Assignment 7: Large Neighborhood Search

Mateusz Tabaszewski 151945 Bartłomiej Pukacki 151942

Source code:
https://github.com/MatTheTab/Evolutionary-Computation/blob/main/Assignment%207%20Large-scale%20neighborhood%20search/Assignment_7_Large_Neighborhood.ipynb

1. Description of the Problem

This report is focused on exploring the possibility of using the Large Neighborhood Search algorithm to improve the quality of the redefined TSP described in the previous reports. Similarly to the previous reports, the problem requires the algorithm to find the cycle with the smallest value of the objective function with both the distance between nodes and the weight of each node in the solution being taken into consideration - however - only half of all nodes need to be included in the final solution. This definition is consistent with the one presented in the previous reports, as such we will skip the presentation of the mathematical formulation of the problem, however, for the sake of comparison we decided to include the results of previous experiments.

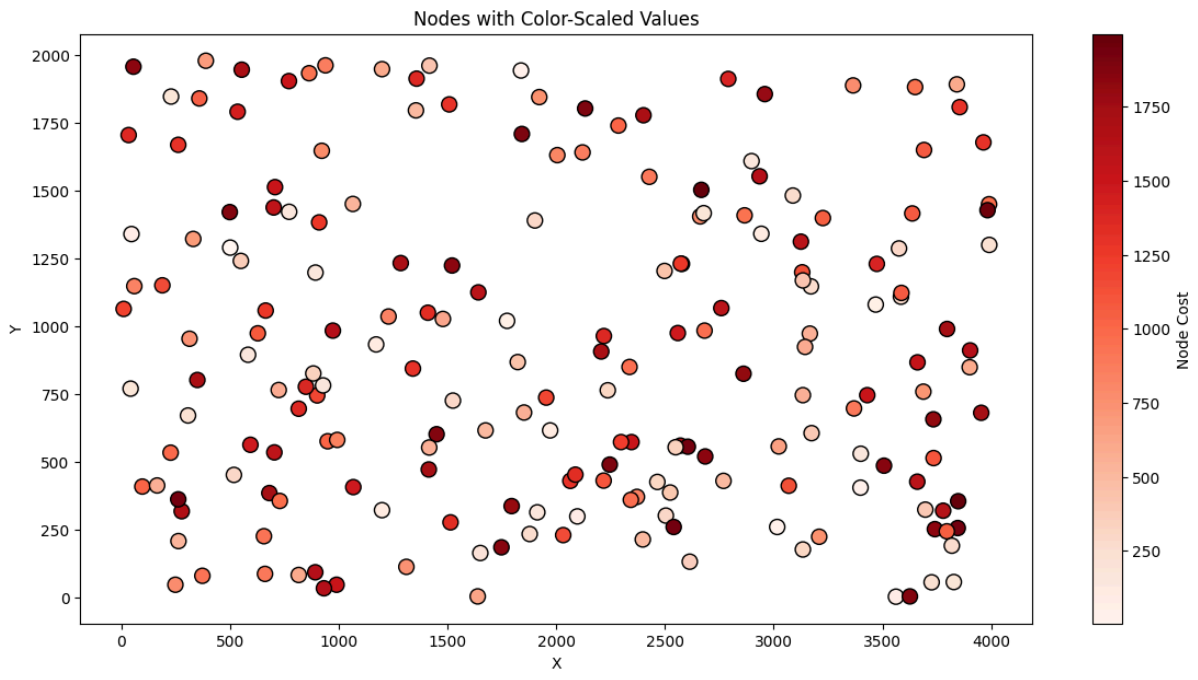

Fig 1. Visualization of the TSPA problem instance, each node's x and y locations on the plot correspond to their given x and y locations and the color intensity signifies the weight/cost of each node. The total length of the cycle and the sum of node weights should be minimized.

