Heuristic and Meta-Heuristic Search to Solve Cycle Finding Problem

Presented by: Khennaoui Mohamed Seif

November 9, 2025

Contents

List of Figures

In this lab, we are going to explore the effectiveness of a local search method using the hill-climbing algorithm and a global stochastic method known as the genetic algorithm.

1 Introduction

In computer science, particularly in artificial intelligence, the complexity of the computation and the quality of algorithms are the two main criteria for the evaluation. The goal is to obtain exact and robust solutions with reasonable time complexity. However, for certain problems, especially non-linear and discrete disconnected problems, achieving exact solutions is challenging. Numerical methods such as gradient descent and Newton's method are often used for nonlinear problems. For disconnected, non-convex problems, methods like branch-and-bound or cutting planes are proposed, though these can be resource-intensive.

To obtain approximate solutions with reasonable complexity, heuristic methods are preferred. Both local and global methods are utilized for solving NP-hard problems such as the Traveling Salesman Problem (TSP).

2 Definition of Our Problem

The Traveling Salesman Problem (TSP) is defined as follows: "Given a list of cities and the distances between each pair of cities, what is the shortest route that visits each city exactly once and returns to the original city?"

The TSP is an NP-hard problem. This does not mean that every instance of the problem is difficult to solve. Rather, it implies that there does not exist an algorithm capable of producing the optimal solution in polynomial time for all instances. Consequently, we cannot predict how long it might take to find the best solution.

Instead, we aim to find a good solution that may not necessarily be the best. For example, it is acceptable to find a route among 100 cities that is only a few miles longer than the optimal route, particularly if obtaining the exact solution would require an inordinate amount of computing time.

Figure 1: **the TSP problem**

The problem under consideration is a constrained variant of the Traveling Salesman Problem (TSP). In this formulation, we introduce an additional constraint while relaxing

one of the original conditions. Specifically, the objective is to determine a cycle that connects all cities.

In this case, not all cities are directly connected. A connection between two cities is feasible only if the distance between them is less than 90 km, within a bounded space of 500 × 500 km. This constrained formulation remains an NP-hard problem.

