Evolutionary Computation Lab VII

Piotr Kaszubski 148283

Sunday, December 10, 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

2.1 Destroy-repair operator

The destroy operator is implemented as removing a random 30% of nodes in a graph.
The repair operator is *greedy cycle* from report 1.

3 Results

ALG.	TSPA TSPB	TSPC TSPD
ls-steepest-random	77,866 (75,315–81,017) 71,322 (68,623–76,002)	51,453 (49,257–53,785) 48,234 (45,351–51,534)
lsc-steepest-random	89,127 (82,350–101,152) 84,469 (74,393–96,069)	62,892 (55,074–74,956) 60,203 (51,897–67,824)
lsd-steepest-random	77,866 (75,315–81,017) 71,322 (68,623–76,002)	51,453 (49,257–53,785) 48,234 (45,351–51,534)
lscd-steepest-random	89,127 (82,350–101,152) 84,469 (74,393–96,069)	62,892 (55,074–74,956) 60,203 (51,897–67,824)

ALG.	TSPA TSPB	TSPC TSPD
msls-steepest-random	75,048 (74,178–75,644) 68,211 (67,828–68,622)	49,128 (48,311–49,620) 45,649 (45,075–46,032)
ils-steepest-random	74,255 (73,754–74,535) 67,389 (67,008–67,744)	48,312 (47,936–48,646) 44,616 (44,105–45,127)
$i_a$ls-steepest-random	74,893 (74,137–75,593) 68,070 (67,676–68,496)	48,969 (48,662–49,308) 45,505 (45,016–46,114)
lsns-random	73,873 (73,202–74,715) 67,991 (66,684–69,670)	48,384 (47,391–49,306) 45,644 (44,006–48,127)
lsns-lsearch	73,523 (72,966–74,238) 66,979 (66,659–67,512)	48,081 (47,434–48,968) 44,701 (43,833–46,319)

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

ALG.	TSPA TSPB	TSPC TSPD
ls-steepest-random	85.737 (71.796–105.250) 88.181 (73.757–107.447)	85.470 (72.226–98.981) 87.450 (67.483–102.099)
lsc-steepest-random	21.973 (17.070–28.766) 22.002 (17.069–29.329)	20.936 (15.504–30.904) 21.179 (15.651–28.530)
lsd-steepest-random	43.684 (33.639–66.647) 43.794 (35.071–54.709)	42.885 (33.038–57.912) 43.747 (30.598–74.864)
lscd-steepest-random	28.267 (18.670–39.157) 28.695 (19.072–38.583)	27.632 (18.752–39.882) 27.669 (19.578–40.003)

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

ALG.	TSPA TSPB	TSPC TSPD
msls-steepest-random	16.339 (15.744–16.838) 16.606 (16.074–17.191)	16.064 (15.530–16.725) 16.282 (15.941–16.693)
ils-steepest-random	16.327 (16.323–16.337) 16.328 (16.323–16.337)	16.328 (16.324–16.334) 16.328 (16.323–16.338)
$i_a$ls-steepest-random	16.335 (16.323–16.387) 16.333 (16.324–16.380)	16.342 (16.323–16.395) 16.342 (16.323–16.403)
lsns-random	16,323 (16,323–16,324) 16,323 (16,323–16,324)	16,323 (16,323–16,324) 16,323 (16,323–16,325)
lsns-lsearch	16,323 (16,323–16,324)	16,323 (16,323–16,324)

ALG.	TSPA TSPB	TSPC TSPD
	16,323 (16,323–16,324)	16,323 (16,323–16,324)

Table 3: Average, minimum, maximum running times for MSLS, ILS and LSNS (s)

ALG.	TSPA TSPB	TSPC TSPD
ils-steepest-random	1,868.5 (1,805–1,920) 1,774.8 (1,599–1,857)	1,663.0 (1,601–1,835) 1,642.5 (1,535–1,829)
$i_a$ls-steepest-random	1,149.3 (1,091–1,209) 1,117.7 (1,056–1,178)	1,207.8 (1,139–1,249) 1,150.0 (1,067–1,194)

