Evolutionary Computation Lab II

Piotr Kaszubski 148283

Monday, October 23, 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 Corrections from report 1

Node costs in neighborhood

It's been pointed out that neighborhood in the previous report should have included node costs (which I intentionally left out when computing nearest neighbor). This led to higher (worse) results than what's possible when including the cost.
Fixed in f4ca6c8.

Major error in greedy cycle

Furthermore, I have discovered a major error in my greedy cycle implementation, that renders all results incredulous. The error regarded pairwise node iteration, which is critical for computing deltas.
Fixed in d66d28c.

Updated results are available in the *Results* section.
For updated visualizations, see here.

3 Pseudocode

3.1 RCL

Input set of graph nodes `G`, set of candidate nodes `N`, size `S`, percentage `P`

Output An RCL `N'` (subset of `N`) of size `S`, chosen by repeatedly sampling from `P`% best nodes

Steps

1. Initialize `N'` to an empty list.
2. While `size(N')` ≠ `S`, do:
   (a) Construct a subset `M` of `N`, which contains a random `P`% of nodes from `N`.
   (b) If `size(G)` = `0`, then:
      i. Pick a random node from `M` and move it from `N` to `N'`.
   (c) Else If `size(G)` = `1`, then:
      i. Pick a node from `M` which is the **nearest neighbor** to the only node in `G` and move it from `N` to `N'`.
   (d) Else:
      i. Pick a node from `M` which constitutes the smallest increase in the objective function (**greedy cycle**) for `G`; move it from `N` to `N'` .

3.2 2_regret

Input set of graph nodes `G`, candidate node `n`

Output 2-regret of `n` with respect to `G`.

Steps

1. Initialize `d1` to the objective function delta if `n` was inserted between the *first* pair of nodes from `G`.
2. Initialize `d2` to the objective function delta if `n` was inserted between the *second* pair of nodes from `G` (”second pair” = 2nd and 3rd, **not** 3rd and 4th!).
3. Ensure `d1` > `d2` (swap values if necessary).
4. For every remaining (3rd, 4th, ...) pair `p` of nodes in `G`, do:
   (a) Compute the objective function delta `d` if `n` was inserted between `p`.
   (b) If `d` > `d1`, then:

            i. Let `d2` = `d1`.
            ii. Let `d1` = `d`.
        (c) Else if `d` > `d2`, then:
            i. Let `d1` = `d`.
5. Return `d1 - d2`.

3.3 2-regret solution

Input A set `N` of nodes.

Output An ordered subset `N'` of `N`, with half of `N`'s cardinality.

Steps

1. Initialize `N'` to an empty list.
2. While `len(N') < len(N)/2`, do:
    (a) Let `rcl` = `RCL(N', 10, 0.04)`.
    (b) Choose node `n` from `N`, such that `2_regret(N', n)` is maximal.
    (c) Remove `n` from `N`.
    (d) Add `n` to `N'`.

3.4 Weighted sum criterion solution (50/50)

Input A set `N` of nodes.

Output An ordered subset `N'` of `N`, with half of `N`'s cardinality.

Steps

1. Initialize `N'` to an empty list.
2. While `len(N') < len(N)/2`, do:
    (a) Let `rcl` = `RCL(N', 10, 0.04)`.
    (b) Choose node `n` from `N`, such that `2_regret(N', n) - delta` is maximal, where `delta` denotes the best (minimal) objective function change attainable by inserting `n` anywhere in `N'`.
    (c) Remove `n` from `N`.
    (d) Add `n` to `N'`.