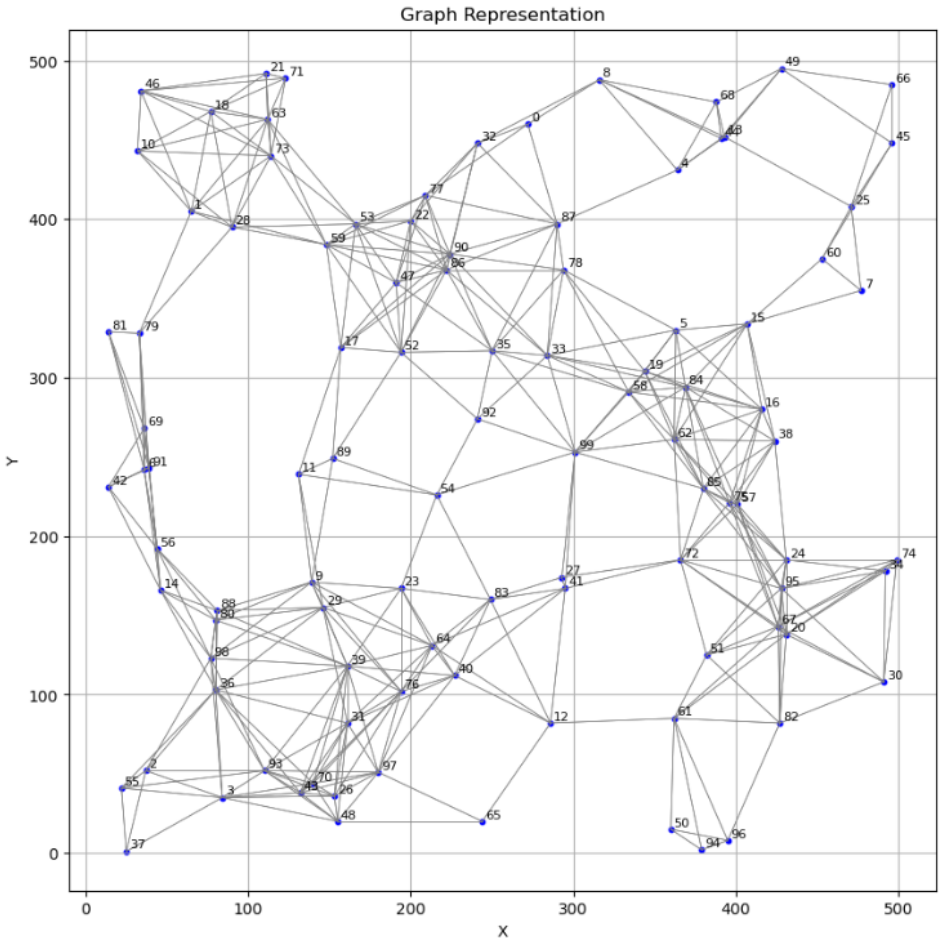

Figure 2: **representation of the problem space(the graph)**

This graph is represented by a function or algorithm that takes as input the coordinates of the cities. These coordinates can be generated randomly or read from a file. The algorithm produces as output a graph in which connections between cities adhere to the constraint that the distance between connected cities is less than 90 km.

The metric used to calculate the distances is the Euclidean distance, determined by the formula:

$$\text{Distance} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{1}$$

where (x_i, y_i) and (x_j, y_j) are the coordinates of two cities i and j. If the distance is less than 90 km, the cities are connected.

3 Resolution of the Problem

3.1 Local Search Method by Hill Climbing Algorithm

Hill climbing is a local search method based on a heuristic. It begins with an initial solution, which can be either generated randomly or obtained using a greedy algorithm. The method is considered "local" because it explores solutions within the neighborhood of the current solution. If a better solution is found, the algorithm moves to it; otherwise, it remains in the current state or solution. The evaluation of different states or solutions is based on the heuristic.

A heuristic serves as a guide or indication toward the optimal solution and is used to evaluate various solutions.

In the context of our problem, a state is defined as a cycle consisting of 100 unique nodes or cities, ensuring that all cities are visited. The length of each cycle varies depending on the solution.

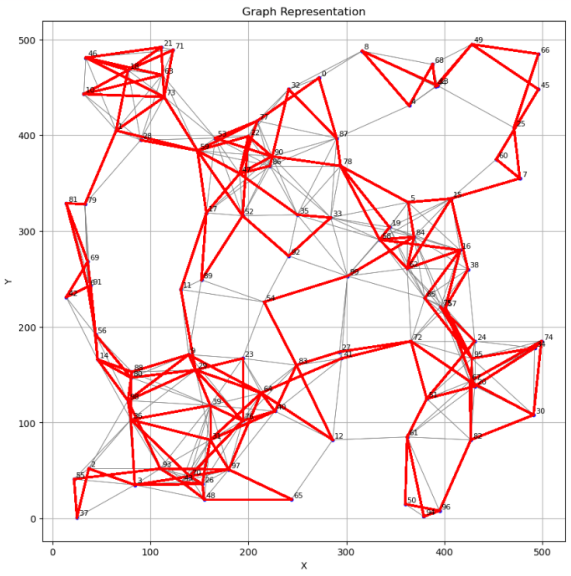

Figure 3: **Exemple of a generated random cycle**

Algorithm: Simple Local Search

```
Require:
 Initial solution s, heuristic function f(s), Neighborhood function N(s)
Ensure: Best solution s*
 1: Initialize s* \gets s
 2: repeat
 3: Select s' \in N(s)
 4: if f(s') > f(s*) then
 5:    s* \gets s'
 6: end if
 7: Update s \gets s'
 8: until Stopping criterion is met
 9: return s*
```

The objective function evaluates the entire graph space. Transitions between solutions are made by replacing a node in the current cycle with another, ensuring valid connections.

$$S\,(1\,4\,[8]\,2\,5\,3\,6\,7\,1) \Longrightarrow (1\,4\,[3]\,2\,5\,3\,6\,7\,1)$$

while 3 and 8 have the same neighbors.

