*electronics*

*Article*

# Combining Genetic Algorithm with Local Search Method in Solving Optimization Problems

**Velin Kralev and Radoslava Kraleva ***

Department of Informatics, Faculty of Mathematics and Natural Sciences, South-West University, 2700 Blagoevgrad, Bulgaria; velin_kralev@swu.bg
* Correspondence: rady_kraleva@swu.bg

**Abstract:** This research is focused on evolutionary algorithms, with genetic and memetic algorithms discussed in more detail. A graph theory problem related to finding a minimal Hamiltonian cycle in a complete undirected graph (Travelling Salesman Problem—TSP) is considered. The implementations of two approximate algorithms for solving this problem, genetic and memetic, are presented. The main objective of this study is to determine the influence of the local search method versus the influence of the genetic crossover operator on the quality of the solutions generated by the memetic algorithm for the same input data. The results show that when the number of possible Hamiltonian cycles in a graph is increased, the memetic algorithm finds better solutions. The execution time of both algorithms is comparable. Also, the number of solutions that mutated during the execution of the genetic algorithm exceeds 50% of the total number of all solutions generated by the crossover operator. In the memetic algorithm, the number of solutions that mutate does not exceed 10% of the total number of all solutions generated by the crossover operator, summed with those of the local search method.

**Keywords:** genetic algorithm (GA); memetic algorithm (MA); local search; optimization

**Citation:** Kralev, V.; Kraleva, R. Combining Genetic Algorithm with Local Search Method in Solving Optimization Problems. *Electronics* **2024**, *13*, 4126. https://doi.org/10.3390/electronics13204126

Academic Editors: Marcin Witczak, Eleuda Nuñez, Marie-Monique Schaper and Bipin Indurkhya

Received: 4 August 2024
Revised: 28 September 2024
Accepted: 17 October 2024
Published: 20 October 2024

## 1. Introduction

A specific but very efficient subclass of evolutionary algorithms is the class of genetic algorithms [1]. The interest in these algorithms has grown significantly over the past few years. They are used to solve specific tasks [2–4], as well as to analyze and study their behavior. Some of their modifications have been used to solve specific optimization problems [5–7]. These algorithms can generate solutions that are optimal (or close to optimal) in an acceptable time [8]. A distinctive characteristic of most heuristic approaches is that they work well for certain problems but it is difficult to adapt the algorithm to a new one [9]. With a large amount of input data, using the exact approaches is inapplicable, due to the large number of possible solutions that should be generated and evaluated. For the class of NP-hard problems, such as schedule theory and graph theory problems, good approximate solutions have been found using genetic algorithms [10–12]. These problems are known to be similar but not identical to others, for which good and efficient algorithms are currently known. A small change in the problem formulation can lead to a large change in the performance (in terms of speed) of the best-known algorithm so far [13].

Unlike exact algorithms, genetic algorithms (GAs) find solutions that are some approximations. This means that these solutions are close to equal to the optimal. These algorithms process a set of acceptable solutions that are modified according to certain rules, to generate even better solutions. Each solution is represented numerically and evaluated by a numerical meter according to a defined optimality criterion. After generating the solutions in the initial solution set, each solution can be kept unchanged in the next solution set, combined with another to produce a new one, or they are discarded and not analyzed further. Some of the solutions can be further modified with a degree of pre-set probability, and these modifications are usually minor [14]. Each iteration of the genetic algorithm generates a new set of solutions, thus performing the main iterative process of the algorithm. A special solution evaluation function (fitness function) determines whether a solution is better or worse. This function "evaluates" each solution by assigning a quantitative measure of its quality. The solutions are classified (as better or worse) based on the scores obtained [15–18].

One possible way to improve genetic algorithms, when the solutions in the solution set are close to the optimal or a large part of them are identical, is to apply local search-based approaches. These are approaches where a better solution is sought by changing an existing one. The difference between such approaches and the application of the genetic operator for mutation is that in mutation, the search principle is guessing, i.e., changing one or more elements of the solution with the expectation that this will lead to a better solution. Unlike with local search, with the genetic mutation operator, a change is always made, regardless of whether it will lead to the construction of a better or a worse solution.

