Lab report:
Biologically-inspired algorithms and models

Part I: Local optimization algorithms, problem: QAP

April 2, 2024

Lecturer: dr hab. inż. Maciej Komosiński, prof. PP

Authors: **Jędrzej Warczyński** inf148234 AI jedrzej.warczynski@student.put.poznan.pl

Thursday classes, 18:20.

1 Problem description

The Quadratic Assignment Problem (QAP) is a combinatorial optimization problem focused on assigning a set of facilities to a set of locations in a way that minimizes the total cost. Each facility has an associated flow and each pair of facilities has an associated distance or cost. The objective is to minimize the sum of the products of the flows and distances, considering the assignment of facilities to locations.

Applications of QAP span various fields including logistics, telecommunications, computer science, and facility layout planning. In logistics, it's used for warehouse organization and facility allocation. In telecommunications, it aids in network optimization. In computer science, it's applied to circuit design and compiler optimizations. QAP also finds relevance in operations research for optimal facility placement.

Interpretations of QAP can range from arranging physical facilities to minimize transportation costs to organizing data to reduce access time. Its complexity is significant, classified as NP-hard, making it challenging to solve for large instances.

1.1 Problem formulation

The Quadratic Assignment Problem can be formulated as follows: Given n facilities and n locations, where n is the number of facilities and locations, the goal is to find a permutation π of the facilities that minimizes the total cost. The total cost is calculated as the sum of the products of the flows and distances for all pairs of facilities. The cost function is defined as:

$$f(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} \cdot b_{\pi(i)\pi(j)}$$

where a_{ij} is the flow between facilities i and j, $b_{\pi(i)\pi(j)}$ is the distance between the locations assigned to facilities i and j, and π is a permutation of the facilities.

The goal is to find the permutation π that minimizes the total cost $f(\pi)$.

1.2 Instance description

The instances used in this study are taken from the QAPLIB library, a collection of QAP instances used for benchmarking algorithms. The instances used in this study are:

- *chr12a*: A 12-facility instance with known optimal solution.
- *chr15a*: A 15-facility instance with known optimal solution.
- *chr18a*: An 18-facility instance with known optimal solution.
- *chr20a*: A 20-facility instance with known optimal solution.
- *chr22a*: A 22-facility instance with known optimal solution.
- *chr25a*: A 25-facility instance with known optimal solution.
- *lipa20a*: A 20-facility instance with known optimal solution.
- *tai12b*: A 12-facility instance with known optimal solution.
- *tai100b*: A 100-facility instance with known optimal solution.

The selection of instances from the *chr* group was based on their progressive difficulty levels, allowing for a comprehensive evaluation spanning from relatively easier to more challenging problem instances. Additionally, the inclusion of the *lipa20a* instance, despite its similar size to the *chr* instances, was motivated by its distinct characteristics, providing a different set of challenges for algorithm evaluation. While generally less demanding than the *chr* instances, it offers valuable insights into algorithm performance across diverse problem landscapes. Furthermore, the inclusion of the *tai* instances was aimed at testing the algorithms across a wider spectrum of problem sizes.

2 Implementation

The Quadratic Assignment Problem was solved using five algorithms:

- **Heuristic (H):** The Heuristic algorithm operates by analyzing the sums of rows in matrices a and b, representing flows and distances. It iteratively identifies the facility with the highest sum in matrix a and assigns it to the location with the lowest sum in matrix b. This process continues until a solution is reached.
- **Steepest (S):** A local search algorithm that iteratively explores the neighborhood of the current solution and selects the best neighbor. It also allows for some number of subsequent non-improving moves before terminating.
- **Greedy (G):** A local search algorithm that iteratively explores the neighborhood of the current solution and selects the first improving neighbor. It terminates when no improvement is possible.
- **Random Walk (RW):** An algorithm that in each step selects a random neighbor and track the best solution encountered.
- **Random Search (RS):** A random search algorithm that in each step generates random permutations of the facilities and selects the best solution encountered.

Implemenatation of these algorithms was done in Rust programming language.

