Move Evaluations in Local Search for the Hamiltonian Cycle Problem

Evolutionary Computation Laboratory: Assignment 5

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 explore the integration of move evaluations (deltas) from previous iterations in the **Steepest Local Search** algorithm to enhance computational efficiency for the Hamiltonian Cycle Problem (HCP). The focus is on leveraging precomputed deltas to reduce redundant computations while maintaining solution quality. Building on insights from previous assignments, we examine both inter-route and intra-route moves and implement a mechanism for handling move validity across iterations.

2 Problem Description and Implementation

The Hamiltonian Cycle Problem involves finding a closed loop that visits a subset of nodes exactly once. Our objective remains to select 50% of the available nodes while minimizing the combined objective function comprising node costs and cycle distances.

2.1 Move Evaluations (Deltas)

The primary goal of this task is to improve time efficiency by maintaining a list of improving moves (LM) and reusing delta evaluations across iterations. Key considerations include:

- **Move Validity**: Moves are removed or marked invalid if edges involved in the move no longer exist or have changed direction.
- **Updated Moves**: When edges remain unchanged, moves are reapplied using precomputed deltas.
- **Neighborhood Structure**: The neighborhood includes inter-route moves (node exchanges) and intra-route moves (edge exchanges).

2.2 Implementation Details

The algorithm was implemented in **Go**, extending the Steepest Local Search algorithm. Key aspects include:

- **Delta Management**: Maintaining and updating a list of deltas for all candidate moves.
- **Move Validation**: Ensuring only valid moves are considered by checking edge consistency.
- **Algorithm Efficiency**: Reducing computational overhead by leveraging precomputed deltas wherever possible.

3 Pseudocode of the Algorithm

Problem 1: Steepest Local Search with Move Evaluations

1. **Input**:
   - Current solution $solution$
   - List of unselected nodes $unselectedNodes$
   - Distance matrix $distanceMatrix$
   - Precomputed moves with deltas $moves$
   - Maximum number of iterations $maxIterations$
2. **Output**:
   - Updated solution $solution$
   - Final fitness value $fitness$
3. **Procedure**:
   (i) **Initialization**:
   - Compute the initial fitness value:

$$fitness \leftarrow \text{ComputeFitness}(solution, distanceMatrix)$$

   - Initialize improvement flag and iteration counter:

$$improved \leftarrow \text{true}, \quad iteration \leftarrow 0$$

   (ii) **Iterative Improvement**: While $improved = \text{true}$ and $iteration < maxIterations$:
   (a) **Find Best Move**:
   - Select the move with the smallest delta:

$$bestMove \leftarrow \text{argmin}_{m \in moves}\ m.\delta$$

   - Retrieve its delta value:

$$bestDelta \leftarrow bestMove.\delta$$

   - If $bestDelta \geq 0$:
     – Set $improved \leftarrow$ false and exit the loop.

   (b) **Apply Best Move**:
   - Update the solution by applying $bestMove$:

$$solution \leftarrow \text{ApplyMove}(solution, bestMove, unselectedNodes)$$

   - Remove $bestMove$ from the list of moves:

$$moves \leftarrow moves \setminus \{bestMove\}$$

   (c) **Update Moves**:
   - For each $move \in moves$, validate and update the delta:
     – If edges involved in $move$ are no longer valid:

$$moves \leftarrow moves \setminus \{move\}$$

     – If edges remain valid but have changed direction, skip updating the delta.

– Otherwise, recompute $move.\delta$ using the appropriate delta function.

(d) **Sort Moves**: Re-sort $moves$ by δ in ascending order.

(e) Increment the iteration counter:

$$iteration \leftarrow iteration + 1$$

(iii) **Post-Processing**:

- Recompute the final fitness value:

$$fitness \leftarrow \text{ComputeFitness}(solution, distanceMatrix)$$

4. **Return**: $solution, fitness$

Problem 2: Supporting Function: GenerateAllMoves

1. **Input**:
   - Current solution $solution$
   - List of unselected nodes $unselectedNodes$
2. **Output**:
   - List of all possible moves $moves$
3. **Procedure**:

(i) Initialize $moves \leftarrow []$.

(ii) Generate intra-route moves:

- For each pair of nodes $i, j \in solution$:

$$\text{Add Move}(\text{type} = \text{“twoNodesExchange”}, i, j)$$

- For each pair of non-adjacent nodes $i, j \in solution$:

$$\text{Add Move}(\text{type} = \text{“twoEdgesExchange”}, i, j)$$

(iii) Generate inter-route moves:

- For each node $i \in solution$ and $j \in unselectedNodes$:

$$\text{Add Move}(\text{type} = \text{“interRouteExchange”}, i, j)$$

(iv) Return $moves$.

Problem 3: Supporting Function: ApplyMove

1. **Input**:
   - Current solution $solution$
   - Move to be applied $move$
   - List of unselected nodes $unselectedNodes$
2. **Output**:
   - Updated solution $solution$

3. **Procedure**:

   (i) If *move*.type = "twoNodesExchange":
   - Swap *move.i* and *move.j* in *solution*.

   (ii) If *move*.type = "twoEdgesExchange":
   - Reverse the segment between *move.i* and *move.j* in *solution*.

   (iii) If *move*.type = "interRouteExchange":
   - Replace *move.i* in *solution* with *move.j* from $unselectedNodes$.

   (iv) Update $unselectedNodes$ to reflect the applied move.

   (v) Return *solution*.