The basic concept of memetic algorithms (MAs) was originally presented in [19]. In that work, an evolutionary algorithm that uses a method based on local search is considered. A similar idea, however, was later formulated in [20]. In that work, a comparative analysis is made between these algorithms, presenting the advantages and disadvantages of both types. In memetic algorithms, a smaller structure of the entire solution is analyzed and used, and it is viewed as a unit of information that can reproduce itself. This information is perceived and used differently from the corresponding elements of a decision. The main difference is that as this information passes between solutions, each solution adapts it in the best possible way for the moment. Unlike memetic algorithms, the genetic ones leave the gene unchanged [21–23].

The most important advantage of using memetic algorithms is that the space of acceptable solutions is reduced to a smaller set (subspace of solutions), among which there is a local optimum (minimum or maximum). Executing some genetic operators can generate a solution that has either a worse or a better score as computed by a fitness function. When using the local search method, the best possible location of the solution component will always be identified for the currently considered solution. This guarantees that a solution will be found that yields the best possible score generated by the local search method [24,25].

Genetic algorithms are often used to solve many optimization problems from various fields of science, such as graph theory. It is an intensively researched area of computer science (or more precisely, an area of discrete mathematics) that has undergone great development over the past few decades. It has a huge practical application because in many cases, the analysis and description of many systems can be successfully described with graph-type structures [26,27]. By adding a local search method to the standard crossover and mutation operators, the computational complexity of the algorithm is increased, and the execution time can increase significantly [10]. Given the advantages that memetic algorithms provide, they will be the subject of further research to determine the influence of the local search method on the quality of the generated solutions. In general, the computational complexity for both algorithms is quadratic and depends on the input data size.

A large class of graph theory problems, such as NP-hard problems, can be solved by some approximate algorithms, such as genetic algorithms [2,28–30]. Finding an exact solution to these problems (but with a large amount of input data) can be unacceptable in terms of the computational time required to generate the corresponding solution [31]. The backtracking approach always guarantees that optimal solutions will be found but is only applicable to problems with a small amount of input data. This approach, although possible, is practically impractical. For example, for a complete undirected graph with 35 vertices and $35 \times (35 - 1)/2 = 595$ edges, the number of all Hamiltonian paths is huge: $(35 - 1)!/2 \approx 1.5 \times 10^{38}$. Therefore, modifying and improving existing heuristic algorithms (such as genetics) is expected [32–34]. Through them, it is possible to solve some "difficult" problems "well", which would take an unacceptably long time.

Many real-world problems can be modeled using graphs. For the current study, the problem of finding a minimal Hamiltonian cycle in a complete, weighted, undirected graph will be used. A characteristic of this problem is that it is a combinatorial optimization problem and has been very intensively studied in recent years. It is known in the literature as the Travelling Salesman Problem (TSP), and its variants and detailed description are presented in [35]. There are several approaches and many algorithms for solving the TSP [36–38]. Two of them are basic—the exact and approximate methods. Exact algorithms guarantee optimal solutions but often suffer from high computational time. They are applicable when the number of vertices in the graph is small [39,40]. On the other hand, approximate algorithms find a solution that is close to optimal, and the time to do so is acceptable. Such algorithms are discussed in [17,41–43]. Most of the developed algorithms for TSP fall precisely into the second category. According to the optimization methods, the TSP algorithms can be classified into more categories [38].

## 2. Materials and Methods

A method to transform a genetic algorithm into a memetic algorithm by adding a local search-based method will be presented in this section. An example of its application will also be presented, dividing the set of elements of a given solution into several commensurable subsets of elements (groups), for example, 3, 4, or 6. This approach applies the rearranging of the elements in each group so that this element occupies the first position in a group for a specific modified solution. When this local search method is used, the last found (and accordingly best) solution cannot be lost, i.e., either some solution improvement will be achieved, or the previous best solution found will remain unchanged.

An example of finding a minimal Hamiltonian cycle will be presented using this algorithm. The example graph, shown in Figure 1, is generated randomly. The minimum Hamiltonian cycle is found by the backtracking method, which is similar to the method presented in [31].