3 Description of the neighborhood operators

For the neighborhood operator "Swap" operator was used. In this operator, two elements within a permutation representing the assignment of facilities to locations are selected, and their positions are swapped. This operation effectively generates a neighboring solution by altering the assignment of two facilities to different locations, potentially leading to a decrease in the total cost. The neighborhood size of the "Swap" operator corresponds to the number of possible swaps that can be performed within a given permutation. For a permutation of length n, there are $\frac{n\times(n-1)}{2}$ possible swaps, as each pair of elements can be swapped, and redundant swaps (swapping an element with itself or swapping two elements back to their original positions) are excluded.

4 Algorithms Performance Comparison

4.1 Metrics

4.1.1 Metrics definition

In this study, the performance of five algorithms - H, S, G, RW, RS - was evaluated on the Quadratic Assignment Problem (QAP). The choice of metrics used for comparison was carefully considered to provide a comprehensive evaluation of each algorithm's performance.

The metrics selected for evaluation are as follows:

- **Running Time (T):** The time of execution of algorithm on a given instance.
- **Quality (Q):** The quality of the solution achieved by each algorithm, calculated using the formula:

$$Q = 1 - \frac{(cost - cost^*)}{cost}$$

where *cost* represents the total cost of the solution and *cost** represents the optimal cost.

- **Instance Efficiency ($\boldsymbol{Eff_i}$):** The efficiency of each algorithm on a given instance, calculated as:

$$Eff_i = \frac{Quality}{1 + \frac{Time - \min(Time)}{\max(Time) - \min(Time)}}$$

Here, $\min(Time)$ and $\max(Time)$ represent the minimal and maximal execution times across all algorithms being compared.

4.1.2 Metrics justification

The quality of the solution is crucial in determining the effectiveness of each algorithm. To quantify how close each algorithm's solution is to the optimal solution, the quality metric, defined as $Q = 1 - \frac{(cost - cost^*)}{cost}$, has been chosen. This metric allows for a direct comparison of the solution quality across different algorithms. It is normalized to the range $[0, 1]$, where $Q = 1$ indicates an optimal solution and as Q approaches 0, the solution quality decreases. This property makes it suitable for comparing the performance of different algorithms on the same instances.

The instance efficiency metric offers a comprehensive measure of algorithm performance by incorporating both solution quality and running time. This metric accounts for variations in solution quality and running time across different instances, allowing for fair comparisons among algorithms. By normalizing the quality of solutions with respect to the range of execution times, it provides a balanced assessment that considers the trade-offs between solution optimality and computational efficiency. Furthermore, the instance efficiency metric facilitates comparisons across multiple algorithms by standardizing the evaluation process. It enables to gauge the relative effectiveness of different algorithms in solving the same set of instances, irrespective of their inherent complexities or computational requirements. However, it's important to note that the instance efficiency metric may not accurately capture the difficulty of individual instances, as the range of running times can vary significantly. Therefore, while it serves as a useful tool for comparing algorithm performance across different instances, it may not fully reflect the intricacies of specific problem instances.

Overall, the instance efficiency metric provides a valuable framework for assessing algorithmic performance in a comprehensive and standardized manner, aiding in making informed decisions about algorithm selection and optimization strategies.

4.2 Results

The experimental results are depicted in 1. It is evident that subsequent instances become increasingly challenging, resulting in a decrease in solution quality for most algorithms. An exception is observed with the Heuristic algorithm, which manages to maintain consistent solution quality across all instances. However, it's worth noting that solutions produced by the Heuristic algorithm are inferior to those of other algorithms for all instances except one (chr18a), where the Random Walk algorithm yields a slightly worse solution. The advantage of the Heuristic algorithm lies in its superior speed compared to other algorithms. This advantage is reflected in the efficiency metric, where the Heuristic algorithm outperforms random algorithms on instances chr12a and chr15a, and emerges as the best-performing algorithm on the remaining two instances. This can be attributed to the deterministic nature of the Heuristic algorithm, which avoids the complexities associated with extensive space exploration.