Table 4: Average, minimum, maximum number of times linear search was run

ALG.	TSPA TSPB	TSPC TSPD
lsns-random	13,525.3 (13,138–13,967) 13,614.9 (13,114–14,137)	13,437.0 (11,166–14,089) 12,115.8 (10,593–14,309)
lsns-lsearch	11,982.5 (10,518–13,251) 12,207.2 (10,914–13,277)	12,288.5 (11,103–13,208) 12,598.0 (11,463–13,370)

Table 5: Average, minimum, maximum number of times repair-destroy iteration was run

4 Visualizations

4.1 TSPA.csv

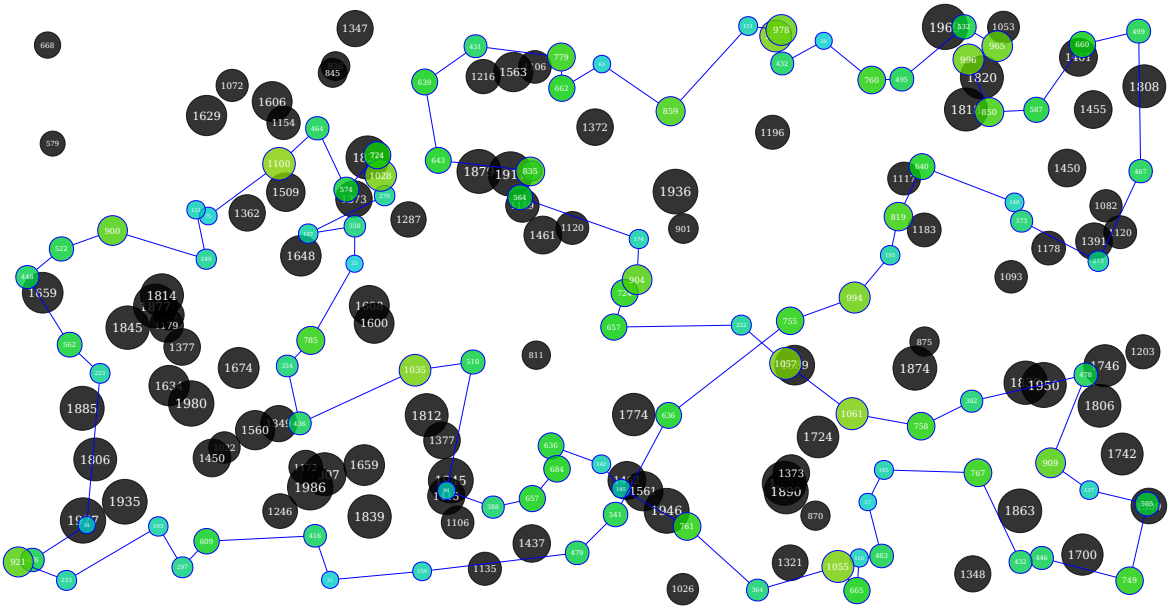

Figure 1: Best lsns-random solution to TSPA (73,202)