A new cycle is generated, and at this stage, the evaluation function plays a critical role. The evaluation function calculates the total distance of the cycle by summing the distances between each pair of consecutive nodes based on their coordinates. The Euclidean distance formula is used for these calculations. The following is the algorithm for the evaluation function:

Evaluation Function Algorithm

```
Algorithm: Evaluate
Input: Solution s
1: distance +=  distance(x,y) for x,y in s
2: return distance
```

3.2 The train of the hill climbing by selecting a random point

The results demonstrate efficient execution, with the best solution achieving a cost of 11681 km over 1000 iterations.

```
Iteration 973: Best cost = 11699.645036093201
Iteration 974: Best cost = 11699.645036093201
Iteration 975: Best cost = 11699.645036093201
Iteration 976: Best cost = 11699.645036093201
Iteration 977: Best cost = 11699.645036093201
Iteration 978: Best cost = 11699.645036093201
Iteration 979: Best cost = 11680.765560538584
Iteration 980: Best cost = 11680.765560538584
Iteration 981: Best cost = 11680.765560538584
Iteration 982: Best cost = 11680.765560538584
Iteration 983: Best cost = 11680.765560538584
Iteration 984: Best cost = 11680.765560538584
Iteration 985: Best cost = 11680.765560538584
Iteration 986: Best cost = 11680.765560538584
Iteration 987: Best cost = 11680.765560538584
Iteration 988: Best cost = 11680.765560538584
Iteration 989: Best cost = 11680.765560538584
Iteration 990: Best cost = 11680.765560538584
Iteration 991: Best cost = 11680.765560538584
Iteration 992: Best cost = 11680.765560538584
Iteration 993: Best cost = 11680.765560538584
Iteration 994: Best cost = 11680.765560538584
Iteration 995: Best cost = 11680.765560538584
Iteration 996: Best cost = 11680.765560538584
Iteration 997: Best cost = 11680.765560538584
Iteration 998: Best cost = 11680.765560538584
Iteration 999: Best cost = 11680.765560538584
Iteration 1000: Best cost = 11680.765560538584
```

Figure 4: **the generation of hill climbing algorithm**

Below is a plot illustrating the evolution of the solution over 1,000 iterations. This visualization highlights the improvement of the cycle discovered by the algorithm at different stages of the optimization process.

Additionally, the final solution's cycle is visualized to provide a clear representation of the optimal or near-optimal route identified.

The discussion and analysis of the results, including the algorithm's performance, convergence behavior, and the quality of the solutions, are provided in the discussion section. The vizualization of the cycle founded The disscussion of the result in the

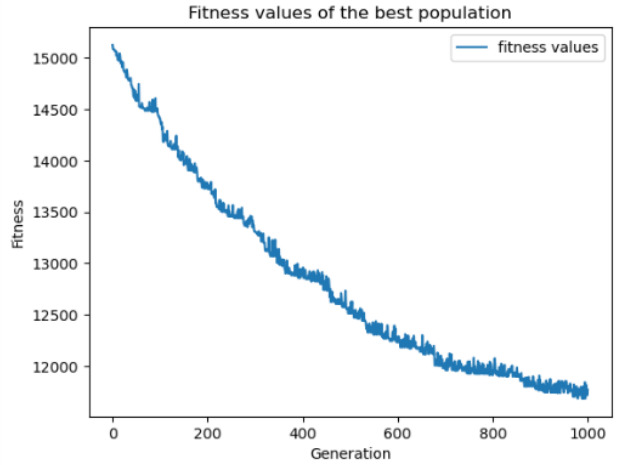

Figure 5: **the evolution of the evaluation function of hill climbing algorithm**