The Steepest and Greedy Algorithms yield the best results, with the Steepest algorithm slightly outperforming the Greedy algorithm in terms of both solution quality and efficiency for the first two instances. Conversely, the Greedy algorithm demonstrates a slight advantage over the Steepest algorithm for the last two instances. One notable distinction between these algorithms is their approach to non-improving moves: the Steepest algorithm allows for a certain number of such moves, whereas the Greedy algorithm terminates when no further improvement is possible. This suggests that, for less challenging instances, permitting non-improving moves could be beneficial. Although there is a difference between these two algorithms, it is not significant, and the choice between them should be based on the specific requirements of the problem. Moreover, the running time is practically identical for both algorithms.

On the other hand, the Random Walk and Random Search algorithms yield poor results. While the quality of solutions decreases with the increasing difficulty of instances, the decline is not as pronounced as with local search algorithms (Greedy and Steepest). The running time of these algorithms was manually constrained to match the execution time of the Greedy and Steepest algorithms.

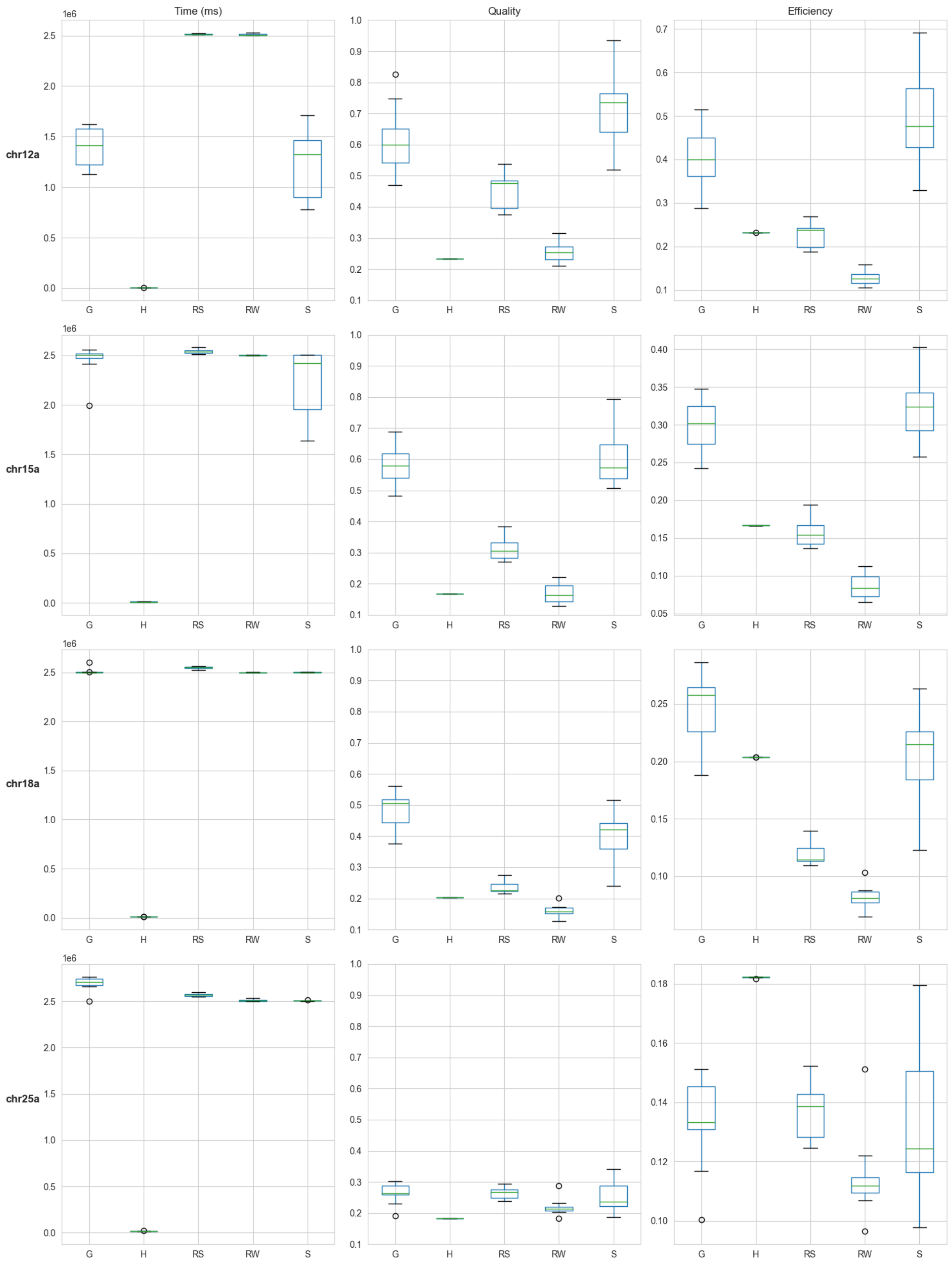

Figure 1: Comparison of algorithms performance

In Figure 2, the number of algorithmic steps (changes to the current solution) is depicted on the right side of the plot, while the number of evaluated (fully or partially) solutions is shown on the left side. The number of algorithmic steps is higher for the Greedy Algorithm compared to the Steepest Algorithm. This disparity arises because the Greedy Algorithm continuously modifies the solution to maximize improvement, while the Steepest Algorithm evaluates all possible neighbors before selecting the best one, resulting in less frequent solution changes. Conversely, the number of evaluated solutions is lowest for the Random Search algorithm. This is because Random Search requires full evaluation of each solution to assess its quality, which is more time-consuming compared to partial evaluation. Additionally, there is a cost associated with updating the best encountered solution, which requires copying the entire solution—a process that is not necessary for algorithms like Greedy and Steepest, where only two elements are swapped.

The Greedy algorithm conducts fewer evaluations than the Steepest and Random Walk algorithms because it does not allow for non-improving moves and can terminate earlier. On the other hand, Random Walk and Steepest algorithms perform a similar number of evaluations since they allow for non-improving moves and conduct partial evaluations at each step.