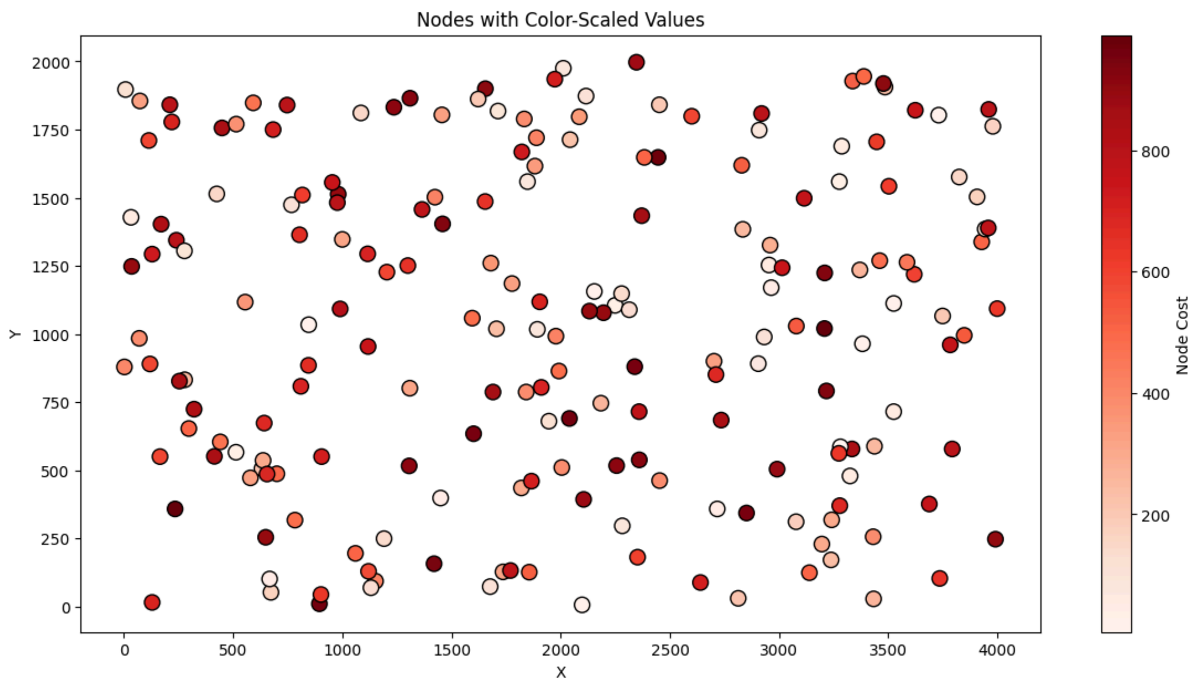

Fig 2. Visualization of the TSPB problem instance, each node's x and y locations on the plot correspond to their given x and y locations and the color intensity signifies the weight/cost of each node. The total length of the cycle and the sum of node weights should be minimized.

2. Algorithms

The algorithm covered in this report is the **Large Neighborhood Search** algorithm. The method requires the definition of two fundamental mechanisms - destroy and repair. However, before these mechanisms are described in greater detail, let us define the workings of the Large Neighborhood Search. First, an initial solution must be defined, in our case it is the random solution. Then optionally, a local search algorithm can be run over the previously generated solution, however, in accordance with the instruction, the local search over the initial solution was run for every experiment, despite being an optional element in the algorithm's definition. Afterward, the solution is destroyed and then repaired using the aforementioned mechanisms. Another optional element allows us to perform a local search on the now repaired solution, both the experiments with this optional local search and without it were included in the results. Lastly, if the new solution is better than the previous one, then it becomes our new default solution to be destroyed and repaired, otherwise, we continue working on the same solution as before. The algorithm continues until the termination conditions are met, in our case it was the runtime reaching the time threshold defined in the earlier experiments by the MSLS algorithm. Now that the basics of the algorithm are well-known, it is important to describe the destroy and repair mechanisms in greater detail. The destroy method should remove a series of consecutive nodes, destroying the original cycle that is our solution. The mechanism should include an element of randomness but can be guided by some heuristic, like preferring to delete paths, however, in the end, the randomness is necessary to the iterative nature of the algorithm. Additionally, the repair algorithm should be able to create a path from all of the available nodes to put into the solution in place of the originally destroyed solution path. The implemented and tested destroy and repair algorithms are described below.

Destroy Single Path - This operator randomly selects a starting node belonging to the solution and removes from the solution some number of consecutive nodes specified in the operator, starting from the starting node. In our experiments, we set the number of deleted consecutive nodes to be 30.