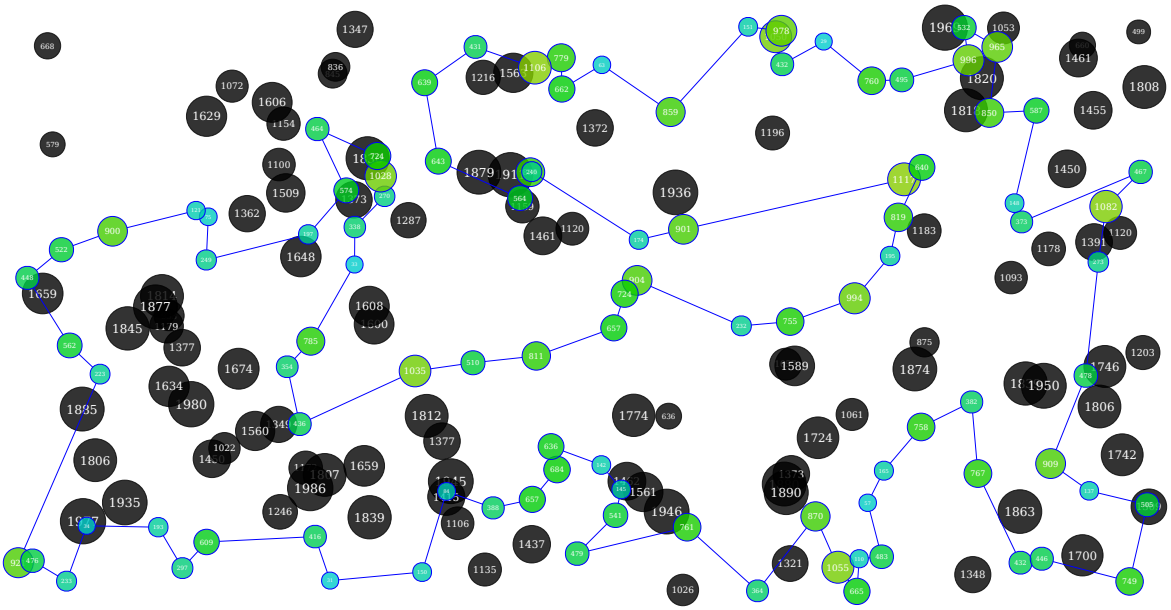

Figure 2: Best lsns-lsearch solution to TSPA (72,966)

4.2 TSPB.csv

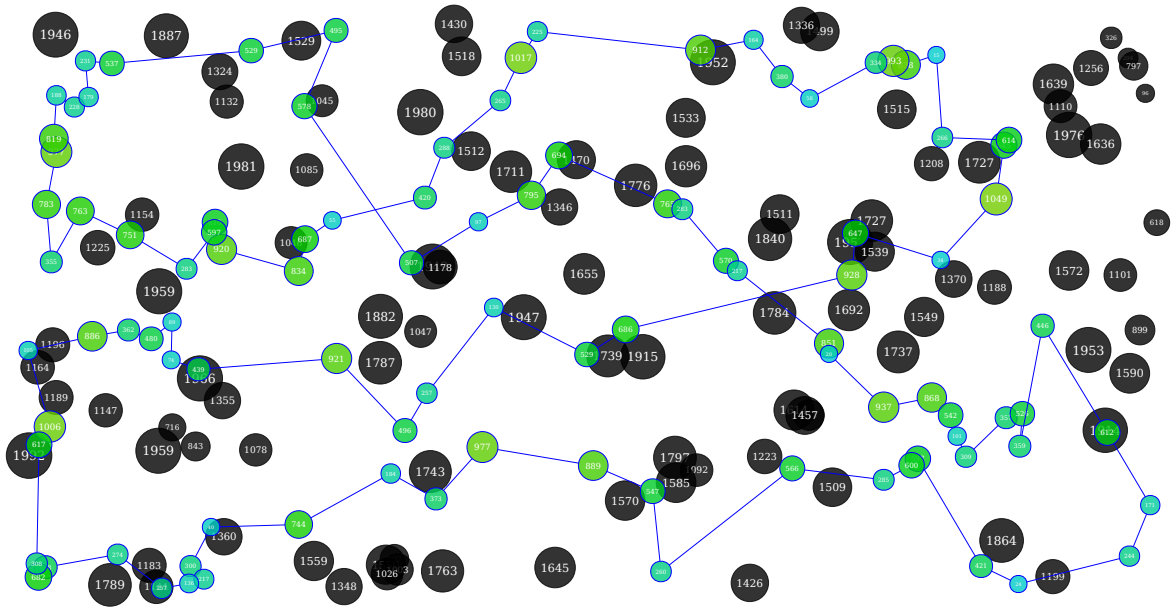

Figure 3: Best lsns-random solution to TSPB (66,684)