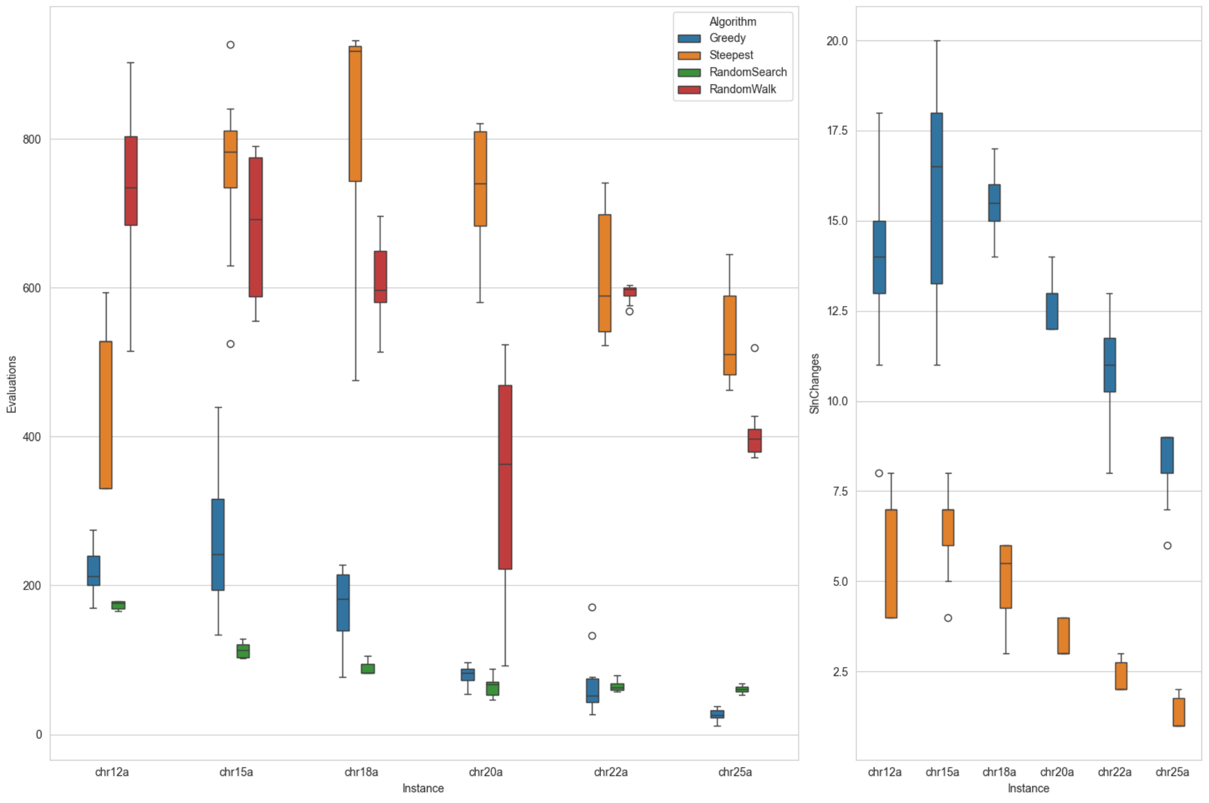

Figure 2: Number of algorithmic steps and evaluated solutions

4.3 Initial solution influence

Figure 3 illustrates the relationship between the quality of the solution and the quality of the initial solution. The test was conducted for the Greedy and Steepest algorithms across three instances: chr12a, chr20a, and lipa20a. Interestingly, the quality of the solution does not appear to depend on the quality of the initial solution. This observation is consistent across both algorithms and all instances, as evidenced by the correlation coefficient, which is close to zero.

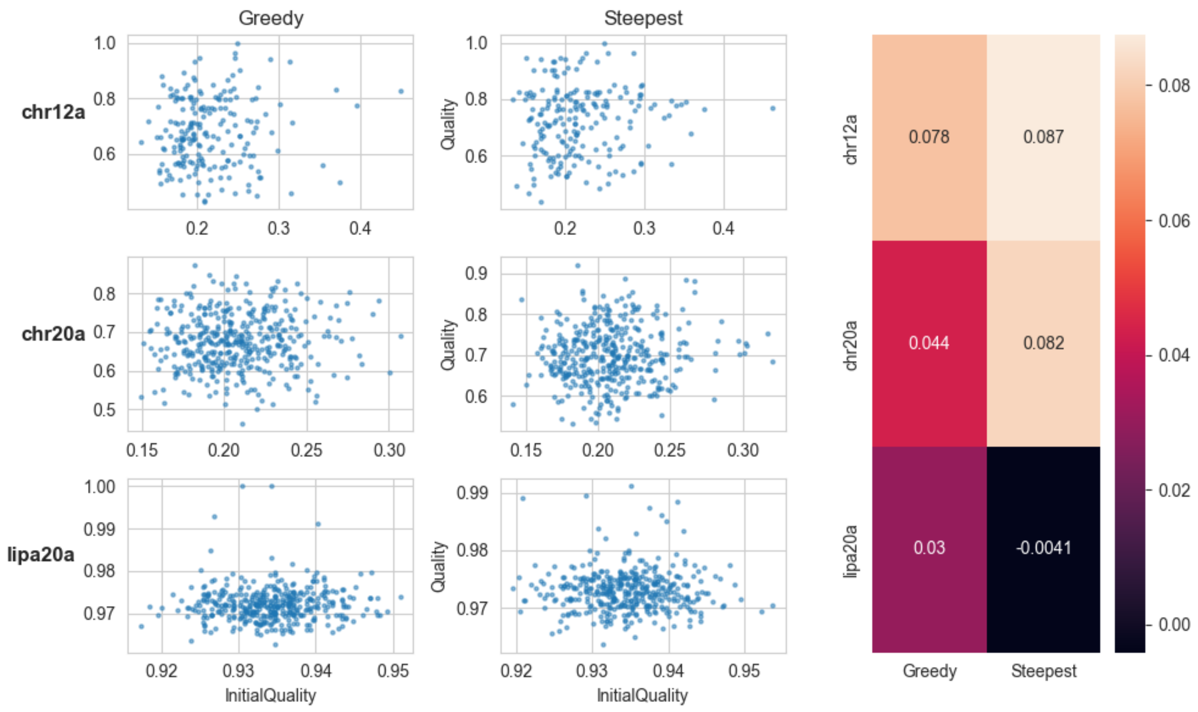

Figure 3: Quality of the solution vs quality of the initial solution

4.4 Algorithm performance over subsequent runs

Figure 4 presents the average and lowest cost of the solution for the Greedy and Steepest algorithms obtained up to the i-th run. It is noticeable that the average cost of the solution remains relatively stable after the first 50 runs. The most significant decrease in cost occurs within the initial 10 runs. Subsequent reductions in cost occur intermittently, with noticeable decreases observed after approximately 100 to 300 runs, depending on the instance. Therefore, it is advisable to run the algorithm for at least 10 or 20 iterations to observe substantial improvements in solution quality.