4 Results

ALG.	FILE	min	avg	max
random	TSPA.csv	241,510	266,062	308,034
	TSPB.csv	241,731	266,549	293,093
	TSPC.csv	189,473	215,587	239,581
	TSPD.csv	195,876	218,919	250,422
nn	TSPA.csv	~~110,035~~ 84,471	~~116,145~~ 87,649	~~125,805~~ 95,013
	TSPB.csv	~~106,815~~ 77,448	~~116,181~~ 79,304	~~124,675~~ 82,669
	TSPC.csv	~~62,629~~ 56,304	~~66,196~~ 58,877	~~71,616~~ 63,304
	TSPD.csv	~~62,788~~ 50,335	~~66,847~~ 54,406	~~71,396~~ 59,846
cycle	TSPA.csv	~~113,298~~ 75,666	~~123,691~~ 77,069	~~129,175~~ 80,321
	TSPB.csv	~~111,981~~ 68,764	~~120,922~~ 70,685	~~131,174~~ 76,324
	TSPC.csv	~~67,077~~ 53,226	~~72,771~~ 55,845	~~75,763~~ 58,876
	TSPD.csv	~~66,193~~ 50,409	~~71,606~~ 55,055	~~77,797~~ 60,077
2-regret	TSPA.csv	86,224	103,385	119,577
	TSPB.csv	83,179	97,807	113,449
	TSPC.csv	58,778	66,161	71,821
	TSPD.csv	53,396	63,387	69,441
wsc	TSPA.csv	80,427	87,553	101,498
	TSPB.csv	73,310	82,983	95,733
	TSPC.csv	54,306	60,334	66,962
	TSPD.csv	51,420	58,478	66,263

5 Visualizations

5.1 TSPA.csv

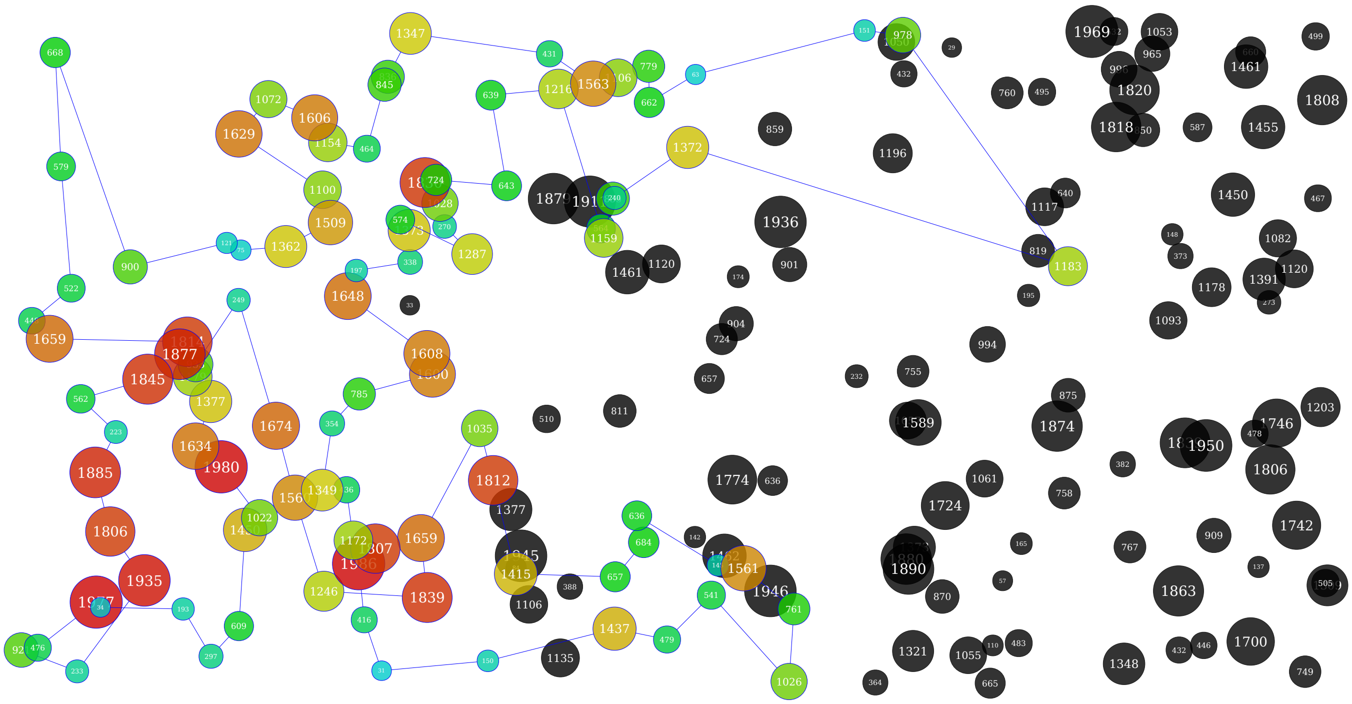

Figure 1: Best 2-regret solution to TSPA (119,577)