4 Computational Experiment

4.1 Experiment Setup

The computational experiments were conducted with the following setup:

- **Methods Compared**:
  - Steepest Local Search with Move Evaluations.
  - Baseline Steepest Local Search without Move Evaluations.
- **Instances**: Provided datasets A and B.
- **Number of Runs**: 200 for each method on each instance.

4.2 Results of Computational Experiment

4.2.1 Objective Function Values

Table 1: Objective Function Values for Each Method and Instance

Method	Instance A	Instance B
Baseline Steepest LS	73863.17 (71355 – 79486)	48364.52 (45452 – 51331)
Steepest LS with Move Evaluations	74056.84 (70801 – 78688)	48425.14 (45605 – 51662)

4.2.2 Running Times

Table 2: Running Times for Each Method and Instance

Method	Instance A (sec)	Instance B (sec)
Steepest LS with Move Evaluations	0.136–0.203 (0.163)	0.139–0.205 (0.165)
Baseline Steepest LS	0.364–0.619 (0.446)	0.343–0.668 (0.413)

4.3 Analysis of Results

- **Solution Quality**: The performance of Steepest LS with Move Evaluations was comparable to the baseline, with slightly higher average fitness values.

- **Computational Efficiency**: The inclusion of move evaluations resulted in longer execution times due to the additional overhead of managing deltas.
- **Best Use Case**: This approach is best suited for problems where evaluating new moves is computationally expensive.

5 Visualization of Best Solutions

5.1 Instance A Best Solution

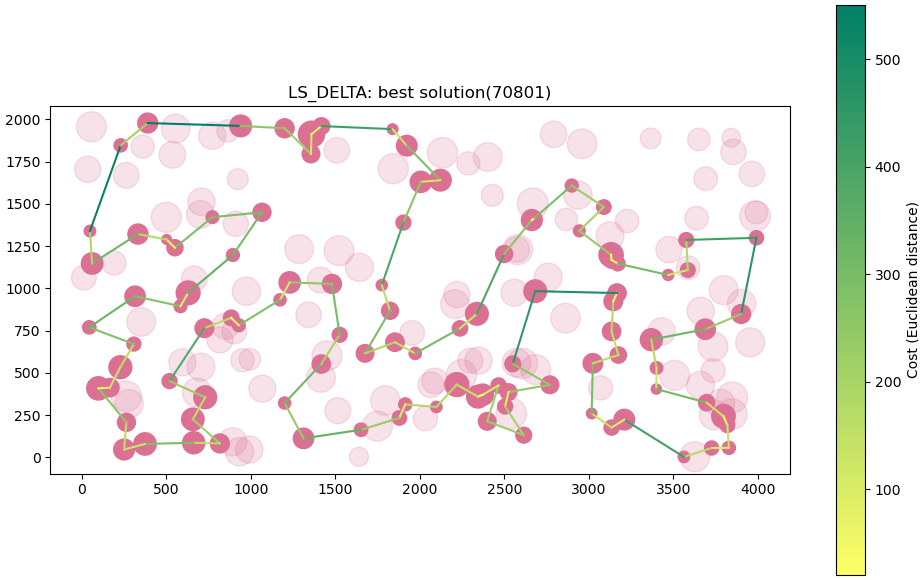

Figure 1: Best solution for Steepest LS with Move Evaluations on Instance A

5.2 Instance B Best Solution

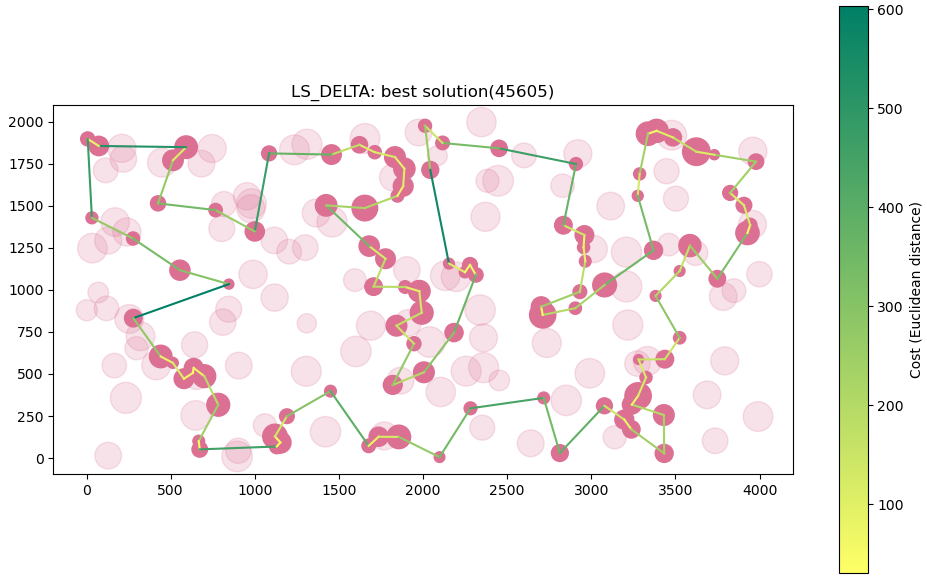

Figure 2: Best solution for Steepest LS with Move Evaluations on Instance B

6 Conclusion

This report introduces an enhanced **Steepest Local Search** algorithm with move evaluations for solving the Hamiltonian Cycle Problem. By reusing precomputed deltas, we aimed to improve time efficiency while maintaining solution quality. Although this approach introduces additional overhead, it demonstrates potential in scenarios with high computational costs for move evaluation.