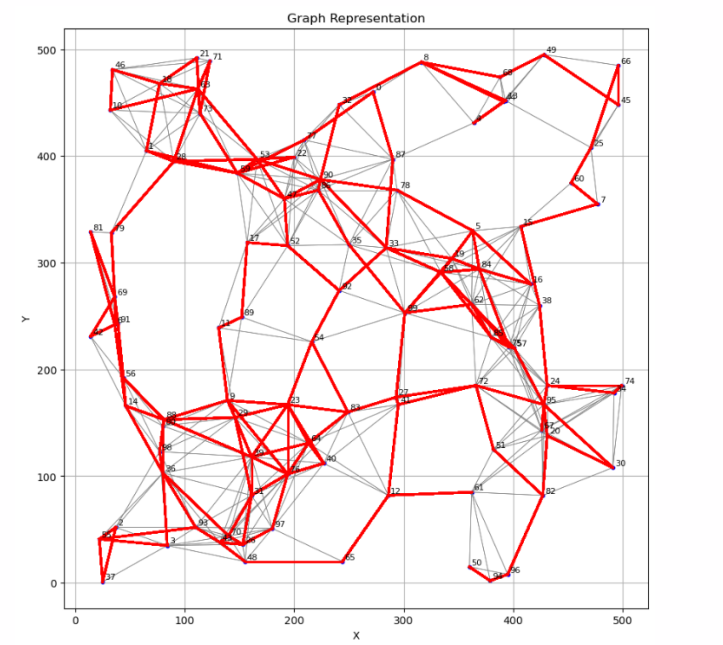

Figure 6: **representation of the final solution**

disscussion section

3.3 Genetic Algorithm Solution

Genetic Algorithms (GAs) are derivative-free, stochastic optimization techniques inspired by biological evolutionary processes, first proposed by Holland. In nature, the principle of "survival of the fittest" dictates that the most suitable individuals are more likely to survive and reproduce, thereby increasing the fitness of subsequent generations.

GAs operate on a population of chromosomes, where each chromosome represents a potential solution as a parameter set. Through processes such as selection, crossover, and mutation, GAs iteratively evolve the population toward better solutions.

A comprehensive overview of Genetic Algorithms and their mechanisms can be found in Goldberg's seminal work on the subject.

Algorithm: Genetic Algorithm

```
Step 1: Create an initial population of P chromosomes.
Step 2: Evaluate the fitness of each chromosome.
Step 3: Choose P/2 parents via proportional selection.
Step 4: Randomly select two parents and create offspring using crossover.
Step 5: Apply mutation operators for minor changes.
Step 6: Repeat Steps 4 and 5 until all parents are mated.
Step 7: Replace the old population with the new one.
Step 8: Evaluate fitness of the new population.
Step 9: Terminate if maximum generations are reached; otherwise, go to Step 3.
```

The major operators in GAs are selection, crossover, and mutation, with crossover playing a critical role.

3.4 Representation

For our constrained TSP, we use path representation, which is the most natural way to represent a cycle. For example, a tour $1 \to 4 \to 8 \to 2 \to 5 \to 3 \to 6 \to 7 \to 1$ is represented as $(1, 4, 8, 2, 5, 3, 6, 7, 1)$. The population consists of n such cycles.

3.5 Fitness Function and Selection

Similar to local search methods such as the hill climbing algorithm, it is necessary to evaluate all individuals in the population to identify the best candidate. The fitness function is defined as the total distance of the cycle, calculated as the sum of the distances between each pair of consecutive cities. The Euclidean distance is used to compute these distances based on the coordinates of the cities. Below is the algorithm for the fitness function.

To ensure the algorithm adheres to the constraint of visiting all nodes, a penalization mechanism is incorporated into the fitness function. This penalization increases the fitness value for invalid solutions (e.g., cycles that do not visit all cities or violate constraints), making them less favorable during the selection process. As a result, the algorithm naturally prioritizes valid solutions while ignoring infeasible ones.

$$\text{Fitness} = \text{Total Distance} + \text{Penalty for Missing Nodes} \tag{2}$$

The penalty increases the fitness value (reduces the quality) for solutions that do not include all nodes.

