Hybrid Evolutionary Algorithm

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

This project describes the implementation of a Hybrid Evolutionary Algorithm (Hybrid EA) and its comparison with previously implemented Multi-Start Local Search (MSLS), Iterated Local Search (ILS), and Large Neighborhood Search (LNS). The Hybrid EA aims to integrate the strengths of evolutionary and local search methods to optimize solutions for the Traveling Salesman Problem (TSP).

The main characteristics of this approach include:

- Maintaining an elite population to ensure diversity and high-quality solutions.
- Application of novel recombination operators to explore solution spaces effectively.
- Integration of local search mechanisms to refine solutions after recombination.
- A focus on diversification and prevention of premature convergence through the elimination of duplicate solutions.

2 Method Description

2.1 Algorithm Parameters

The Hybrid EA operates based on the following key parameters:

- **Elite Population Size**: 20 individuals are maintained in the elite population.
- **Generations per Run**: Each run is allowed a maximum of 200 generations.
- **Recombination Operators**: Two operators are used:
  - Operator 1 locates common nodes and edges in parents and fills the rest at random.
  - Operator 2 starts from a parent, removes edges absent in the other parent, and repairs the solution heuristically.
- **Execution Time Limit**: The algorithm runs for a maximum of 24 seconds per instance.

2.2 Pseudocode

Problem 1: Hybrid Evolutionary Algorithm

1. **Input**:
   - Cost matrix $costMatrix$.
   - Number of nodes n.
2. **Output**:

- Best solution $bestSolution$.
- Best fitness $bestFitness$.

3. **Procedure**:

   (i) Set the maximum number of generations and the execution time limit.

   (ii) Initialize an elite population with random solutions using a local search algorithm and evaluate them.

   (iii) **While** time elapsed is less than time limit:

      i. Select two parents from the population with uniform probability.

      ii. Apply one of the recombination operators to create an offspring.

      iii. Refine the offspring using a local search mechanism.

      iv. Evaluate the fitness of the offspring.

      v. If the offspring is not a duplicate, replace the worst solution in the population.

   (iv) Return the best solution found in the elite population.

Problem 2: Recombine Operator 1

1. **Input**:

   - $parent1$, $parent2$: Two parent solutions (arrays of integers).

2. **Output**:

   - A new hybrid solution as a child combining elements of $parent1$ and $parent2$.

3. **Procedure**:

   (i) Create an empty array $child$ of size equal to $parent1$.

   (ii) Create a map $inChild$ to track nodes already added to $child$.

   (iii) For each index i in $child$:

      (a) If $parent1[i]$ is in $parent2$, set $child[i] \leftarrow parent1[i]$ and mark $parent1[i]$ as added in $inChild$.

      (b) Otherwise, set $child[i] \leftarrow -1$ (indicating an unfilled position).

   (iv) For each index i in $child$ where $child[i] = -1$:

      (a) Repeat:

         (i) Randomly select a node from $parent1$ or $parent2$.

         (ii) If the selected node is not in $inChild$, add the node to $child[i]$ and mark it in $inChild$, then break the loop.

   (v) Return a new hybrid solution with $child$.

Problem 3: Recombine Operator 2

1. **Input**:

   - $parent1$, $parent2$: Two parent solutions (arrays of integers).
   - $costMatrix$: A matrix containing the pairwise costs or distances between nodes.

2. **Output**:

- A new hybrid solution as a repaired child based on common nodes and cost minimization.

3. **Procedure**:

   (i) Initialize an empty list *commonNodes*.

   (ii) For each node in *parent*1:

      (a) If the node is present in *parent*2, add it to *commonNodes*.

   (iii) Perform a nearest-neighbor repair starting from *commonNodes*:

      (a) Use the function `NearestNeighborFlexibleFromSolution(costMatrix, commonNodes)` to repair the solution.

   (iv) Return a new hybrid solution with the repaired child.

3 Results

3.1 Comparison with Previous Methods

Table 1: Objective Function Values for All Methods and Instances, Sorted by Average Fitness on TSPA