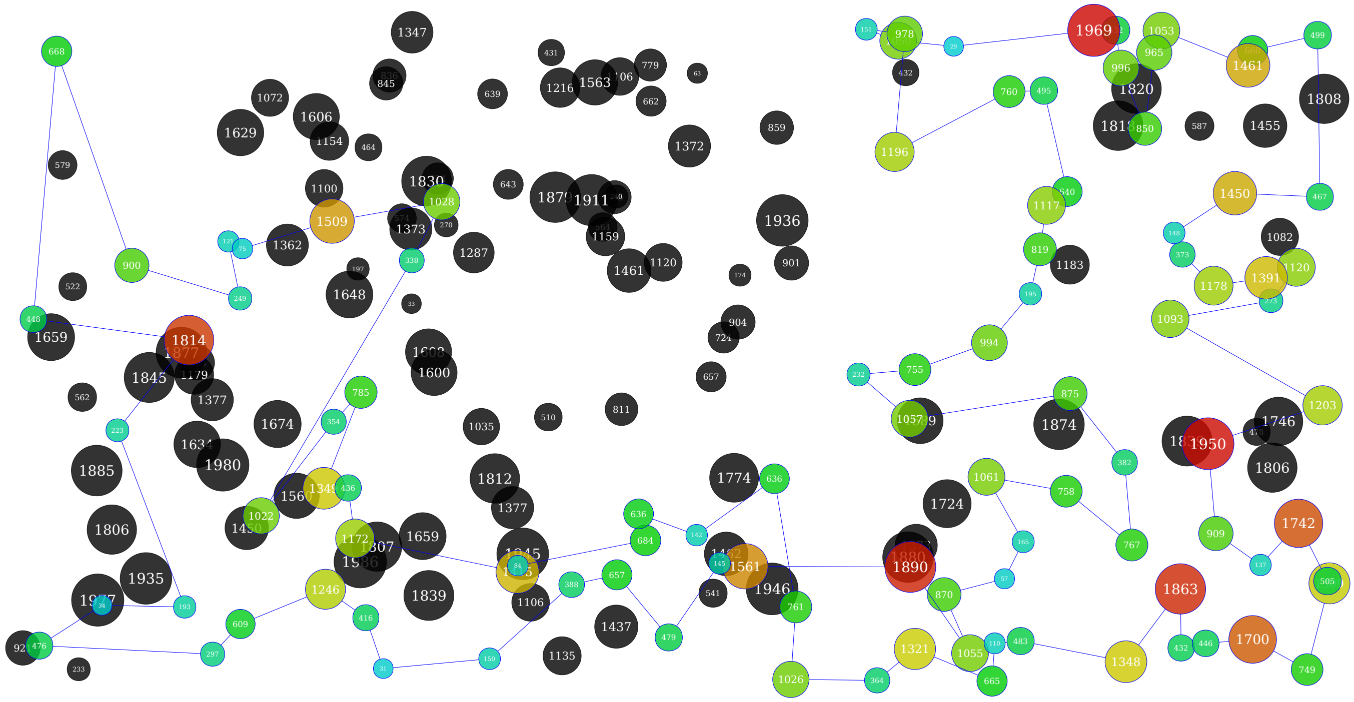

Figure 2: Best weighted-sum-criterion solution to TSPA (101,498)

