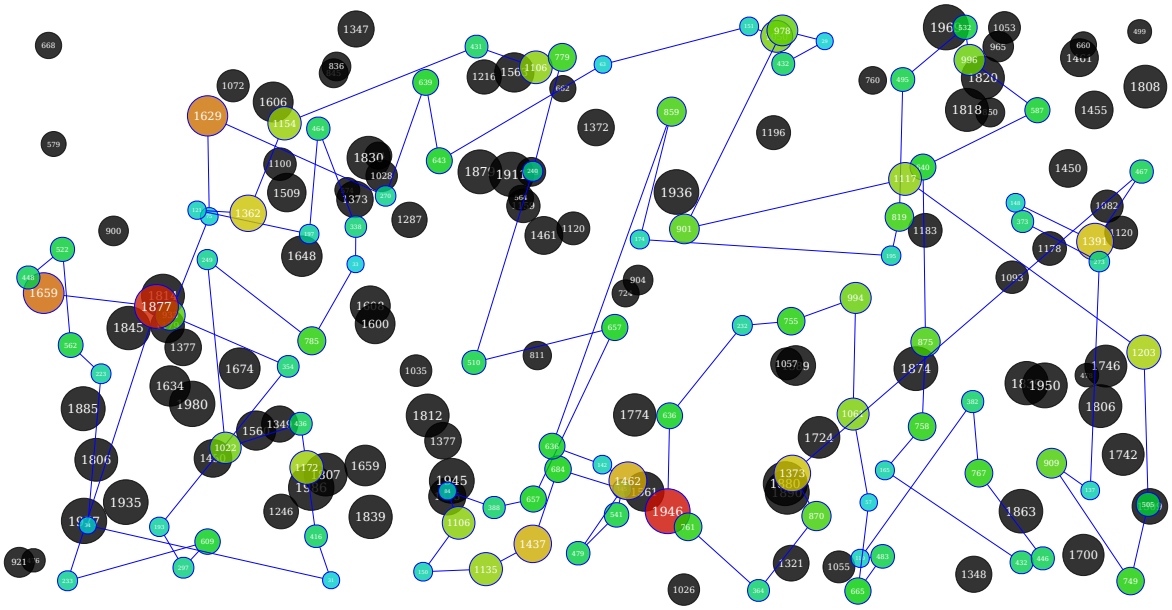

Figure 2: Best lsc-steepest-random solution to TSPA (83,559)

4.2 TSPB.csv

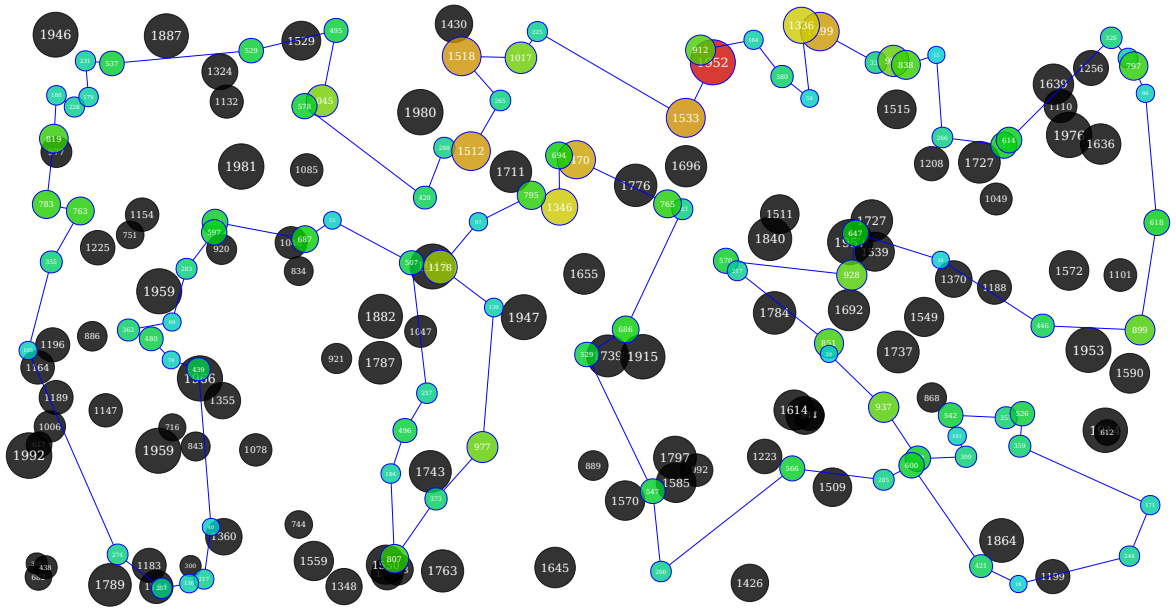

Figure 3: Best ls-steepest-random solution to TSPB (68,623)

