Large Neighborhood Search for the Hamiltonian Cycle Problem

Evolutionary Computation Laboratory: Assignment 7

151927 Samuel Janas, samuel.janas@student.put.poznan.pl
151955 Bruno Urbaniak, bruno.urbaniak@student.put.poznan.pl
151960 Patryk Maciejewski, patryk.maciejewski.1@student.put.poznan.pl

1 Introduction

In this report, we extend our exploration of optimization methods for the Hamiltonian Cycle Problem (HCP) by implementing and evaluating a metaheuristic known as **Large Neighborhood Search (LNS)**. Specifically, we examine two versions of LNS: one that includes a local search phase after the destroy and repair operations (**LNS-LS**), and one that does not (**LNS-noLS**).

The goal is to assess whether these advanced methods can achieve better results than the previously implemented methods, including Multiple Start Local Search (MSLS), Iterated Local Search (ILS), and various greedy heuristics. The computational experiments involve evaluating both versions of LNS on the same datasets, ensuring comparability of results.

2 Problem Description and Implementation

2.1 Hamiltonian Cycle Problem Overview

The HCP involves finding a closed loop that visits a subset of nodes exactly once. In our study, the objective is to select 50% of the available nodes to form such a cycle while minimizing the combined objective function, comprising the total cost of selected nodes and the total distance of the cycle.

2.2 Implemented Methods

2.2.1 Large Neighborhood Search (LNS)

The LNS method is a metaheuristic that iteratively improves a solution by partially destroying it and then repairing it, potentially escaping local optima by exploring a larger neighborhood. The general idea is to remove a significant portion of the solution (e.g., 20%) and then reconstruct it using a heuristic method.

In our implementation, we consider two versions of LNS:

- **LNS without Local Search (LNS-noLS)**: After the destroy and repair steps, the solution is directly evaluated without any further local search.
- **LNS with Local Search (LNS-LS)**: After the repair step, a steepest local search is applied to the repaired solution to further improve it.

The methods are as follows:

- **Initialization**: Generate an initial random solution and apply local search if applicable.
- **Main Loop**:
  1. **Destroy**: Remove a percentage of nodes from the current solution to create a partial solution.
  2. **Repair**: Reconstruct the solution using the Nearest Neighbor heuristic with flexible insertion.

3. **Local Search (optional)**: For LNS-LS, apply steepest local search to the repaired solution.
4. **Acceptance Criterion**: If the new solution is better than the current one, accept it.

- **Termination**: Repeat the main loop until the average running time of MSLS from the previous assignment is reached.

2.3 Pseudocode for Implemented Algorithms

Problem 1: Large Neighborhood Search with Local Search (LNS-LS)

1. **Input**:
   - Cost matrix $costMatrix$
   - Time limit t (e.g., 24 seconds)
   - Destroy percentage p (e.g., 20%)
2. **Output**:
   - Best solution $bestSolution$
   - Best fitness $bestFitness$
3. **Procedure**:
   (i) Initialize $bestFitness \leftarrow \infty$
   (ii) Initialize $callCount \leftarrow 0$
   (iii) Start timer
   (iv) While time elapsed $< t$:
   - $startNode \leftarrow callCount \mod \text{length}(costMatrix)$
   - If $callCount = 0$:
     - $solution \leftarrow \text{RandomSteepestIntraEdge}(costMatrix, startNode)$
   - Else:
     - $solution \leftarrow bestSolution$
   - $callCount \leftarrow callCount + 1$
   - Destroy the solution:
     $$destroyedSolution \leftarrow \text{DestroySolution}(solution, p)$$
   - Repair the solution:
     $$repairedSolution \leftarrow \text{NearestNeighborFlexibleFromSolution}(costMatrix, destroyedSolution)$$
   - Apply local search:
     $$solution \leftarrow \text{SteepestIntraEdgeFromSolution}(repairedSolution, costMatrix, startNode)$$
   - Evaluate fitness $fitness \leftarrow \text{Fitness}(solution, costMatrix)$
   - If $fitness < bestFitness$:
     - $bestFitness \leftarrow fitness$
     - $bestSolution \leftarrow solution$

   (v) Return $bestSolution, bestFitness$

Problem 2: Large Neighborhood Search without Local Search (LNS-noLS)