5.2 TSPB.csv

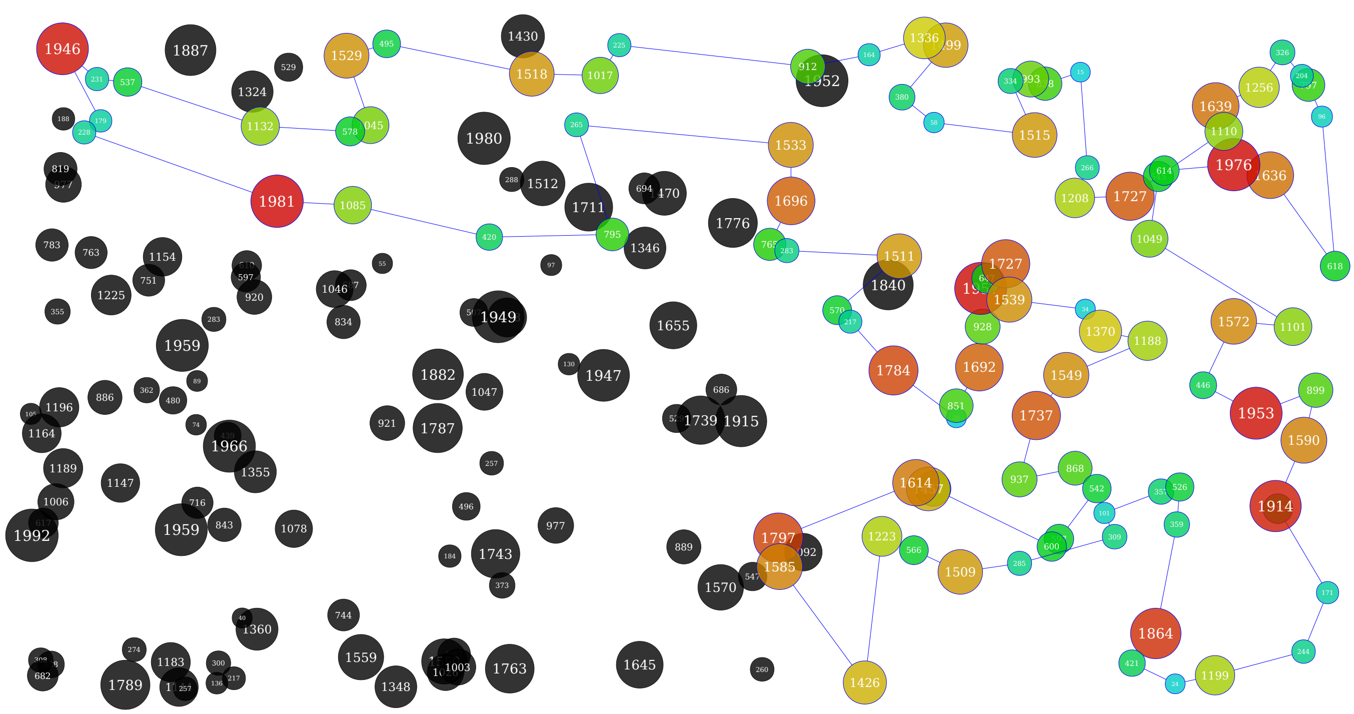

Figure 3: Best 2-regret solution to TSPB (113,449)