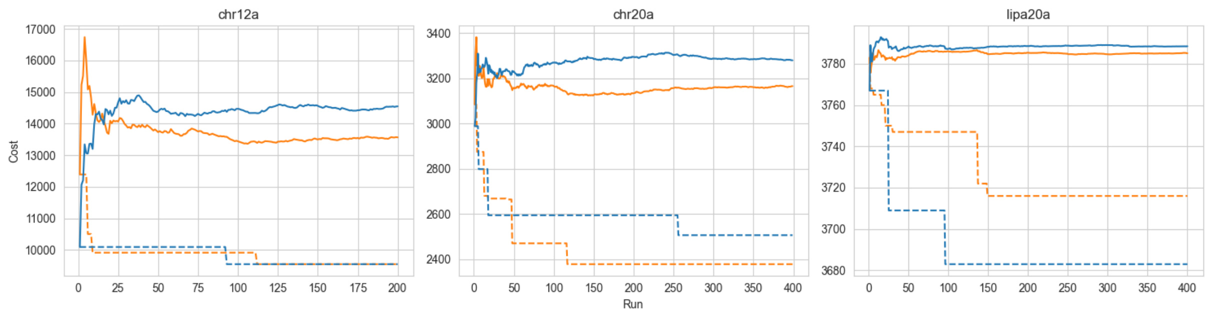

Figure 4: Average and lowest cost of the solution

4.5 Distance from optimal solution in decision space

The correlation between the quality of the solution and the distance in decision space from the optimal solution, calculated as the number of elements that differ between the permutation representing the solution and the optimal permutation, is depicted in Table 1 for two instances, tai12b and tai100b. Corresponding scatter plots are provided in Figure 5. For both instances, tai12b and tai100b, the correlation is negative, indicating that the quality of the solution tends to decrease as the distance from the optimal solution increases. However, the correlation is not significant, suggesting that local minima are distributed across the decision space.

Instance	Algorithm	Correlation
tai12b	Greedy	-0.46
tai12b	Steepest	-0.51
tai100b	Greedy	-0.63
tai100b	Steepest	-0.53

Table 1: Correlation between the quality of the solution and the distance in decision space from optimal solution