1. **Input**:
   - Cost matrix $costMatrix$
   - Time limit t (e.g., 24 seconds)
   - Destroy percentage p (e.g., 20%)
2. **Output**:
   - Best solution $bestSolution$
   - Best fitness $bestFitness$
3. **Procedure**:
   (i) Initialize $bestFitness \leftarrow \infty$
   (ii) Initialize $callCount \leftarrow 0$
   (iii) Start timer
   (iv) While time elapsed $< t$:
   - $startNode \leftarrow callCount \mod \text{length}(costMatrix)$
   - If $callCount = 0$:
     – $solution \leftarrow \text{RandomSteepestIntraEdge}(costMatrix, startNode)$
   - Else:
     – $solution \leftarrow bestSolution$
   - $callCount \leftarrow callCount + 1$
   - Destroy the solution:
     $$destroyedSolution \leftarrow \text{DestroySolution}(solution, p)$$
   - Repair the solution:
     $$solution \leftarrow \text{NearestNeighborFlexibleFromSolution}(costMatrix, destroyedSolution)$$
   - Evaluate fitness $fitness \leftarrow \text{Fitness}(solution, costMatrix)$
   - If $fitness < bestFitness$:
     – $bestFitness \leftarrow fitness$
     – $bestSolution \leftarrow solution$

   (v) Return $bestSolution, bestFitness$

Problem 3: DestroySolution

1. **Input**:
   - Current solution $solution$
   - Destroy percentage p
2. **Output**:
   - Destroyed solution $destroyedSolution$
3. **Procedure**:

(i) Compute the number of nodes to remove:

$$k \leftarrow \lfloor p \times \text{length}(solution) \rfloor$$

(ii) If $k = 0$, return *solution*

(iii) Divide k into three group sizes $[g_1, g_2, g_3]$ as evenly as possible.

(iv) Initialize $modifiedSolution \leftarrow solution$

(v) For each group size g in $[g_1, g_2, g_3]$:

- If $g > 0$:
  - Randomly select a starting index s in $modifiedSolution$ such that $s + g \leq \text{length}(modifiedSolution)$
  - Remove g nodes from $modifiedSolution$ starting at index s:

$$modifiedSolution \leftarrow modifiedSolution[: s] + modifiedSolution[s + g :]$$

(vi) Return $modifiedSolution$

3 Computational Experiment

3.1 Experiment Setup

- **Methods Compared:**
  - Large Neighborhood Search without Local Search (**LNS-noLS**)
  - Large Neighborhood Search with Local Search (**LNS-LS**)
- **Instances Used:**
  - Instance A (TSPA)
  - Instance B (TSPB)
- **Metrics Recorded:**
  - Best, worst, and average objective function values.
  - Number of iterations (calls) in each run.

3.2 Execution Time Setup

We used the average runtime from the previous MSLS experiments as the time limit for our LNS methods. Specifically, the time limit t was set to approximately 24 seconds for both instances.

3.3 Number of Iterations in LNS

Table 1: Number of Iterations (Calls) for LNS-noLS and LNS-LS on TSPA and TSPB

Run	TSPA LNS-noLS Calls	TSPA LNS-LS Calls	TSPB LNS-noLS Calls	TSPB LNS-LS Calls
1	47440	9770	48363	13731
2	46873	10010	47229	10842
3	46641	10003	46930	12573
4	46962	8435	47128	10787
5	47219	9370	47025	12518
6	43645	8948	43478	11550
7	41374	7598	43505	10531
8	40337	7176	43906	11879
9	41245	7005	43587	12701
10	42575	6808	43287	12719
11	40532	7435	42662	11516
12	40411	7636	42843	10758
13	39166	8935	42588	12106
14	41020	9125	42680	11063
15	41910	9498	41513	10801
16	40673	8279	42154	12719
17	42733	8831	42824	10551
18	43523	9834	38863	12118
19	40351	8468	33799	11820
20	39166	8935	44026	11131

3.4 Results

3.4.1 Objective Function Values

Table 2: Objective Function Values for All Methods and Instances, Sorted by Average Fitness