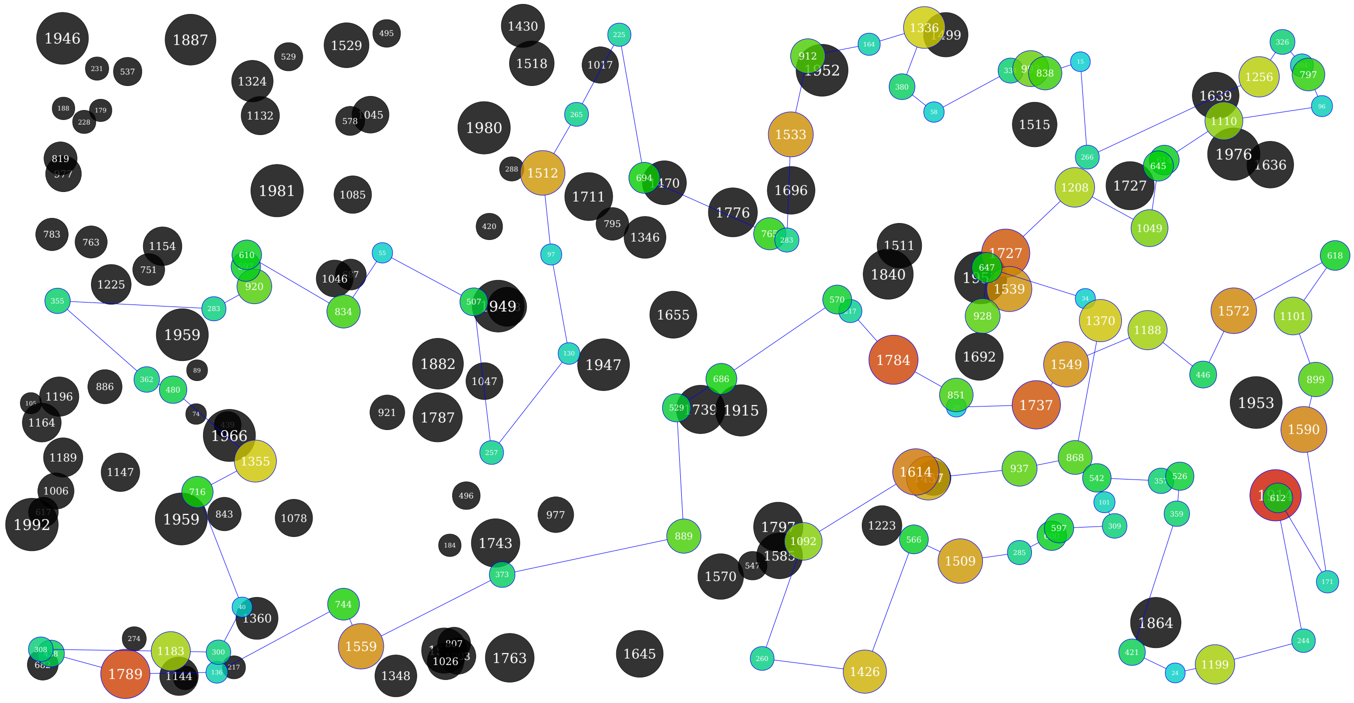

Figure 4: Best weighted-sum-criterion solution to TSPB (95,733)

