Evolutionary Computation - Assignment 6

Sebastian Chwilczyński 148248, Karol Cyganik 148250

December 4, 2023

1 Description of the problem

We are given a graph with n n nodes, each described by its x and y coordinates and a node cost. The goal is to select precisely $ceil(n/2)$ nodes that form a Hamiltonian cycle (closed path) and minimize

$$\sum_{e \in E} cost(e) + \sum_{v \in V} cost(v)$$

Where E is a set of selected edges, $cost(e)$ is Euclidean distance between two nodes rounded mathematically to an integer value, V is a set of selected nodes and $cost(v)$ is node cost.

2 Pseudocode

2.1 Multiple Start Local Search

Algorithm 1 Algorithm that runs LS n times.

Input: n - number of repetitions in one run
Output: The best-found solution

```
1: best_solution = generate_random_solution()
2: best_cost = ∞
3: for all i ∈ n do
4:     starting_solution = generate_random_solution()
5:     result = perform_local_search(starting_solution)
6:     cost = calculate_cost(result)
7:     if cost < best_cost then
8:         best_cost = cost
9:         best_solution = result
10:    end if
11: end for
12: return best_solution
```

2.2 Perturbation generation

We used two types of perturbations:

1. Random intra or inter neighbourhood perturbation
2. Random shuffle of subsolution

Experimentally, we observed that maintaining a constant number of perturbations leads to faster convergence of the algorithm, which hinders further improvement. To mitigate this problem, we adopted a strategy of progressively increasing the number of perturbations as the algorithm advances. Our hypothesis is that as we get better and better local optimum it is harder to escape it. Thus, bigger perturbations are needed.

$$f(n\_iterations) = \begin{cases} 10 & \text{if } n\_iterations < 100 \\ 20 & \text{elif } n\_iterations < 300 \\ 25 & \text{elif } n\_iterations < 500 \\ 30 & \text{elif } n\_iterations < 700 \\ 40 & \text{otherwise} \end{cases}$$

Figure 1: Function which returns number of inter/intra perturbations

$$g(n\_iterations) = \begin{cases} 5, 20 & \text{if } n\_iterations < 100 \\ 5, 30 & \text{elif } n\_iterations < 300 \\ 10, 30 & \text{elif } n\_iterations < 500 \\ 10, 40 & \text{elif } n\_iterations < 700 \\ 20, 50 & \text{otherwise} \end{cases}$$

Figure 2: Function which returns bounds of length of the shuffle

Algorithm 2 Algorithm that generates random perturbation.

Input: solution, n_iterations
Output: randomly permuted solution

if $random(0, 1) > 0.5$ **then**
  for $p \leftarrow 1$ to $f(n\_iterations)$ **do**
    if $random(0, 1) > 0.5$ **then**
      $solution \leftarrow random\_intra\_change(solution)$
    else
      $solution \leftarrow random\_inter\_change(solution)$
    end if
  end for
else
  $min\_n\_shuffle, max\_n\_shuffle \leftarrow g(n\_iterations)$
  $n\_shuffled \leftarrow randint(min\_n\_shuffle, max\_n\_shuffle)$
  $solution \leftarrow shuffle\_solution\_part(solution, n\_shuffled)$
end if
return *solution*

2.3 Iterated Local Search

Algorithm 3 Algorithm that uses move evaluations (deltas) from previous iterations in local search.

Input: initial solution
Output: The best-found solution

```
1: solution ← initial solution
2: max_time ← MSLS_avg_time
3: time ← start_timer()
4: while time < max_time do
5:     perturb(solution)
6:     new_solution ← LocalSearch(solution)
7:     if cost(new_solution < cost(solution) then solution ← new_solution
8:
9:         return solution
```

3 Results

Table 1: Combined Results for TSP A & B Variants

Method	TSPA (avg(min-max))
steepest LS, edges, random start	77956(74782-81590)
MSLS	75183.3(74705-75631)
ILS	74133.4(**73213**-74934) height

Method	TSPB (avg(min-max))
steepest, edges, random start	71373(68219-75872)
MSLS	68174.8(66596-68707)
ILS	67076.1(**66596**-67787) height

Table 2: Combined Results for TSP C & D Variants

Method	**TSPC (avg(min-max))**
steepest LS, edges, random start	51362(48804-54796)
MSLS	49102.3(48544-49405)
ILS	48404.5(**47841**-49526) height
Method	**TSPD (avg(min-max))**
steepest LS, edges, random start	48335(45155-53428)
MSLS	45492.9(44783-46017)
ILS	44416(**43667**-45289) height