Method	**Instance A (TSPA)**	**Instance B (TSPB)**
Random Solution	264724.25 (225047 – 290346)	213180.80 (190200 – 235839)
Nearest Neighbor (End Insertion)	85108.51 (83182 – 89433)	54390.43 (52319 – 59030)
Nearest Neighbor (Flexible Insertion)	73178.55 (71179 – 75450)	45870.25 (44417 – 53438)
Greedy Cycle	72634.87 (71488 – 74410)	51397.91 (49001 – 57262)
Greedy Heuristic (Weighted Sum)	73762.84 (71544 – 76341)	50992.64 (46990 – 58454)
Greedy 2-Regret Heuristic	73731.69 (71809 – 76323)	50794.27 (45814 – 59121)
LS Random Steepest Intranode	88159.52 (80186 – 96426)	63017.28 (55773 – 70444)
LS Random Greedy Intranode	86187.72 (79078 – 93575)	61026.47 (54087 – 69709)
LS Random Steepest Intraedge	73863.17 (71355 – 79486)	48364.52 (45452 – 51331)
LS Random Greedy Intraedge	73836.13 (71571 – 77616)	48330.82 (45905 – 52361)
LS Nearest Neighbour Flexible Greedy Intranode	72785.85 (71034 – 74904)	45450.70 (43826 – 50886)
LS Nearest Neighbour Flexible Greedy Intraedge	71173.25 (69997 – 73545)	45021.37 (43790 – 50495)
LS Nearest Neighbour Flexible Steepest Intranode	72805.67 (71034 – 74904)	45414.50 (43826 – 50876)
LS Nearest Neighbour Flexible Steepest Intraedge	70972.69 (69864 – 73068)	44976.43 (43921 – 50495)
Steepest LS with Candidate Moves (k = 10)	74433.93 (71729 – 78375)	48741.32 (46322 – 52626)
Steepest LS with Move Evaluations	74056.84 (70801 – 78688)	48425.14 (45605 – 51662)
MSLS	71326.70 (70802 – 71851)	45798.30 (45207 – 46236)
ILS	70386.50 (69836 – 70891)	45048.30 (44449 – 45803)
LNS-noLS	69282.60 (69107 – 69593)	43929.15 (43550 – 44399)
LNS-LS	69197.25 (69102 – 69593)	43802.10 (43446 – 44319)

4 2D Visualization of the Best Solutions

4.1 Instance A Best Solution Visualizations

4.1.1 LNS-noLS for Instance A

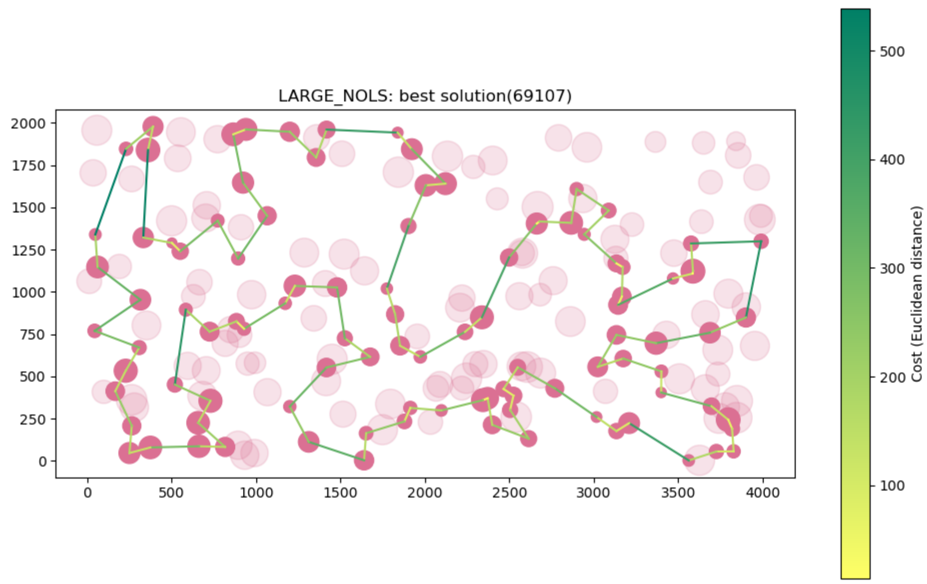

Figure 1: Best solution for Instance A (TSPA) using LNS-noLS.