Repair Weighted Regret - The implemented Repair mechanism is based on the use of the weighted regret heuristic algorithm to find the best nodes for insertion into the deleted path. The algorithm works by iterating over each node and each insert position. For each node, it calculates the regret defined as the difference between the score for the best possible insertion position and the score for the second best insertion position. Also, a score difference between the solution before and after the insertion of this node is calculated. Both the regret and the score difference are taken into account while calculating the weighted score for the node (-0.5 * regret and 0.5 * score_difference), the weighted score is to be minimized and the node with the smallest weighted score is inserted in its best position. Then the process is repeated until the destroyed path is fixed, however, the nodes that are already in the solution are not taken into consideration.

```
FUNCTION Large_Neighborhood_Search(distance_matrix, weights, allowed_time,
destroy, repair, optional_steepest):
    INPUT:
        distance_matrix - matrix of distances between nodes
        weights - an array of weights associated with each node
        allowed_time - maximum time allowed for the algorithm's runtime
        destroy - destroy operator to be used
        repair - repair operator to be used
        optional_steepest - if the optional local search algorithm should be used

    current_solution, current_score ← FIND random solution
    current_solution, current_score ← RUN local search algorithm ON current_solution
    num_iterations = 0

    WHILE current runtime < allowed_time:
       num_iterations += 1
       current_solution, current_score ← destroy(current_solution,
                                     current_score, distance_matrix, weights)
        current_solution, current_score ← repair(current_solution,
current_score,
                                                distance_matrix, weights)
        IF optional_steepest:
            current_solution, current_score ← RUN local search algorithm ON
                                                    current_solution
        IF current_score IS BETTER THAN best_score:
            best_score ← current_score
            best_solution ← current_solution
        ELSE:
            current_score ← best_score
            best_solution ← current_solution
    RETURN best_solution, best_score, num_iterations
```

```
FUNCTION destroy(solution, score, distance_matrix, weights, num_destroy_nodes =
30):
    INPUT:
        solution - the solution to be destroyed
        score - the current score of the solution
        distance_matrix - matrix of distances between nodes
        weights - an array of weights associated with each node
        num_destroy_nodes - number of consecutive nodes to be removed

    start_node ← CHOOSE random node FROM solution
    end_node ← start_node + num_destroy_nodes
    DELETE solution[start_node : end_node]
    UPDATE score OF solution
    RETURN solution, score

FUNCTION repair(solution, score, distance_matrix, weights, num_destroy_nodes =
30):
    INPUT:
        solution - the solution to be destroyed
        score - the current score of the solution
        distance_matrix - matrix of distances between nodes
        weights - an array of weights associated with each node
        num_destroy_nodes - number of consecutive nodes to be removed


    available_nodes ← ALL NODES EXCEPT FOR nodes IN solution
    WHILE num_destroy_nodes > 0:
        for node in available_nodes:
            best_score ← BEST score FOR node FROM ALL insertion locations
            second_best ← SECOND BEST score FOR node FROM ALL insertion
                                                                   locations
            regret ← best_score - second_best
            regret_location ← insert location FOR best_score
            score_difference ← best_score - score
            weghted_score ← (0.5 * score_difference) + (-0.5 * regret)

            if weghted_score IS BETTER THAN best_weighted_score:
                best_weighted_score ← weghted_score
                best_node ← node
                best_location ← regret_location

        INSERT best_node AT best_location INTO solution
        UPDATE score
        REMOVE best_node FROM available_nodes
        num_destroy_nodes -= 1

    RETURN solution, score
```

3. Experiments

Experiments include all previous methods along with the Large Neighborhood Search. Greedy methods were run 200 times for each instance while multi-start methods were run 20 times each.

Method	TSPA av (min - max)	TSPB av (min - max)	Avr. Num Iterations (TSPA/TSPB)
Large Neighborhood Search (optional LS)	**69457 (69207 - 69821)**	**44133 (43873 - 44463)**	**387/357**
Large Neighborhood Search (no LS)	**69640 (69250 - 70805)**	**44361 (44174 - 44657)**	**403/391**
MSLS	**71340 (70919-71756)**	**45952 (45365-46428)**	**200/200**
ILS - Partial Shuffle	**69545 (69141 - 70200)**	**43952 (43448 - 44659)**	**1548/1495**
Weighted Greedy Regret Cycle	**72133 (71108-73395)**	**50882 (47144-55700)**	
Steepest Delta Search	**73910 (71118-78710)**	**48574 (46300-51342)**	**-**
ILS - MST Perturbation	70133 (69246 - 71271)	44360 (43658 - 45019)	1582/1525
ILS - Coordinate Change	69882 (69460 - 70419)	44201 (43574 - 44853)	315/345
Steepest Candidate Search	77944 (73159-84951)	48497 (45342-52178)	**-**
Greedy LS Rand	85812 (78831-93289)	61000 (53759-69662)	-
Steepest LS Rand	87935 (75935-95175)	63036 (55323-70187)	-
Greedy LS Edges Rand	73781 (71507-76491)	48427 (45646-51763)	-
Steepest LS Edges Rand	73954 (70948-77934)	48366 (45576-51616)	**-**

Greedy LS Best	71627 (70687-72882)	45460 (43826-51301)	-
Steepest LS Best	71619 (70626-72950)	45415 (43826-50876)	-
Greedy LS Edges Best	71515 (70571-72460)	45040 (43790-50495)	-
Steepest LS Edges Best	71468 (70510-72614)	44976 (43921-50495)	-
Random	264301 (223539-308435)	213397 (179796-253866)	-
Nearest Neighbor Closest	85109 (83182-89433)	54390 (52319-59030)	-
Nearest Neighbor All	73180 (71179-75450)	45870 (44417-53438)	-
Greedy Cycle	72606 (71488-74350)	51345 (48765-57262)	-
Greedy Regret Cycle	115630 (105852-123171)	72656 (67568-77329)	-

Table 1. Minimum, average, and maximum scores achieved by each method on both problem instances.