Table 3: Consolidated Time Results

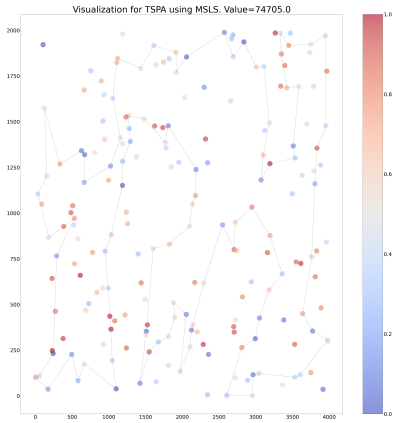
Method	**TSPA (s)**	**TSPB (s)**
steepest LS, edges, random start	7.01 (6.95 - 7.14)	7.19 (7.13 - 7.28)
MSLS	1403 (1391 - 1430)	1450 (1422 - 1496)
ILS	1403	1450
Method	**TSPC (s)**	**TSPD (s)**
steepest LS, edges, random start	6.96 (6.89 - 7.02)	7.06 (7.03 - 7.09)
MSLS	1398 (1389 - 1401)	1420 (1413 - 1440)
ILS	1398	1420

Table 4: Number of iterations in ILS

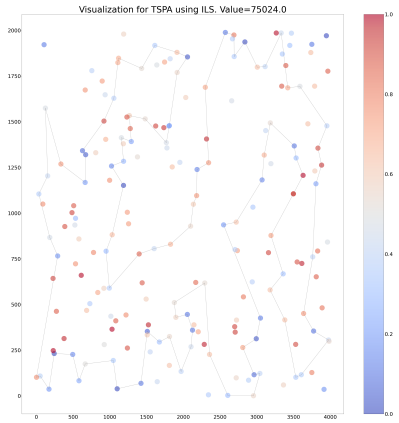
Method	**TSPA (n)**	**TSPB (n)**	**TSPC (n)**	**TSPD (n)**
Avg	278	250	237	240
Best	196	211	207	217
Worst	276	290	288	273

4 Code

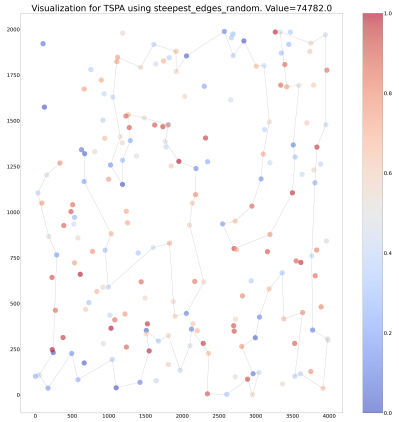
Implementation of algorithms and visualizations is available here



(a) MSLS

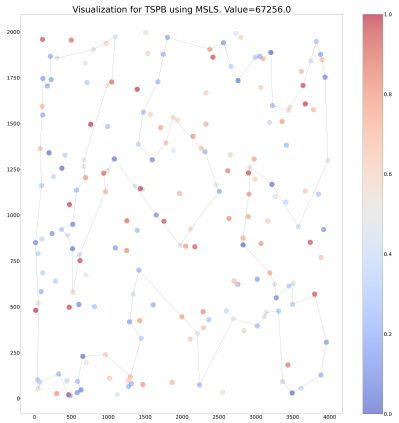


(b) ILS

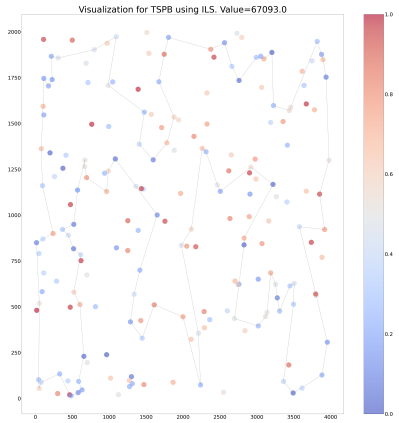


(c) just steepest

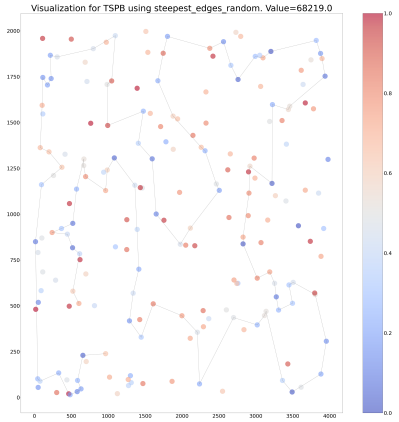
Figure 3: MSLS vs ILS vs just steepest algorithm