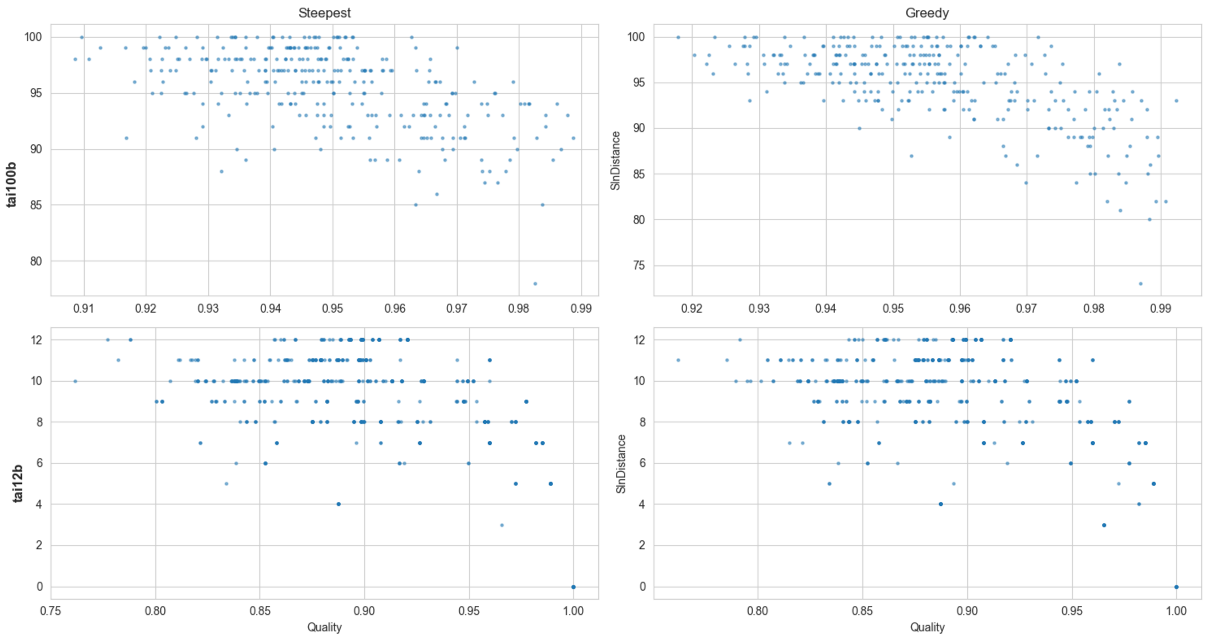

Figure 5: Quality of the solution vs distance from optimal solution

5 Conclusions

The Quadratic Assignment Problem (QAP) poses a significant challenge in combinatorial optimization, finding applications across diverse fields. This study evaluated the performance of five algorithms - Heuristic, Steepest, Greedy, Random Walk, and Random Search - on a range of QAP instances. The results demonstrate that the Greedy and Steepest algorithms consistently outperform the Random Walk and Random Search algorithms in terms of solution quality. Although the Heuristic algorithm yields suboptimal solutions, its advantage lies in its speed, making it suitable for less challenging instances prioritizing efficiency. Both the Greedy and Steepest algorithms emerge as top performers in solution quality. However, the choice between them should consider specific problem requirements, as the difference in performance is not substantial. Conversely, the Random Walk and Random Search algorithms deliver poor results, with solution quality deteriorating as instance difficulty increases. Variations in the number of algorithmic steps and evaluated solutions are observed among algorithms, with the Greedy algorithm conducting fewer evaluations compared to Steepest and Random Walk algorithms. Furthermore, for the Greedy and Steepest algorithms, the quality of the solution is found to be independent of the initial solution quality. The stability of average and lowest solution costs after a certain number of runs suggests the benefit of running the algorithm for at least 10 to 20 iterations. Regarding the relationship between solution quality and distance from the optimal solution, a negative correlation is observed for both instances tai12b and tai100b. This indicates that solution quality tends to decrease as the distance from the optimal solution increases. However, the lack of significance in the correlation implies that local minima are scattered throughout the decision space.

In summary, while the Greedy and Steepest algorithms exhibit superior performance, careful consideration of algorithmic characteristics and problem requirements is essential for algorithm selection. Further research could explore algorithmic enhancements and strategies for addressing challenges posed by local minima.

6 Future Work

In future studies, enhancing the heuristic algorithm for the Quadratic Assignment Problem (QAP) by considering pairs of rows in each assignment step could be promising. Instead of assigning the highest sum row in matrix A to the lowest sum row in matrix B, we can explore assigning the two highest sum rows in A to the two lowest sum rows in B. This approach aims to minimize the total cost by carefully selecting pairs of rows that offer the most favorable assignment options. Implementing this refined assignment strategy within the heuristic algorithm would involve adapting the assignment mechanism to evaluate pairs of rows from matrices A and B systematically. By considering multiple potential assignments simultaneously, the algorithm could explore a broader range of possible solutions and potentially identify more favorable combinations leading to improved solution quality.

However, it's important to note that this enhanced approach may require significantly more computational power, resulting in slightly longer execution times. The increased complexity of evaluating pairs of rows in each assignment step could lead to a higher computational burden, particularly for larger QAP instances.

Evaluating the performance of this enhanced heuristic algorithm across various QAP instances would be essential to assess its effectiveness and efficiency.

In summary, exploring the idea of considering pairs of rows in each assignment step represents a promising avenue for advancing this heuristic algorithm and enhancing its performance in solving the Quadratic Assignment Problem.