5.3 TSPC.csv

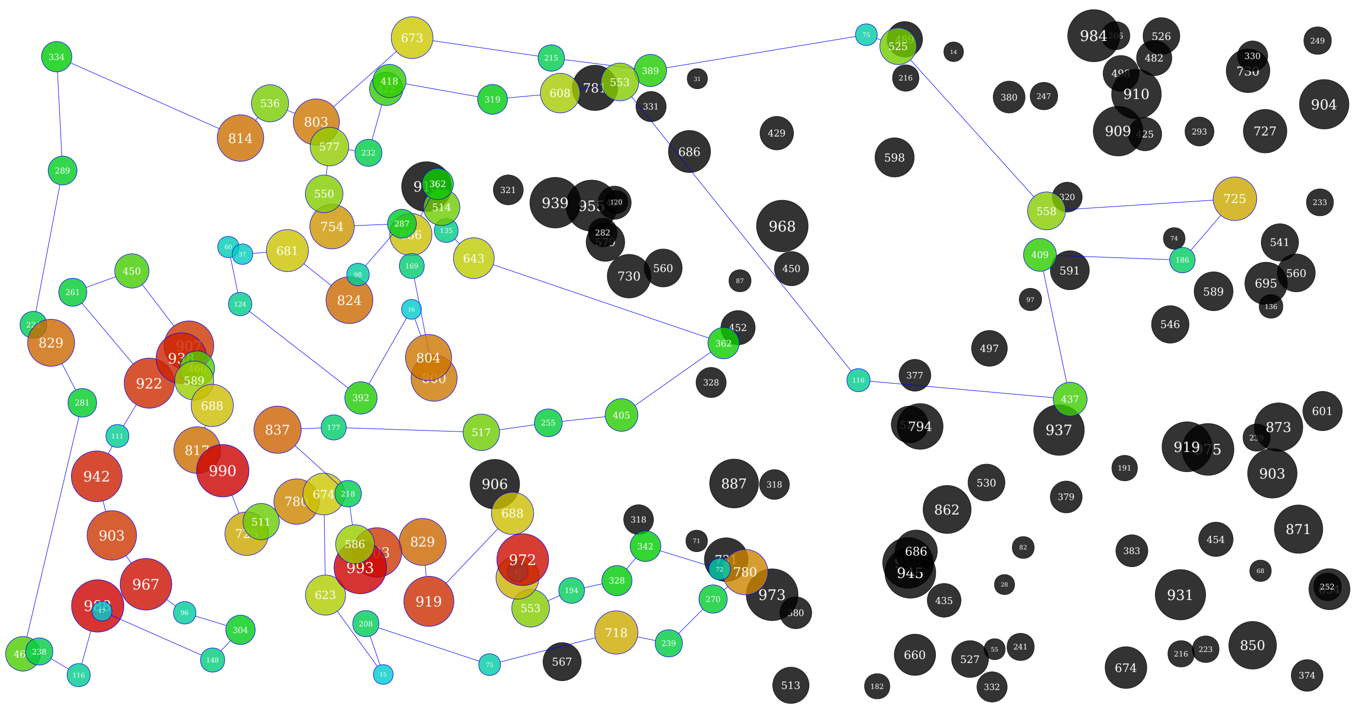

Figure 5: Best 2-regret solution to TSPC (71,821)