4.1.2 LNS-LS for Instance A

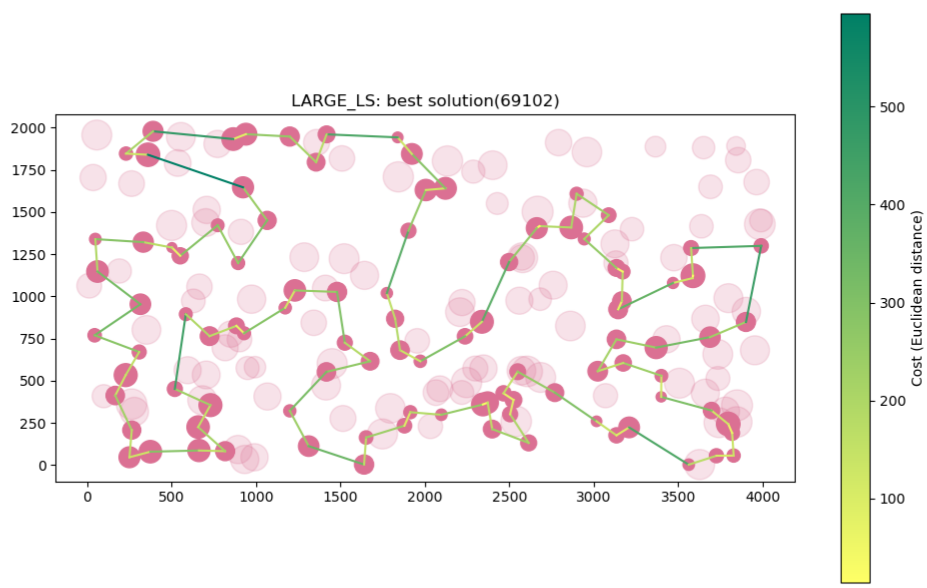

Figure 2: Best solution for Instance A (TSPA) using LNS-LS.

4.2 Instance B Best Solution Visualizations

4.2.1 LNS-noLS for Instance B

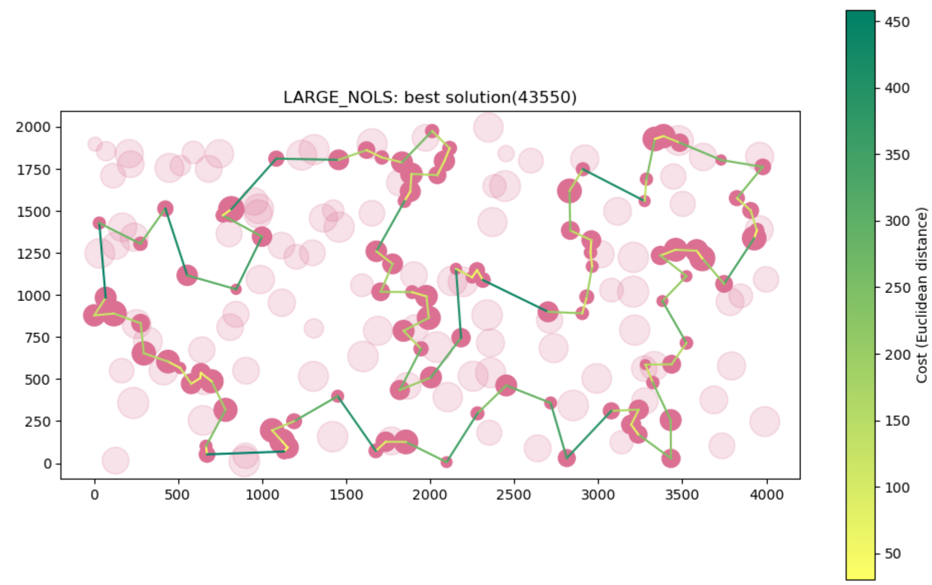

Figure 3: Best solution for Instance B (TSPB) using LNS-noLS.