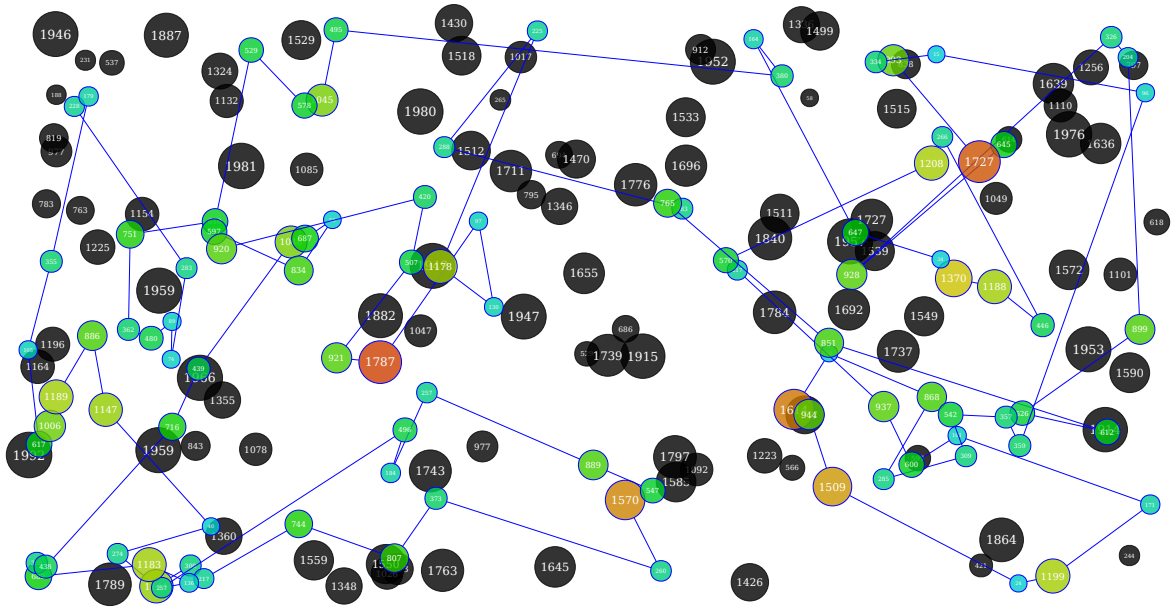

Figure 4: Best lsc-steepest-random solution to TSPB (74,393)