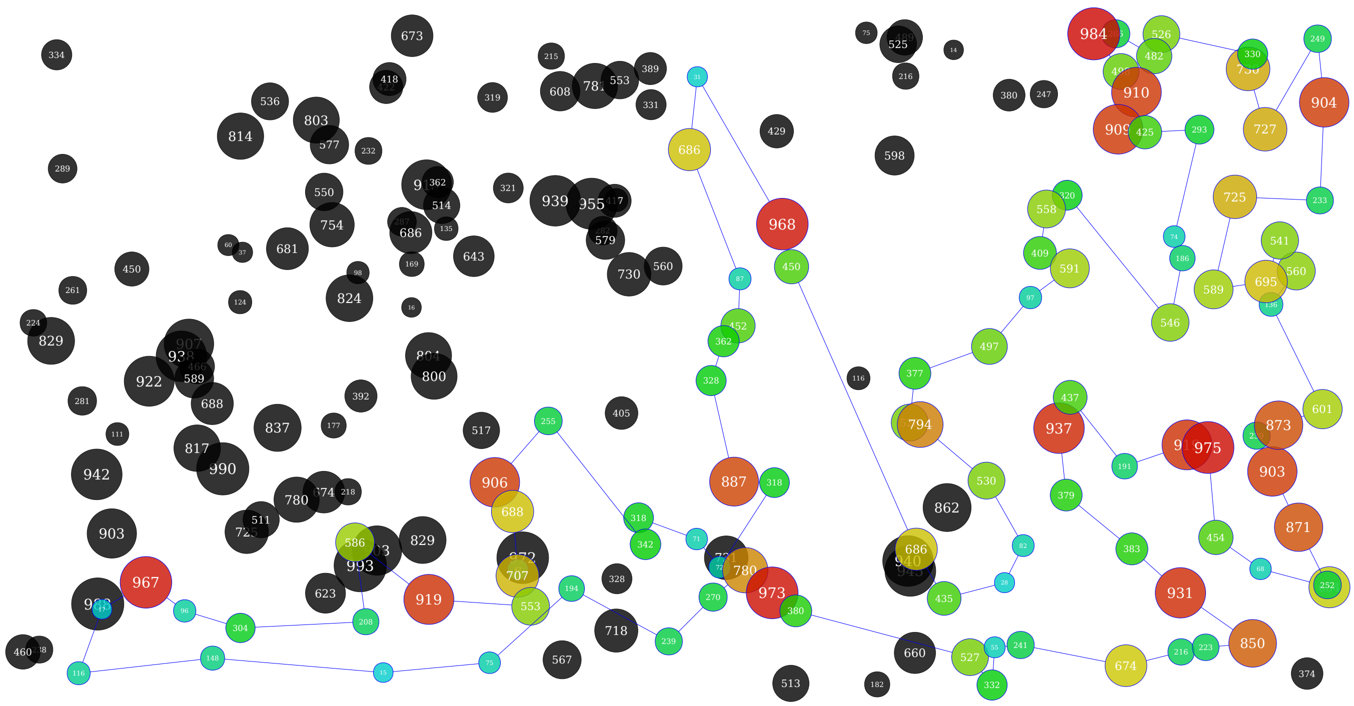

Figure 6: Best weighted-sum-criterion solution to TSPC (66,962)

5.4 TSPD.csv

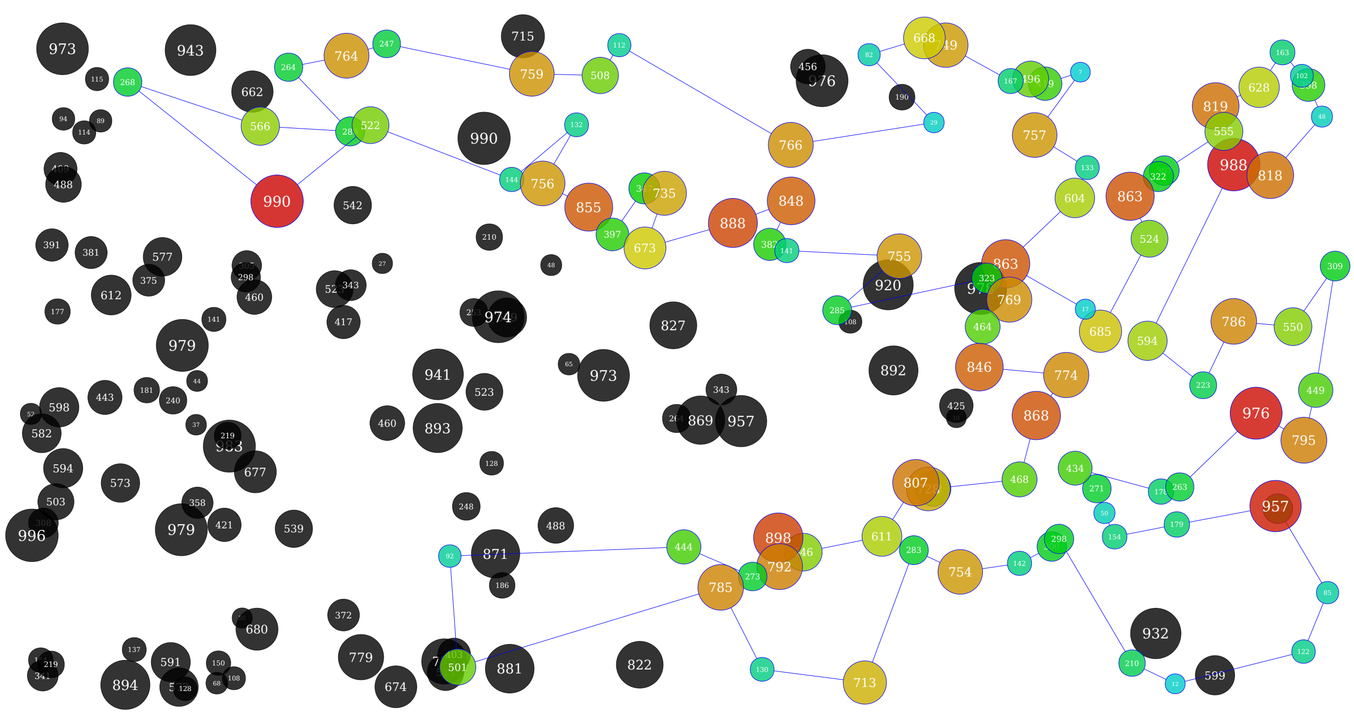

Figure 7: Best 2-regret solution to TSPD (69,441)