3.6 Crossover Operators

Dynamic crossover points are chosen randomly, ensuring valid cycles. For example:

```
Parent 1: (1, 2 | 4, 5, 6 | 7, 2, 1)
Parent 2: (3, 6, 4, 5, 6, 7, 2 | 1, 3, 4, 6 | 7, 9, 7)
Offspring 1: (1, 2 | 1, 3, 4, 6 | 7, 2, 1)
Offspring 2: (3, 6, 4, 5, 6 | 4, 5, 6 | 7, 9, 7)
```

3.7 Mutation Function

The same principle applied in the hill climbing operation can be used here: selecting a random point as the starting solution. This approach ensures a diverse starting point, allowing the algorithm to explore a wide range of potential solutions effectively.

$$S(1\,4\,[8]\,2\,5\,3\,6\,7\,1) \rightarrow (1\,4\,[3]\,2\,5\,3\,6\,7\,1)$$

while 3 and 8 have the same neighbors . and here is the algorithm

Algorithm: Genetic Algorithm Require: Population size P, Crossover rate CR, Mutation rate MR, Objective function f(x), Maximum generations Gmax Ensure: Best solution $x*$

```
Step 1: Initialize a population of P candidate solutions randomly.
Step 2: Evaluate the fitness f(x) of each individual in the population.
Step 3: for $g = 1$ to Gmax do
Step 4: Select parents from the population based on fitness (e.g., roulette wheel, to
Step 5: Perform crossover with probability CR to generate offspring.
Step 6: Apply mutation with probability MR to introduce diversity.
Step 7: Evaluate the fitness f(x) of the offspring.
Step 8: Select individuals for the next generation based on fitness (e.g., elitism or
Step 9: end for
Step10: return Best solution $x*$ from the final population.
```

3.8 The train of the genetic algorithm

By implementing the algorithm on python we got this train result by setting max generation at 300, mutation $rate = 0.2$ and population size $= 50$ The train as follow

The algorithm well converge and stack at the solution of cost 5954 km . The evolution of the fitness function per generation : As we noticed the genetic algorithm success to generate better population in term of fitness along of generation and here is the plot of the cycle

```
Generation 271: Best fitness = 5954.849348195695, len unique = 100
Generation 272: Best fitness = 5954.849348195695, len unique = 100
Generation 273: Best fitness = 5954.849348195695, len unique = 100
Generation 274: Best fitness = 5954.849348195695, len unique = 100
Generation 275: Best fitness = 5954.849348195695, len unique = 100
Generation 276: Best fitness = 5954.849348195695, len unique = 100
Generation 277: Best fitness = 5954.849348195695, len unique = 100
Generation 278: Best fitness = 5954.849348195695, len unique = 100
Generation 279: Best fitness = 5954.849348195695, len unique = 100
Generation 280: Best fitness = 5954.849348195695, len unique = 100
Generation 281: Best fitness = 5954.849348195695, len unique = 100
Generation 282: Best fitness = 5954.849348195695, len unique = 100
Generation 283: Best fitness = 5954.849348195695, len unique = 100
Generation 284: Best fitness = 5954.849348195695, len unique = 100
Generation 285: Best fitness = 5954.849348195695, len unique = 100
Generation 286: Best fitness = 5954.849348195695, len unique = 100
Generation 287: Best fitness = 5954.849348195695, len unique = 100
Generation 288: Best fitness = 5954.849348195695, len unique = 100
Generation 289: Best fitness = 5954.849348195695, len unique = 100
Generation 290: Best fitness = 5954.849348195695, len unique = 100
Generation 291: Best fitness = 5954.849348195695, len unique = 100
Generation 292: Best fitness = 5954.849348195695, len unique = 100
Generation 293: Best fitness = 5954.849348195695, len unique = 100
Generation 294: Best fitness = 5954.849348195695, len unique = 100
Generation 295: Best fitness = 5954.849348195695, len unique = 100
Generation 296: Best fitness = 5954.849348195695, len unique = 100
Generation 297: Best fitness = 5954.849348195695, len unique = 100
Generation 298: Best fitness = 5954.849348195695, len unique = 100
Generation 299: Best fitness = 5954.849348195695, len unique = 100
Generation 300: Best fitness = 5954.849348195695, len unique = 100
```