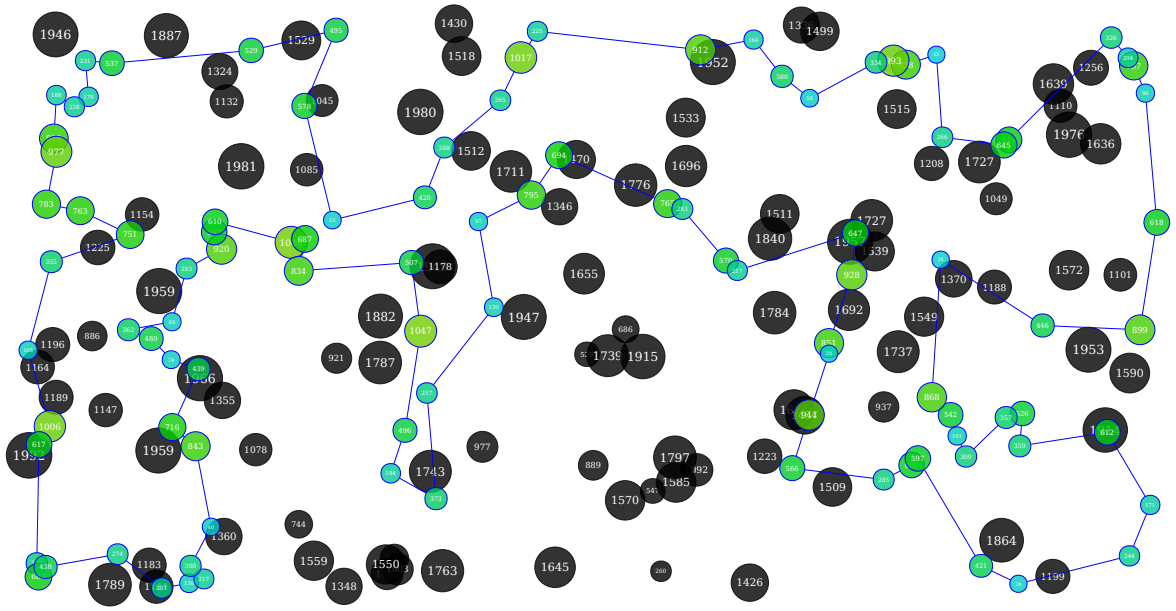

Figure 4: Best lsns-lsearch solution to TSPB (66,659)

4.3 TSPC.csv

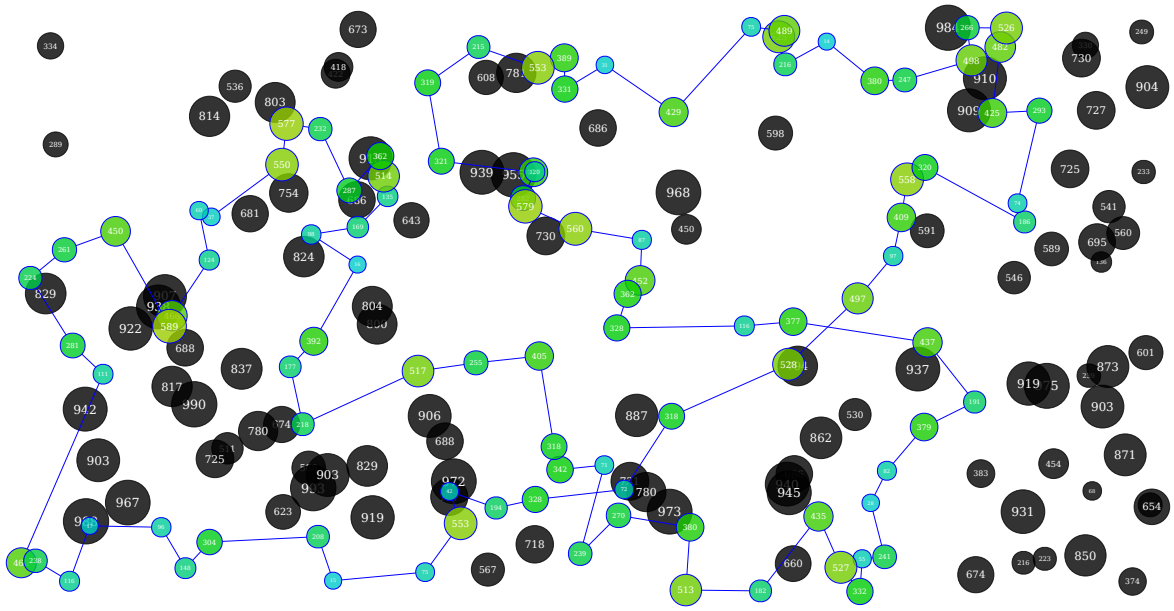

Figure 5: Best lsns-random solution to TSPC (47,391)