4.2.2 LNS-LS for Instance B

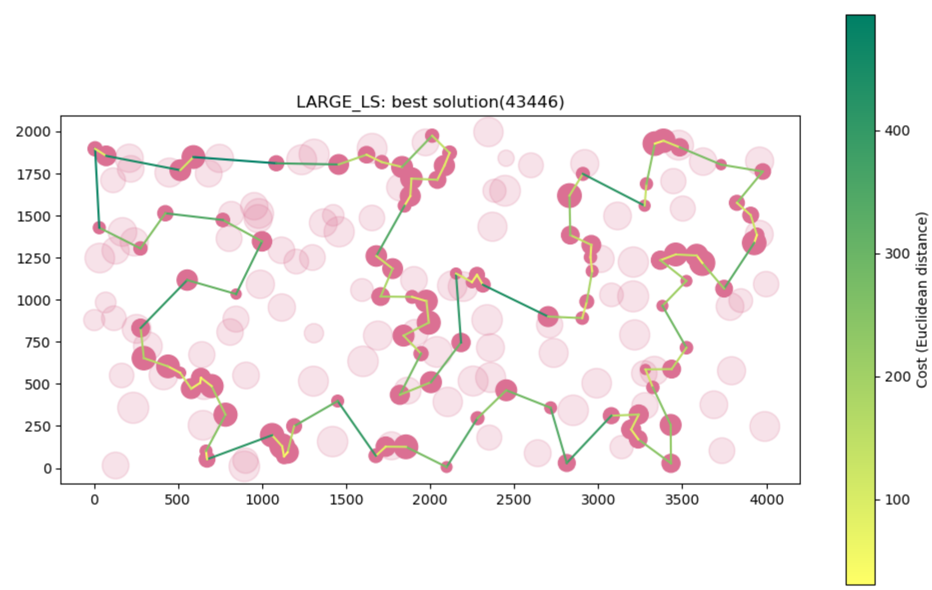

Figure 4: Best solution for Instance B (TSPB) using LNS-LS.

4.2.3 Best Solutions for Each Method

Table 3: Best Solutions for Instances A and B by Method

Instance	Method	Best Solution
TSPA	LNS-noLS	[148, 9, 62, 102, 144, 14, 49, 178, 106, 52, 55, 185, 40, 119, 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, 53, 180, 154, 135, 70, 127, 123, 162, 133, 151, 51, 118, 59, 65, 116, 43, 42, 184, 35, 84, 112, 4, 190, 10, 177, 54, 48, 160, 34, 181, 146, 22, 18, 108, 69, 159, 193, 41, 139, 115, 46, 68, 140, 93, 117, 0, 143, 183, 89, 186, 23, 137, 176, 80, 79, 63, 94, 124]
TSPB	LNS-noLS	[80, 190, 136, 73, 54, 31, 193, 117, 198, 156, 1, 16, 27, 38, 63, 135, 122, 131, 121, 51, 90, 191, 147, 6, 188, 169, 132, 70, 3, 15, 145, 13, 195, 168, 139, 11, 138, 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, 179, 22, 99, 130, 95, 185, 86, 166, 194, 176, 113, 114, 137, 127, 89, 103, 163, 187, 153, 81, 77, 141, 91, 61, 36, 177, 5, 45, 142, 78, 175]
TSPA	LNS-LS	[68, 46, 115, 139, 41, 193, 159, 22, 146, 181, 34, 160, 48, 54, 177, 10, 190, 4, 112, 84, 35, 184, 42, 43, 116, 65, 59, 118, 51, 151, 133, 162, 123, 127, 70, 135, 154, 180, 53, 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, 119, 40, 185, 55, 52, 106, 178, 49, 14, 144, 102, 62, 9, 148, 124, 94, 63, 79, 80, 176, 137, 23, 183, 89, 186]
TSPB	LNS-LS	[77, 141, 91, 61, 36, 177, 5, 78, 175, 142, 45, 80, 190, 136, 73, 54, 31, 193, 117, 198, 156, 1, 131, 121, 51, 90, 122, 135, 63, 40, 107, 133, 10, 147, 6, 188, 169, 132, 70, 3, 15, 145, 13, 195, 168, 139, 11, 138, 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, 179, 22, 99, 130, 95, 185, 86, 166, 194, 176, 113, 114, 137, 127, 89, 103, 163, 187, 153, 81]

5 Conclusion

This study demonstrates the effectiveness of the Large Neighborhood Search (LNS) method in solving the Hamiltonian Cycle Problem. Both versions of LNS outperformed previously implemented methods, achieving lower average objective function values. The inclusion of local search in LNS-LS provided marginal improvements over LNS-noLS, suggesting that the local search phase contributes to finding better solutions. The results indicate that LNS is a valuable addition to the suite of heuristics for tackling combinatorial optimization problems like the HCP.