The **best scores achieved** are visualized below.

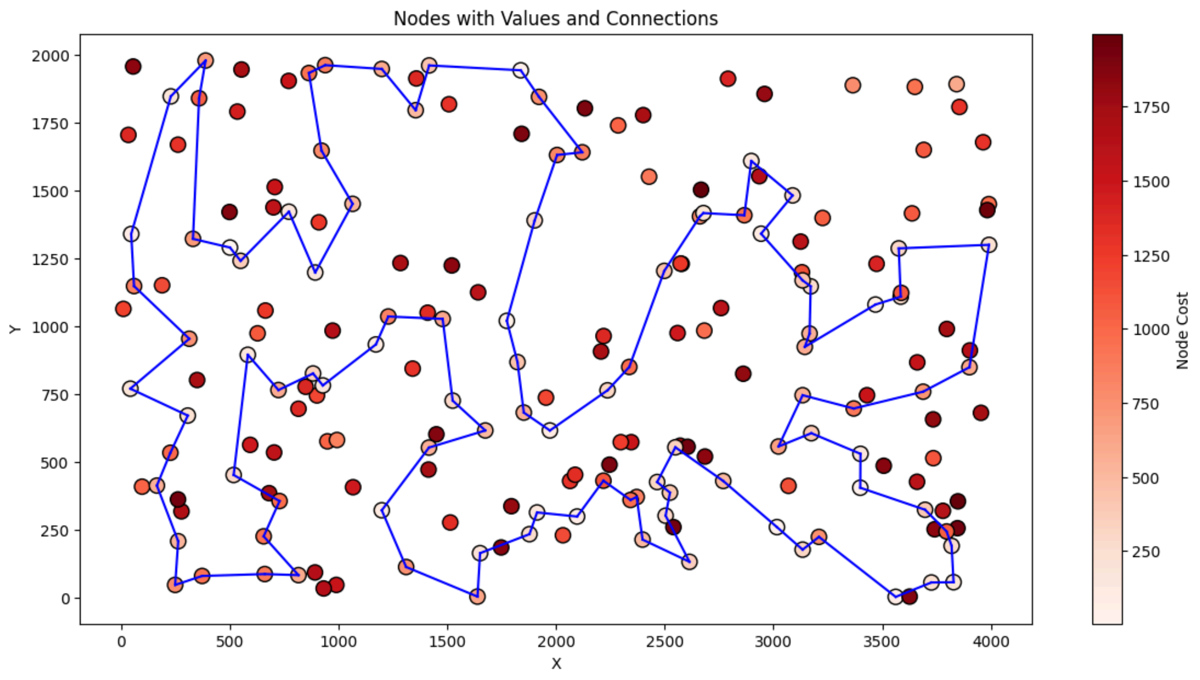

Fig 3. Visualization of the best solution found by the **Large Neighborhood Search (optional LS)** on the TSPA problem instance starting from a random solution.