Method	**Instance A (TSPA)**	**Instance B (TSPB)**
LS Random Steepest Intranode	88159.52 (80186 – 96426)	63017.28 (55773 – 70444)
LS Random Greedy Intranode	86187.72 (79078 – 93575)	61026.47 (54087 – 69709)
Nearest Neighbor (End Insertion)	85108.51 (83182 – 89433)	54390.43 (52319 – 59030)
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)
LS Random Steepest Intraedge	73863.17 (71355 – 79486)	48364.52 (45452 – 51331)
LS Random Greedy Intraedge	73836.13 (71571 – 77616)	48330.82 (45905 – 52361)
Greedy Heuristic (Weighted Sum)	73762.84 (71544 – 76341)	50992.64 (46990 – 58454)
Greedy 2-Regret Heuristic	73731.69 (71809 – 76323)	50794.27 (45814 – 59121)
Nearest Neighbor (Flexible Insertion)	73178.55 (71179 – 75450)	45870.25 (44417 – 53438)
LS Nearest Neighbour Flexible Steepest Intranode	72805.67 (71034 – 74904)	45414.50 (43826 – 50876)
LS Nearest Neighbour Flexible Greedy Intranode	72785.85 (71034 – 74904)	45450.70 (43826 – 50886)
Greedy Cycle	72634.87 (71488 – 74410)	51397.91 (49001 – 57262)
MSLS	71326.70 (70802 – 71851)	45798.30 (45207 – 46236)
LS Nearest Neighbour Flexible Greedy Intraedge	71173.25 (69997 – 73545)	45021.37 (43790 – 50495)
LS Nearest Neighbour Flexible Steepest Intraedge	70972.69 (69864 – 73068)	44976.43 (43921 – 50495)
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)
HybridEA	69583.55 (69139 – 69952)	43623.95 (43446 – 44200)

3.2 Visualizations

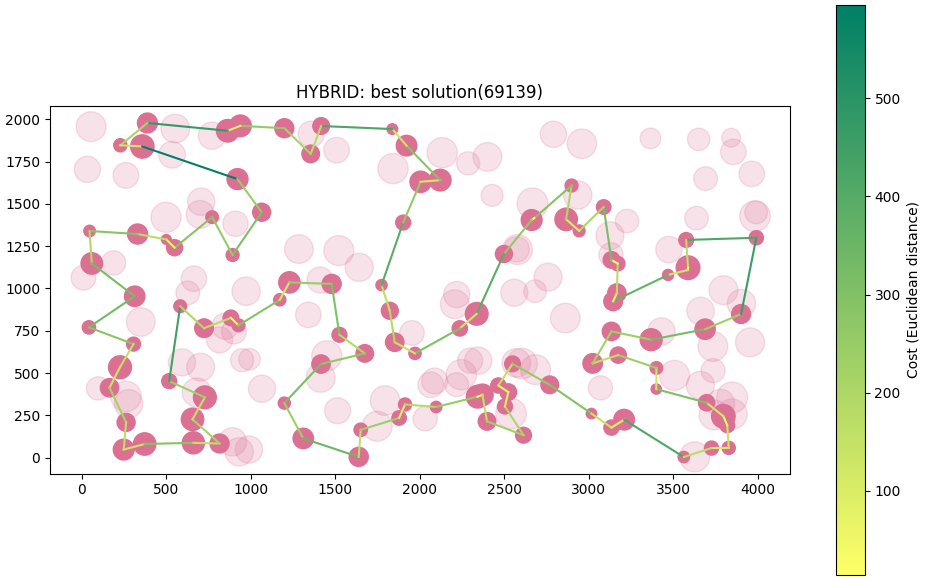

Figure 1: Best solution for Instance A (TSPA) using HybridEA. Fitness = 69139.

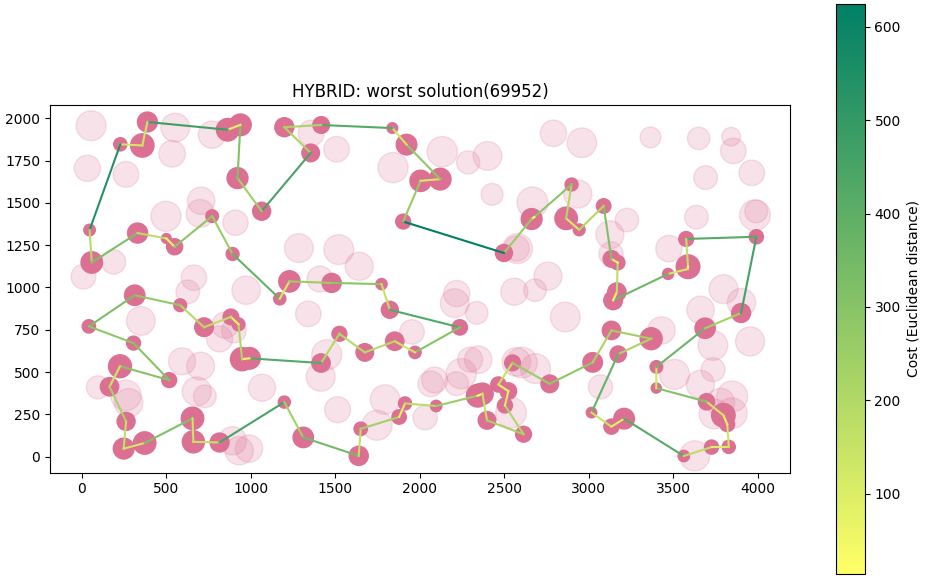

Figure 2: Worst solution for Instance A (TSPA) using HybridEA. Fitness = 69952.