4.3 TSPC.csv

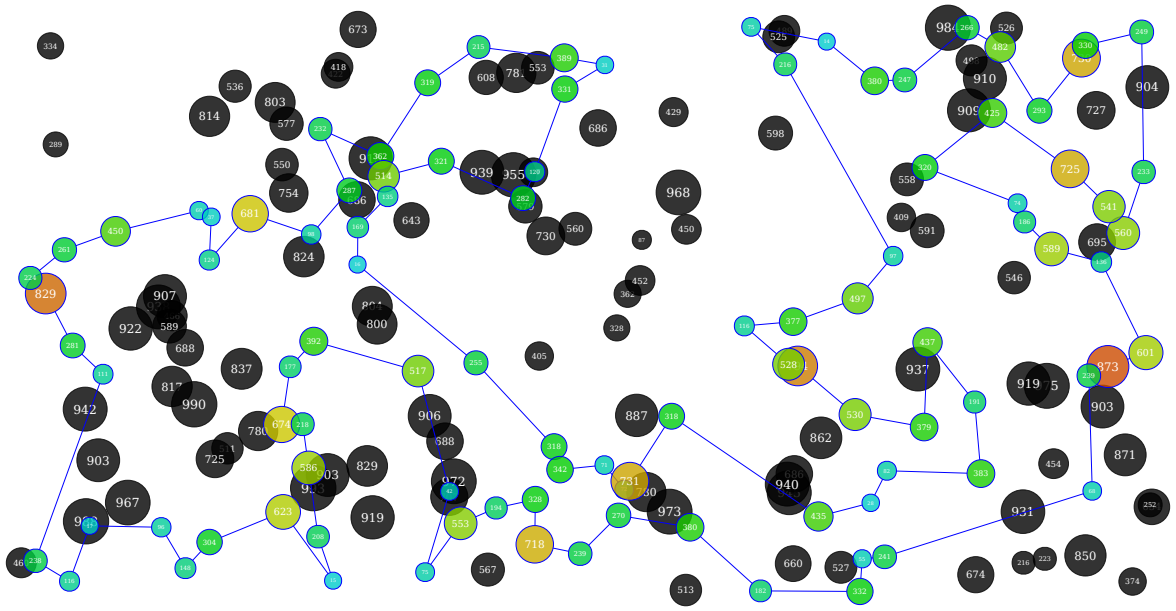

Figure 5: Best ls-steepest-random solution to TSPC (49,257)

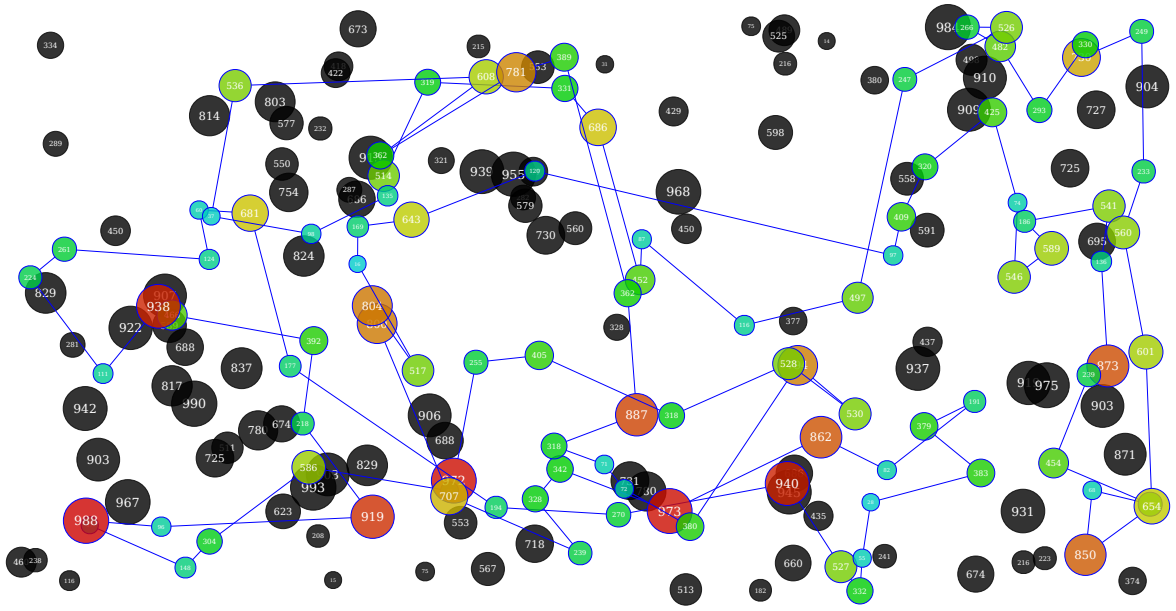

Figure 6: Best lsc-steepest-random solution to TSPC (56,990)