(a) MSLS

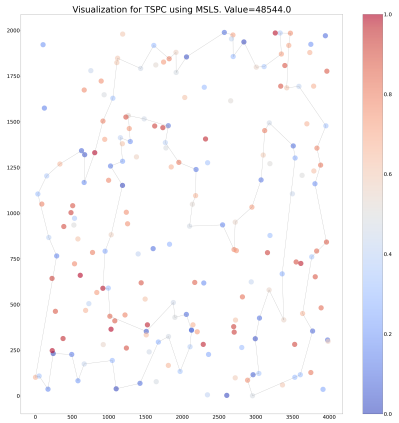


(b) ILS

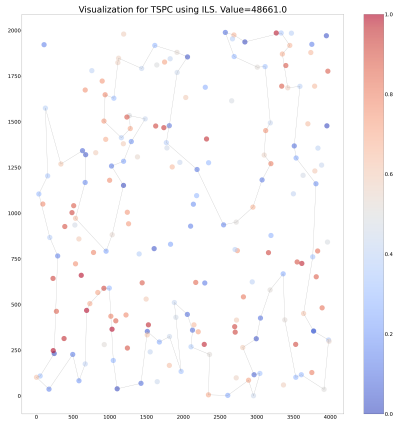


(c) just steepest

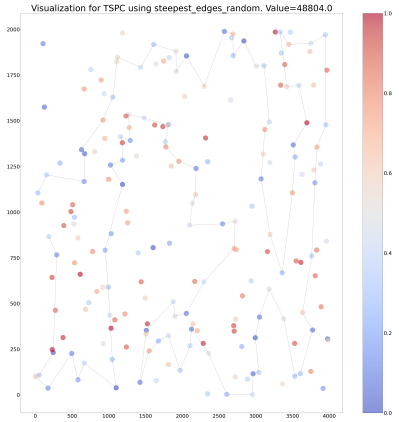
Figure 4: MSLS vs ILS vs just steepest algorithm



(a) MSLS

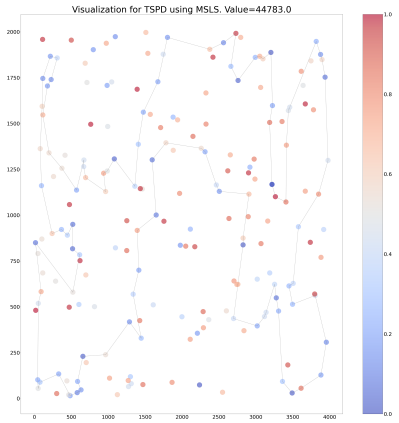


(b) ILS

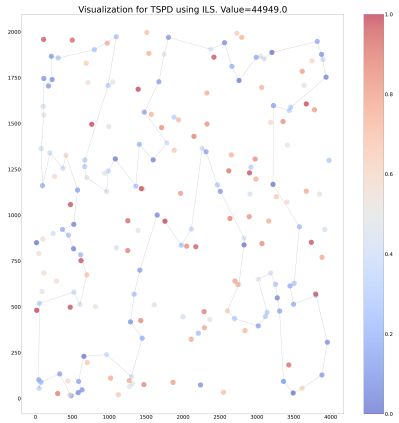


(c) just steepest

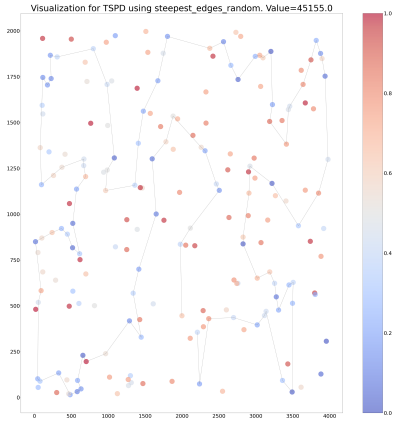
Figure 5: MSLS vs ILS vs just steepest algorithm



(a) MSLS



(b) ILS



(c) just steepest

Figure 6: MSLS vs ILS vs just steepest algorithm

5 Conclusions

As anticipated, giving MSLS additional time resulted in the discovery of marginally improved solutions. However, the extent of improvement was relatively modest, despite employing a computational resource that was 20 times more extensive. Importantly, ILS, with the same amount of computing time, has beaten MSLS. This phenomenon can be attributed to a dual-fold rationale:

1. Small perturbations still keep us in potentially valuable areas of the solution space.

2. The optimization process for perturbed solutions is notably faster, leading to a higher effective iteration count and, consequently, more refined outcomes.