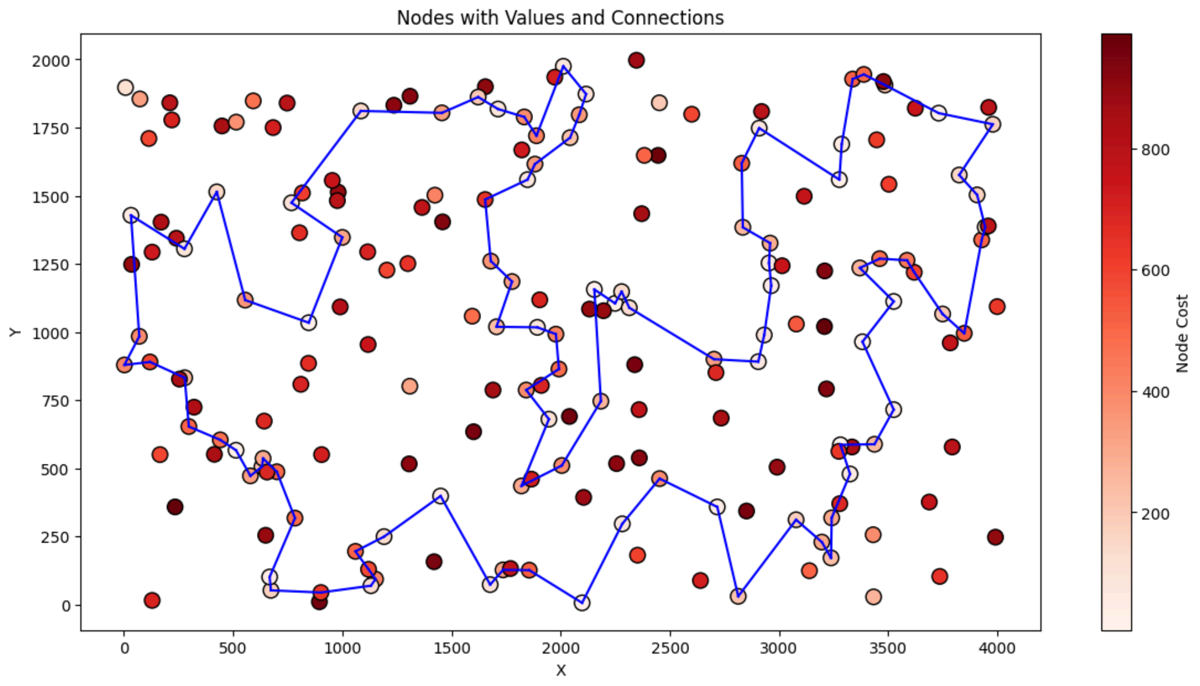

Fig 4. Visualization of the best solution found by the **Large Neighborhood Search (optional LS)** on the TSPB problem instance starting from a random solution.

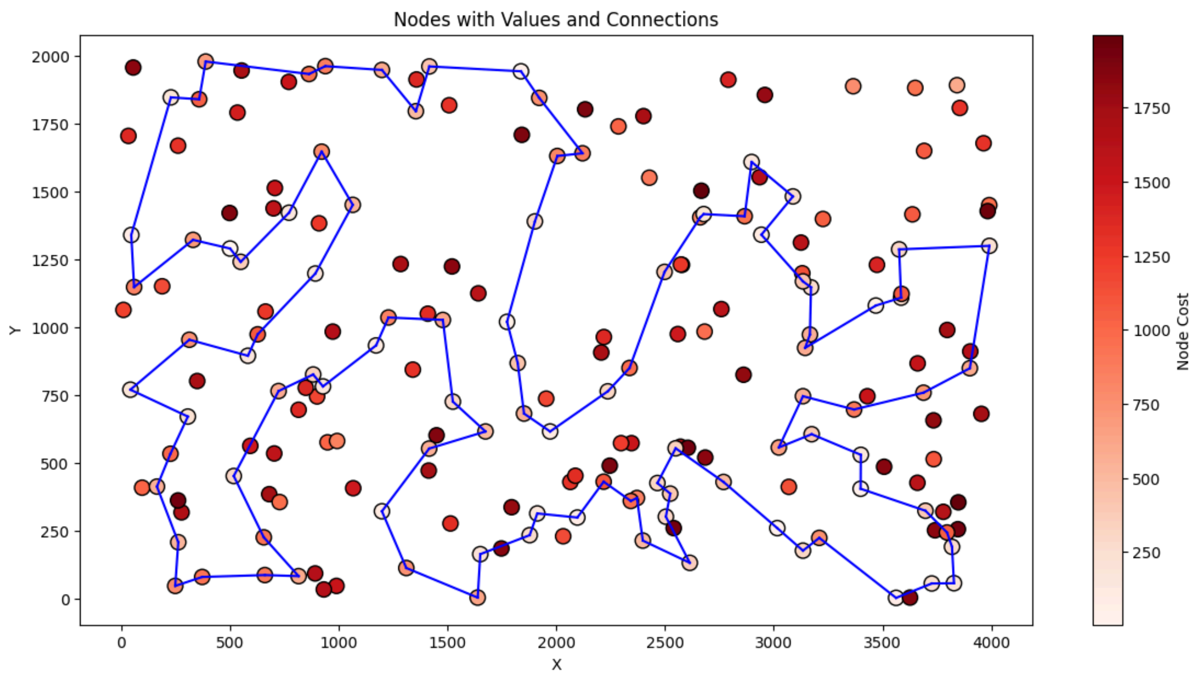

Fig 5. Visualization of the best solution found by the **Large Neighborhood Search (no LS)** on the TSPA problem instance starting from a random solution.