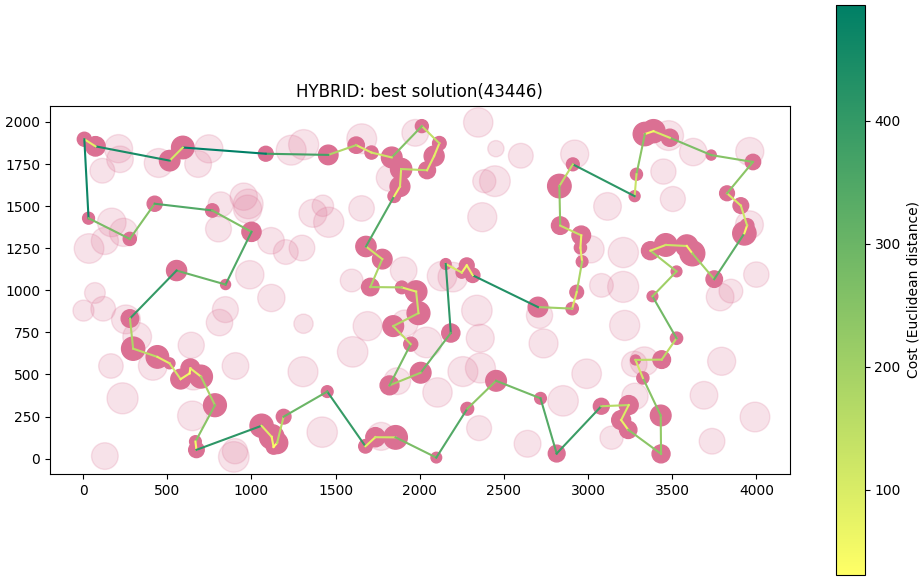

Figure 3: Best solution for Instance B (TSPB) using HybridEA. Fitness = 43446.

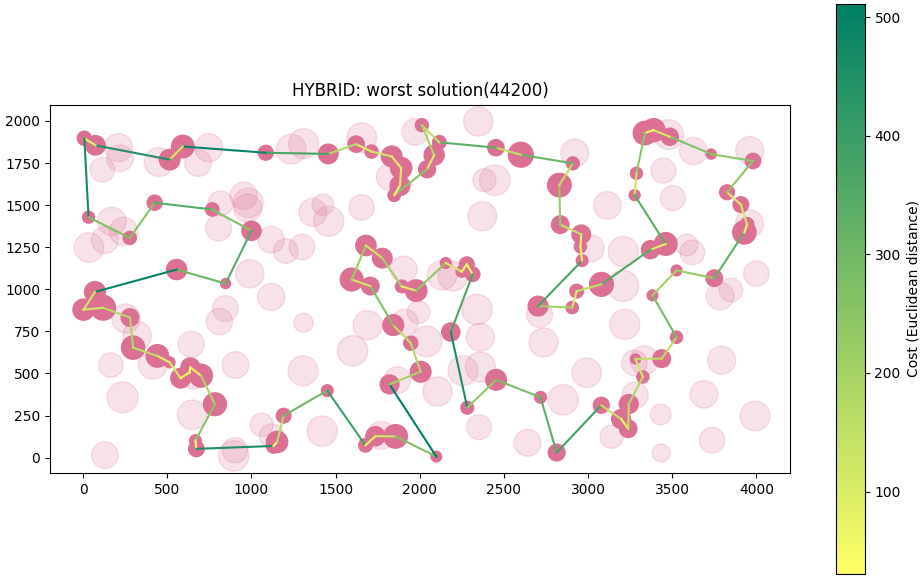

Figure 4: Worst solution for Instance B (TSPB) using HybridEA. Fitness = 44200.

4 Discussion

The Hybrid EA method showcased promising results for the Hamiltonian Cycle Problem (HCP), particularly excelling in the TSPB instance. With an average fitness and a best fitness , it outperformed LNS-LS in terms of average and matched best results, underscoring the robustness and adaptability of its design.

However, for the TSPA instance, the method achieved an average fitness, which, while competitive, slightly underperformed compared to the best average result obtained by LNS-LS. This discrepancy highlights a potential trade-off where the enhanced exploration mechanisms of Hybrid EA may reduce performance on easier instances due to over-diversification.

Key advantages of the Hybrid EA method include:

- **Destruction and Repair Mechanisms**: These operators introduced diversity in the solution pool while leveraging the underlying structure of the problem for efficient repair.
- **Evolutionary Operators**: The combination of crossover and mutation techniques ensured effective exploration and exploitation of the search space, preventing premature convergence.
- **Integration with Local Search**: The addition of steepest intraedge local search enabled fine-tuning of candidate solutions, leading to consistent performance improvements.

5 Conclusion

In conclusion, while Hybrid EA demonstrates significant strengths on some problem instances by leveraging destruction-repair mechanisms, evolutionary operators, and local search, its performance on other instances suggests a need for further calibration. Refining its parameterization to adapt dynamically to instance complexity could further enhance its effectiveness across a broader range of problems.