4.4 TSPD.csv

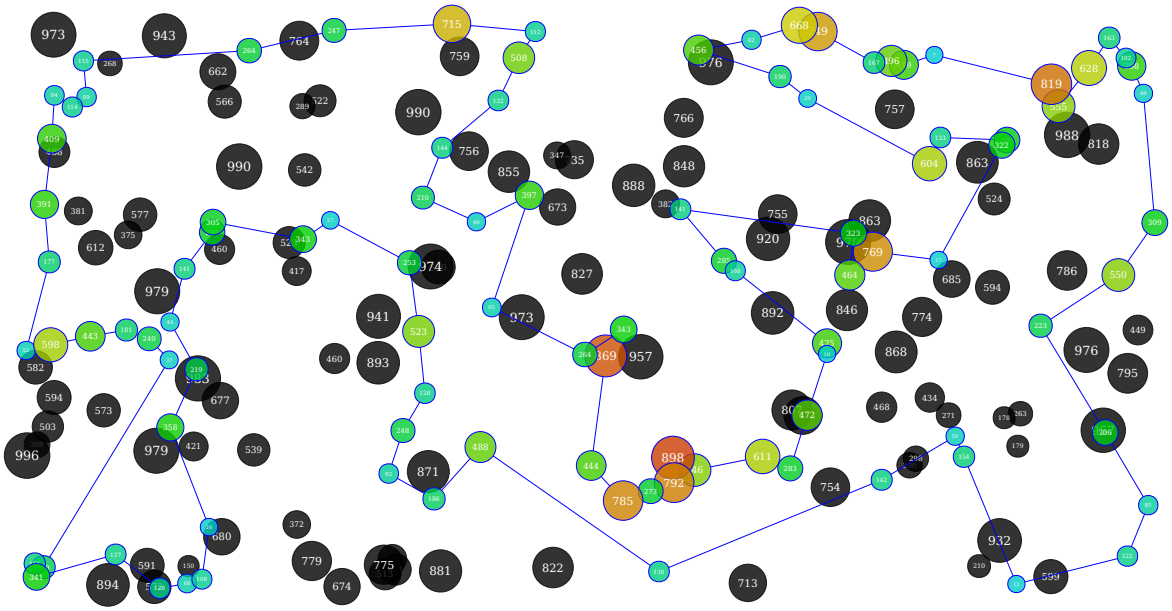

Figure 7: Best ls-steepest-random solution to TSPD (45,351)

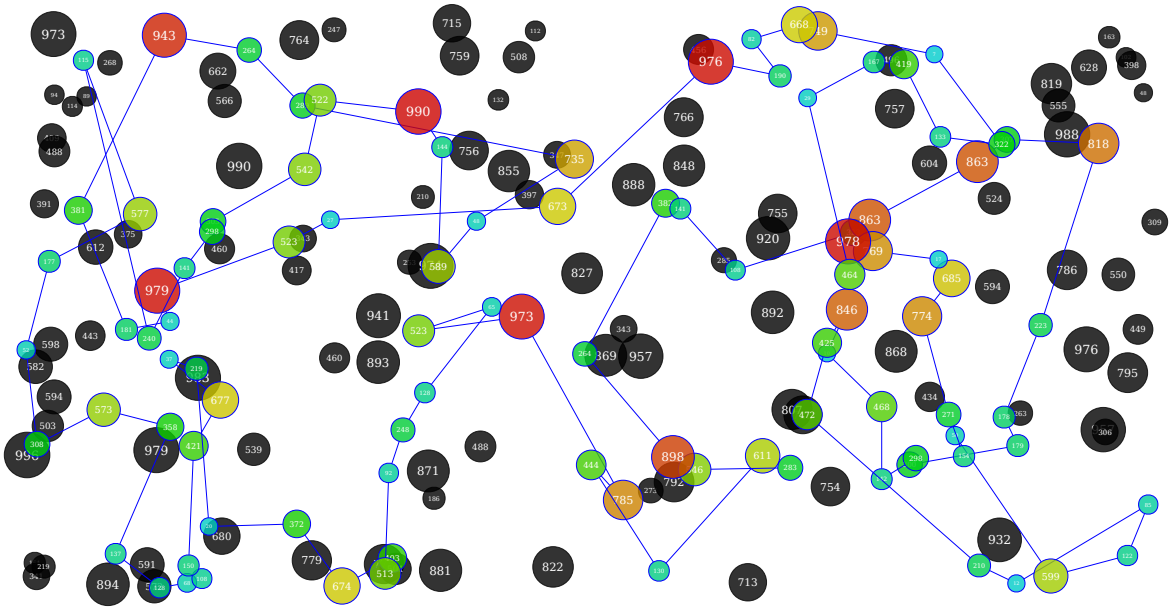

Figure 8: Best lsc-steepest-random solution to TSPD (53,205)

5 Source code

The source code for all the experiments and this report is hosted on GitHub: https://github.com/RoyalDonkey/put-ec-tasks

6 Conclusions

Candidate moves are a nice way to introduce a parameterized trade-off between the accuracy of the search and the running time. I tested the experiments with `n=15` as well, and it extended the time from around 15 seconds to 22 seconds, but managed to find solutions better by about 3,000.

It is visible from the plots that the candidate method settled for more high-cost nodes. I wonder if the tendency to pick longer edges is proportionally the same (it is difficult to tell from the plots, because edge lengths are not as easy to compare).