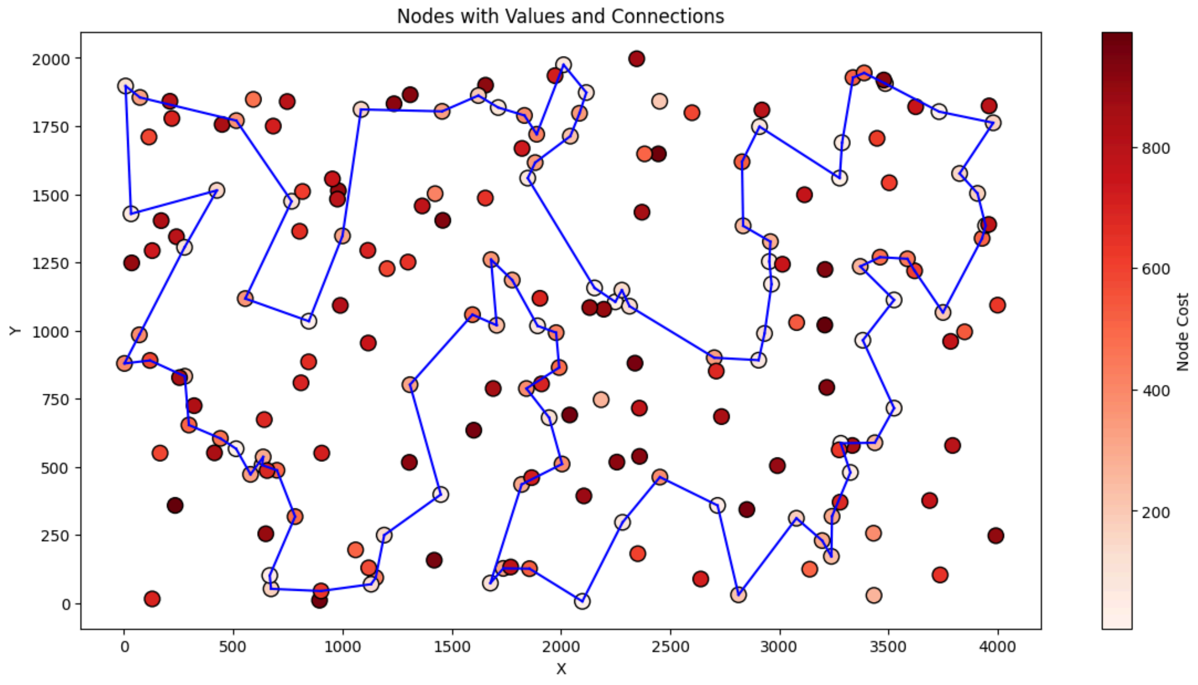

Fig 6. Visualization of the best solution found by the **Large Neighborhood Search (no LS)** on the TSPB problem instance starting from a random solution.

All best solutions were checked using the solution checker spreadsheet available on eKursy. The lists of node indices in the best solutions and their scores are presented in the table below.