Figure 7: **the generation of GA**

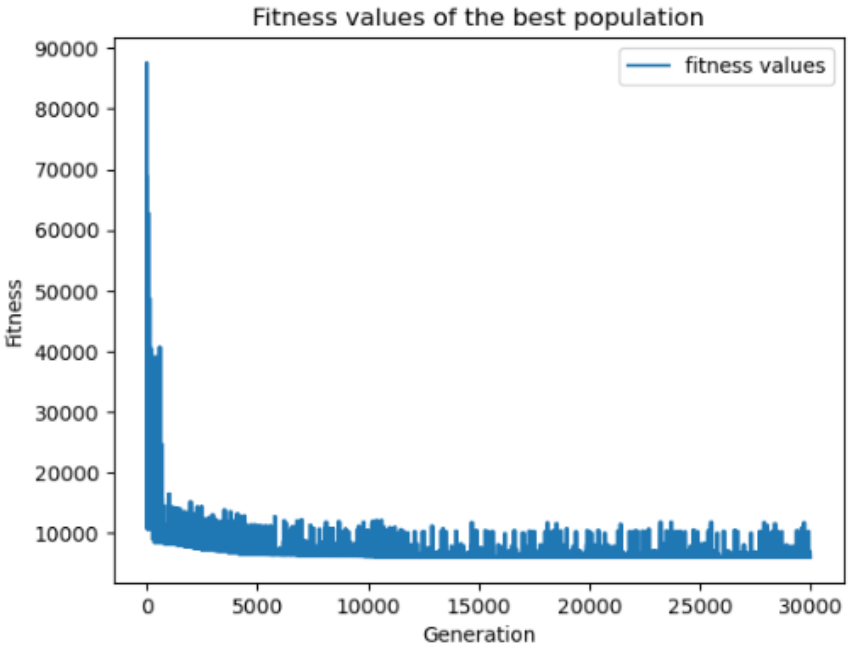

Figure 8: **evolution of fitness function of GA per state**

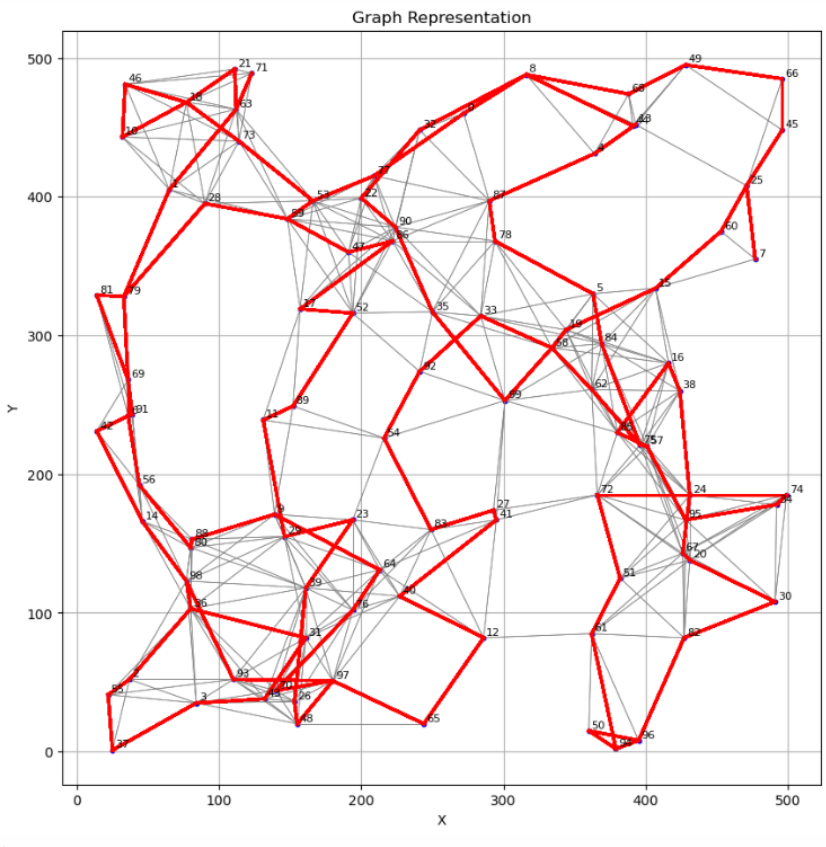

Figure 9: **representation of the final solution by GA**

4 Comparison between the result obtain from hill climbing algorithm and genetic algorithme

Algorithm	Hill Climbing	Genetic Algorithm
Execution Time (ns)	688,23	38141.99
Current Solution (km)	11681	5954
Best Solution (km)	7289	5208

From analyzing the table, we can conclude the following:

Local Search Methods: Local search methods, such as hill climbing, are significantly faster in terms of both spatial and temporal complexity when compared to global methods. However, they are highly sensitive to the initial solution. If the starting point is poorly chosen, the algorithm may become trapped in a local optimum and fail to converge to a better solution. This limitation highlights their reliance on the quality of the initial solution.

Global Search Methods: Global methods, such as Genetic Algorithms (GAs), are less sensitive to the initial solution due to their ability to explore a broader solution space. This flexibility allows them to escape local optima and often results in finding better solutions compared to local search methods. However, this comes at the cost of increased computational complexity and resource consumption.

Comparison and Dependency: Both local and global methods are influenced by the initial solution. The primary difference lies in the range and diversity of solutions they can explore. Local methods focus on a smaller neighborhood of solutions, while global

methods, through mechanisms like crossover and mutation, consider a wider and more diverse set of possibilities, ultimately improving the likelihood of finding an optimal or near-optimal solution.

Hybrid approaches can be implemented between the Hill Climbing algorithm and Genetic Algorithm (GA) in several ways:

Generating the Initial Population: The initial population of the GA can be generated using the Hill Climbing algorithm. This ensures that the initial solutions are already locally optimized, providing a better starting point for the GA.

Optimizing GA Solutions: The Hill Climbing algorithm can be applied to refine the solutions generated by the GA during its evolution, such as after mutation or crossover.

Optimizing Selected Parents: Each selected parent in the GA can be optimized using the Hill Climbing algorithm before being used in crossover. This improves the quality of the offspring produced by the parents.

This hybrid approach can effectively generate better solutions by combining the local optimization capabilities of Hill Climbing with the global exploration of Genetic Algorithms. However, the time complexity of this approach can grow exponentially, especially if the Hill Climbing algorithm is applied to optimize every selected parent or multiple solutions during each generation. In summary, while local search methods are faster and resource-efficient, global methods provide a more robust exploration of the solution space, albeit at a higher computational cost. The choice between the two depends on the specific requirements of the problem, such as solution quality, time constraints, and computational resources.

5 Conclusion

Local search methods are computationally efficient but are constrained by their sensitivity to initial conditions, often leading to suboptimal solutions if the starting point is poorly chosen. In contrast, global methods, such as Genetic Algorithms (GAs), offer greater diversity and robustness, making them well-suited for addressing complex, constrained problems where exploration of a broader solution space is necessary.

Furthermore, metaheuristic methods, including GAs, can be effectively adapted to training large deep learning models. Their simplicity and flexibility make them a promising approach for optimizing complex, high-dimensional problems in this domain, providing efficiency and robustness in tackling challenges such as hyperparameter tuning and escaping local optima in non-convex optimization landscapes.