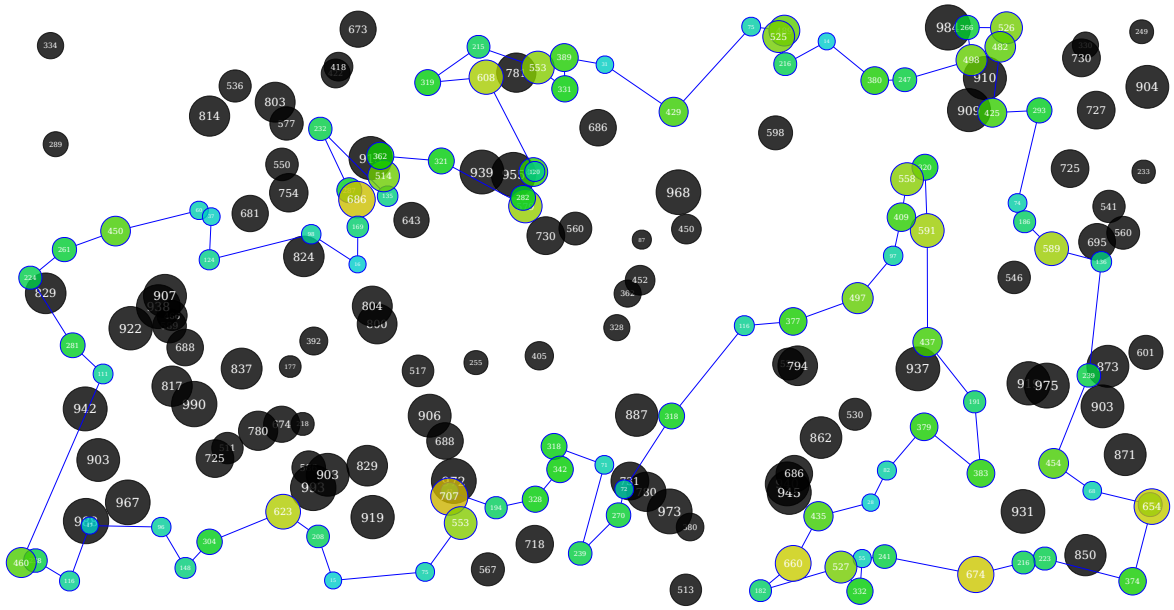

Figure 6: Best lsns-lsearch solution to TSPC (47,434)