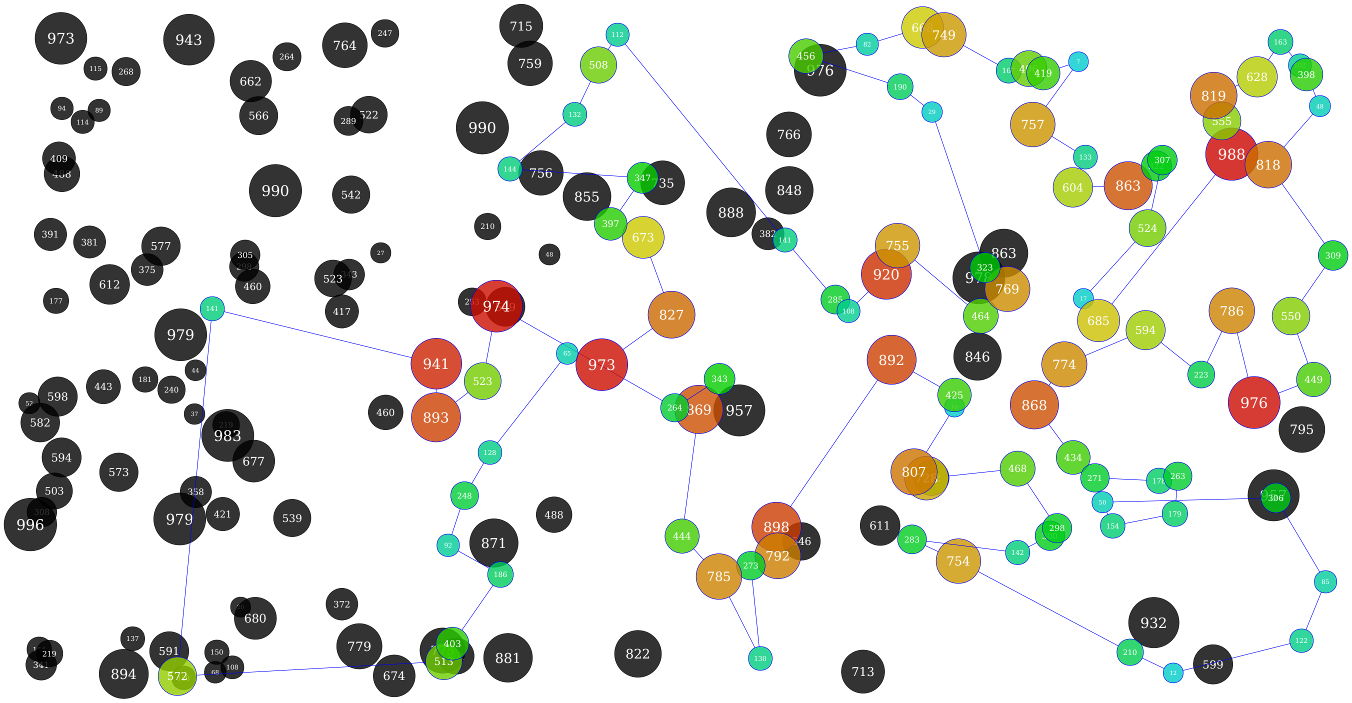

Figure 8: Best weighted-sum-criterion solution to TSPD (66,263)

6 Source code

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

7 Conclusions

After updating the first report, greedy cycle became the best method, and results obtained in this report seem to reinforce that – While 2-regret is pretty good, it gives worse solutions and weighing it with greedy cycle always leads to improvements.

In hindsight, I don't think implementing Restricted Candidate List was at all necessary for this assignment, but I did it anyway (initially I thought I needed it, and then it was too late to turn back).