Problem instance	Algorithm	Score	Solution
TSPA	**Large Neighborhood Search (optional LS)**	**69207**	**42, 43, 116, 65, 59, 118, 51, 151, 133, 162, 123, 127, 70, 135, 154, 180, 53, 121, 100, 26, 86, 75, 101, 1, 97, 152, 2, 120, 44, 25, 16, 171, 175, 113, 56, 31, 78, 145, 92, 129, 57, 179, 196, 81, 90, 165, 40, 185, 55, 52, 106, 178, 49, 14, 144, 102, 62, 9, 148, 124, 94, 63, 79, 80, 176, 137, 23, 186, 89, 183, 143, 0, 117, 93, 140, 68, 46, 115, 139, 41, 193, 159, 69, 108, 18, 22, 146, 181, 34, 160, 48, 54, 177, 10, 190, 4, 112, 84, 35, 18**
	Large Neighborhood Search (no LS)	**69250**	**186, 23, 137, 176, 80, 79, 63, 94, 124, 148, 9, 62, 102, 144, 14, 49, 178, 106, 52, 55, 185, 40, 165, 90, 81, 196, 179, 57, 129, 92, 145, 78, 31, 56, 113, 175, 171, 16, 25, 44, 120, 2, 152, 97, 1, 101, 75, 86, 26, 100, 121, 53, 180, 154, 135, 70, 127, 123, 162, 133, 151, 51, 118, 59, 65, 116, 43, 184, 84, 112, 4, 190, 10, 177, 54, 48, 160, 34, 181, 42, 5, 115, 46, 68, 139, 41, 193, 159, 146, 22, 18, 69, 108, 140, 93, 117, 0, 143, 183, 89**
	MSLS	70919	151, 162, 123, 127, 112, 4, 84, 184, 177, 54, 48, 160, 34, 181, 42, 43, 116, 65, 131, 149, 59, 46, 68, 139, 115, 41, 193, 159, 22, 18, 69, 108, 140, 93, 117, 0, 143, 183, 89, 23, 137, 148, 9, 62, 102, 49, 144, 14, 138, 3, 178, 106, 52, 55, 57, 185, 40, 119, 165, 39, 27, 90, 81, 196, 31, 113, 175, 171, 16, 25, 44, 120, 78, 145, 179, 92, 129, 2, 152, 97, 1, 101, 75, 86, 26, 100, 121, 53, 158, 180, 154, 70, 135, 133, 79, 63, 94, 80, 176, 51
	ILS - Partial Shuffle	69141	184, 84, 112, 4, 190, 10, 177, 54, 48, 160, 34, 146, 22, 18, 69, 108, 140, 93, 117, 0, 143, 183, 89, 186, 23, 137, 176, 80, 79, 63, 94, 124, 148, 9, 62, 102, 144, 14, 49, 178, 106, 52, 55, 57, 129, 92, 179, 185, 40, 165, 90, 81, 196, 145, 78, 31, 56, 113, 175, 171, 16, 25, 44, 120, 2, 152, 97, 1, 101, 75, 86, 26, 100, 53, 180, 154, 135, 70, 127, 123, 162, 133, 151, 51, 118, 59, 115, 46, 68, 139, 41, 193, 159, 181, 42, 43, 116, 65, 149, 131
	ILS - MST Perturbation	69246	9, 62, 102, 144, 14, 49, 178, 106, 52, 55, 185, 40, 165, 90, 81, 196, 145, 78, 31, 56, 113, 175, 171, 16, 25, 44, 120, 92, 57, 129, 2, 152, 97, 1, 101, 75, 86, 26, 100, 53, 180, 154, 135, 70, 127, 123, 162, 133, 151, 51, 118, 59, 115, 46, 68, 139, 41, 193, 159, 181, 42, 43, 116, 65, 149, 131, 35, 184, 84, 112, 4, 190, 10, 177, 54, 48, 160, 34, 146, 22, 18, 69, 108, 140, 93, 117, 0, 143, 183, 89, 186, 23, 137, 176, 80, 79, 63, 94, 124, 148
	ILS - Coordinate Change	69460	100, 53, 180, 154, 135, 70, 127, 123, 162, 133, 151, 51, 118, 59, 149, 131, 65, 116, 43, 42, 5, 115, 46, 68, 139, 41, 193, 159, 181, 160, 184, 84, 112, 4, 190, 10, 177, 54, 48, 34, 146, 22, 18, 108, 140, 93, 117, 0, 143, 183, 89, 186, 23, 137, 176, 80, 79, 63, 94, 124, 148, 9, 62, 144, 14, 49, 3, 178, 106, 52, 55, 185, 40, 165, 90, 81, 196, 31, 56, 113, 175, 171, 16, 78, 145, 179, 57, 92, 129, 25, 44, 120, 2, 152, 97, 1, 101, 75, 86, 26
TSPB	**Large Neighborhood Search (optional LS)**	**43873**	**33, 160, 144, 104, 8, 21, 82, 111, 29, 0, 109, 35, 143, 106, 124, 62, 18, 55, 34, 170, 152, 183, 140, 4, 149, 28, 20, 60, 148, 47, 94, 66, 172, 179, 22, 99, 130, 95, 185, 86, 166, 194, 176, 113, 103, 127, 89, 163, 187, 153, 81, 77, 141, 91, 61, 36, 177, 5, 45, 142, 78, 175, 162, 80, 190, 136, 73, 54, 31, 193, 117, 198, 156, 1, 16, 27, 38, 63, 135, 122, 131, 121, 51, 90, 147, 6, 188, 169, 132, 13, 70, 3, 15, 145, 195, 168, 43, 139, 11, 138**