4.4 TSPD.csv

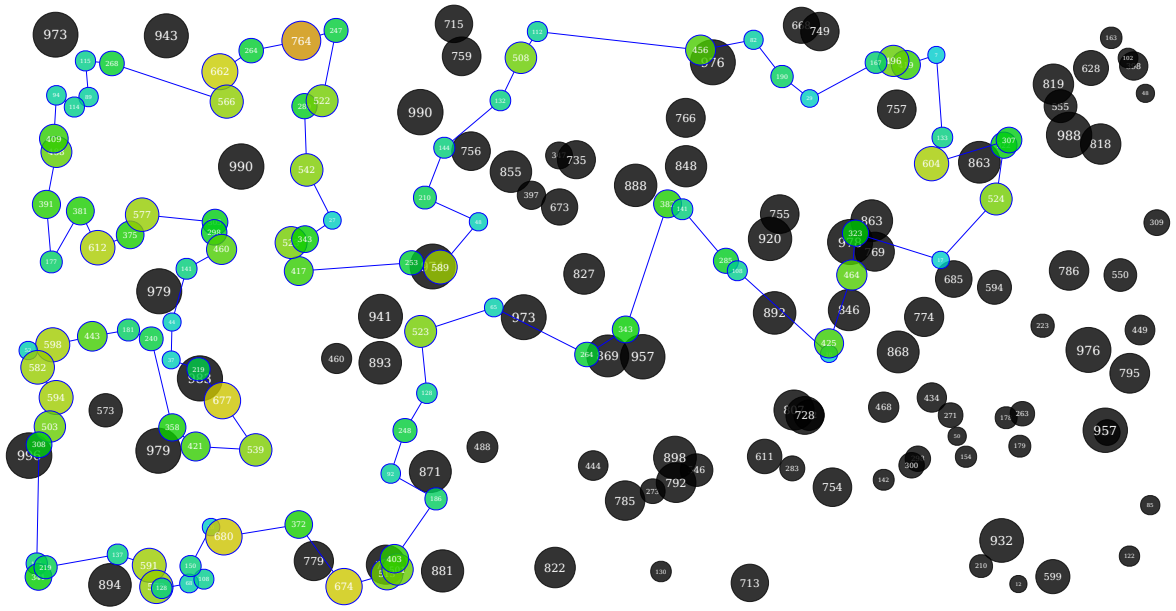

Figure 7: Best lsns-random solution to TSPD (44,006)

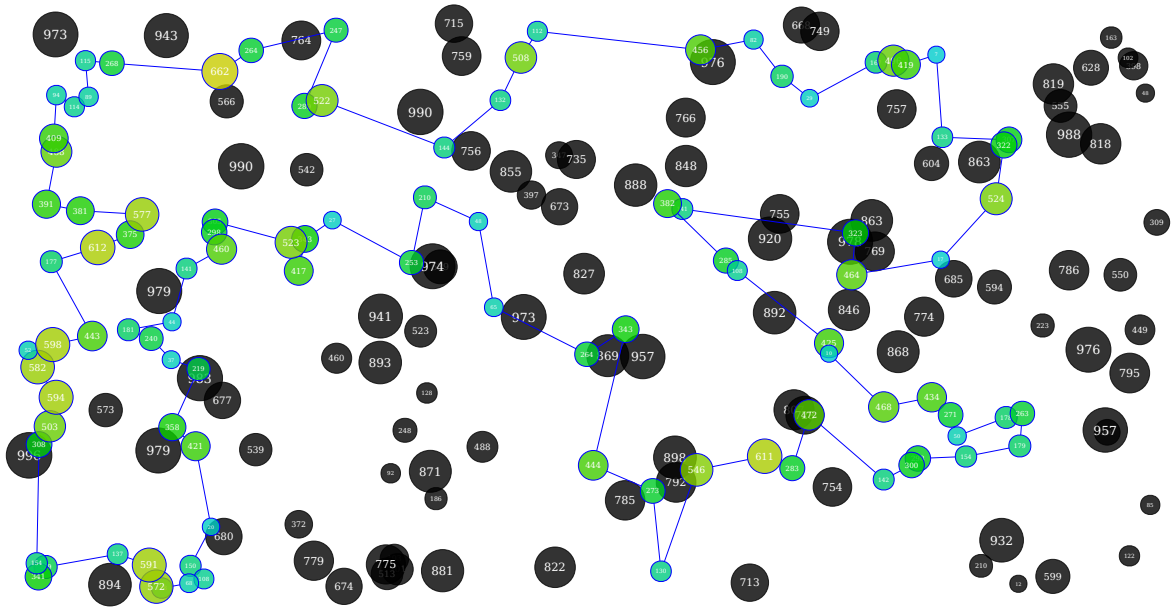

Figure 8: Best lsns-lsearch solution to TSPD (43,833)

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

Large scale neighborhood search was incredibly simple to implement (at least the variant I chose), yet it sufficed to improve the results for all instances.

I've noticed that the visualizations of the graphs look much different than in previous reports – there is a tendency for the graphs to be more "compact" and occupy a smaller area of all nodes in the solution. Also, there don't seem to be any really high-cost nodes. Both of these changes are a really good sign.

Perhaps more improvements can be found by tweaking the repair-destroy operator (adding some fast heuristic, like taking most costly nodes out), or by building off of different solutions (not random, not local search optimum → maybe greedy?).