	Large Neighborhood Search (no LS)	**44174**	**193, 54, 31, 73, 136, 190, 80, 162, 175, 78, 5, 177, 25, 182, 138, 139, 11, 33, 160, 144, 104, 8, 82, 21, 36, 61, 91, 141, 77, 81, 153, 187, 163, 89, 127, 103, 113, 176, 194, 166, 86, 185, 95, 130, 99, 22, 179, 66, 94, 47, 148, 60, 20, 28, 149, 4, 140, 183, 152, 170, 34, 55, 18, 62, 124, 106, 143, 35, 109, 0, 29, 168, 195, 145, 15, 3, 70, 13, 132, 169, 188, 6, 147, 51, 121, 131, 90, 133, 107, 40, 63, 122, 135, 38, 27, 16, 1, 156, 198, 117**
	MSLS	45365	10, 147, 6, 188, 169, 132, 13, 195, 168, 145, 15, 70, 3, 155, 184, 152, 170, 34, 55, 18, 62, 124, 106, 86, 95, 130, 183, 140, 199, 4, 149, 28, 20, 60, 148, 47, 94, 179, 22, 99, 185, 166, 194, 88, 176, 180, 113, 26, 103, 89, 114, 137, 127, 165, 163, 153, 81, 77, 141, 36, 61, 21, 82, 8, 111, 35, 109, 0, 29, 160, 33, 11, 139, 138, 182, 25, 177, 5, 142, 78, 175, 80, 190, 73, 54, 31, 193, 117, 198, 1, 38, 63, 135, 131, 121, 51, 191, 90, 122, 133
	ILS - Partial Shuffle	43448	81, 153, 187, 163, 103, 89, 127, 137, 114, 113, 176, 194, 166, 86, 185, 95, 130, 99, 22, 179, 66, 94, 47, 148, 60, 20, 28, 149, 4, 140, 183, 152, 170, 34, 55, 18, 62, 124, 106, 143, 35, 109, 0, 29, 111, 8, 104, 144, 160, 33, 138, 11, 139, 168, 195, 13, 145, 15, 3, 70, 132, 169, 188, 6, 147, 90, 51, 121, 131, 135, 122, 133, 107, 40, 63, 38, 27, 16, 1, 156, 198, 117, 193, 31, 54, 73, 136, 190, 80, 45, 142, 175, 78, 5, 177, 36, 61, 91, 141, 77
	ILS - MST Perturbation	43658	89, 127, 137, 114, 103, 113, 176, 194, 166, 86, 95, 130, 185, 179, 66, 94, 47, 148, 60, 20, 28, 149, 4, 140, 183, 152, 170, 34, 55, 18, 62, 128, 124, 106, 143, 35, 109, 0, 29, 111, 82, 8, 104, 144, 160, 33, 138, 182, 11, 139, 168, 195, 13, 145, 15, 3, 70, 132, 169, 188, 6, 147, 191, 90, 51, 121, 131, 135, 122, 107, 40, 63, 38, 27, 1, 156, 198, 117, 193, 31, 54, 73, 136, 190, 80, 45, 142, 175, 78, 5, 177, 36, 61, 91, 141, 77, 81, 153, 187, 163
	ILS - Coordinate Change	43574	81, 153, 187, 163, 103, 89, 127, 137, 114, 113, 176, 194, 166, 86, 185, 95, 130, 99, 22, 179, 66, 94, 47, 148, 60, 20, 28, 149, 4, 140, 183, 152, 170, 34, 55, 18, 62, 124, 106, 143, 35, 109, 0, 29, 111, 8, 104, 144, 160, 33, 138, 11, 139, 168, 195, 13, 145, 15, 3, 70, 132, 169, 188, 6, 147, 90, 51, 121, 131, 135, 122, 133, 107, 40, 63, 38, 27, 16, 1, 156, 198, 117, 193, 31, 54, 73, 136, 190, 80, 45, 142, 175, 78, 5, 177, 36, 61, 91, 141, 77

Table 2. Best solutions and their scores found by each algorithm in both instances.

Method	TSPA av (min - max) [s]	TSPB av (min - max) [s]
Greedy LS Rand	1.273 (1.047 - 1.975)	1.258 (0.991 - 1.646)
Steepest LS Rand	4.283 (3.261 - 6.218)	4.501 (3.292 - 5.609)
Greedy LS Edges Rand	1.171 (0.981 - 1.34)	1.113 (0.945 - 1.446)
Steepest LS Edges Rand	3.571 (2.978 - 4.168)	3.654 (2.976 - 4.364)
Greedy LS Best	0.067 (0.025 - 0.145)	0.077 (0.033 - 0.187)

Steepest LS Best	0.170 (0.055 - 0.529)	0.196 (0.09 - 0.746)
Greedy LS Edges Best	0.062 (0.025 - 0.115)	0.078 (0.036 - 0.212)
Steepest LS Edges Best	0.194 (0.078 - 0.379)	0.229 (0.114 - 0.836)
Steepest Candidate Search	0.584 (0.479 - 0.705)	0.562 (0.481 - 0.693)
MSLS	97.988 (92.859 - 104.159)	90.370 (83.360 - 98.981)
ILS	97.988	90.370
Large Neighborhood Search	**97.988**	**90.370**

Table 3. Minimum, average, and maximum run time achieved by local search methods on both problem instances.

4. Conclusions

The Large Neighborhood Search algorithm utilizing local search outperforms all other algorithms and even the version not using the local search afterwards performs surprisingly well. The main reason behind the success of the algorithm is most likely the combination of starting from a local optimum and the use of a greedy heuristic, both the MSLS on its own and the greedy regret weighted algorithms would have performed well on their own. However, in this algorithm, we combine both methods only updating the solution to be destroyed and fixed when a better one is found, this allows us to continuously search the space for better solutions. This also necessitates the random aspect of the destroy operator, without it, the algorithm could iterate over the same solutions, not effectively utilizing the given time limit. Furthermore, the addition of the local search in one of the versions further improves the algorithm performance, since the use of the destroy and fix mechanism works like a guided escape from the local optimum that still has a chance of being near another local optimum (as can be seen thanks to the relatively small number of changes of iterations, the repaired solution is likely still close to some optimum), this way the algorithm can achieve even better scores with fewer iterations, however, it can be seen that at this point, the improvements become less significant despite utilizing much more complex algorithms, this is likely because of the diminishing results, according to which even as we put much more thought behind the approach the improvements of the objective function may not be very significant.