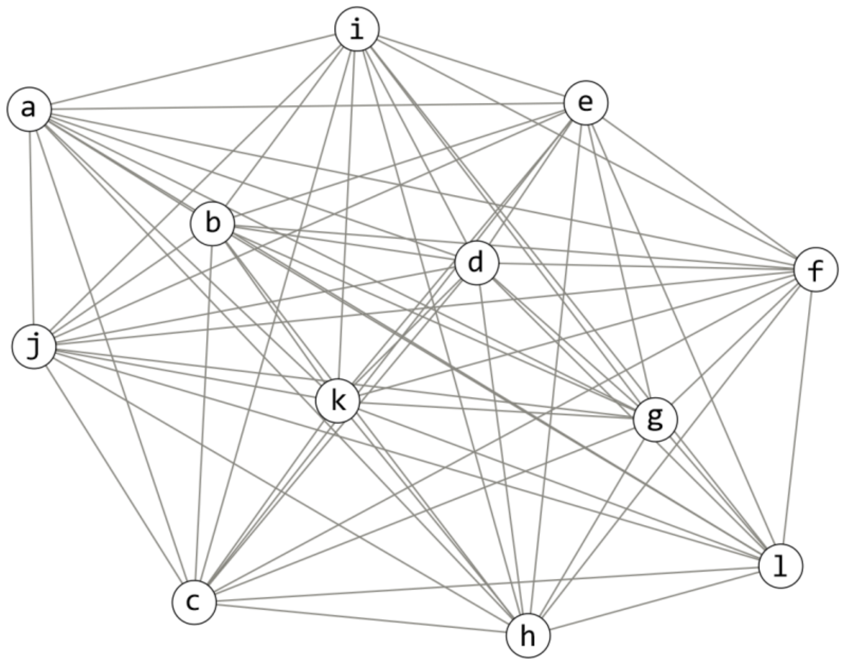

**Figure 1.** Complete undirected graph G (∣V∣ = 12, ∣E∣ = 66).

Table 1 presents a weight matrix of the graph G (12, 66). The values are the distances between vertices (in pixels).

**Table 1.** Weight matrix of graph G (|V| = 12, |E| = 66).

| V\V | a | b | c | d | e | f | g | h | i | j | k | l |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | 0 | 112 | 269 | 269 | 300 | 403 | 381 | 391 | 200 | 100 | 212 | 447 |
| b | 112 | 0 | 200 | 158 | 206 | 300 | 269 | 283 | 112 | 112 | 112 | 335 |
| c | 269 | 200 | 0 | 212 | 320 | 361 | 269 | 200 | 269 | 180 | 112 | 304 |
| d | 269 | 158 | 212 | 0 | 112 | 158 | 112 | 158 | 112 | 250 | 112 | 180 |
| e | 300 | 206 | 320 | 112 | 0 | 112 | 158 | 250 | 100 | 316 | 212 | 224 |
| f | 403 | 300 | 361 | 158 | 112 | 0 | 112 | 224 | 206 | 403 | 269 | 150 |
| g | 381 | 269 | 269 | 112 | 158 | 112 | 0 | 112 | 212 | 354 | 200 | 71 |
| h | 391 | 283 | 200 | 158 | 250 | 224 | 112 | 0 | 269 | 335 | 180 | 112 |
| i | 200 | 112 | 269 | 112 | 100 | 206 | 212 | 269 | 0 | 224 | 158 | 283 |
| j | 100 | 112 | 180 | 250 | 316 | 403 | 354 | 335 | 224 | 0 | 158 | 412 |
| k | 212 | 112 | 112 | 112 | 212 | 269 | 200 | 180 | 158 | 158 | 0 | 255 |
| l | 447 | 335 | 304 | 180 | 224 | 150 | 71 | 112 | 283 | 412 | 255 | 0 |

The algorithm, used to find an exact solution, is based on the backtracking approach and is optimized by the branch-and-bound method. The minimum Hamiltonian cycle has a length of 1393 (in pixels) with a total of 18,767,942 solutions generated and 29 improvements found by the algorithm in the search process after 875 ms (milliseconds). The minimum Hamiltonian cycle and the corresponding sequence of vertices in the order of their traversal are shown in Figure 2. The minimal Hamiltonian cycle contains the following vertex traversal sequence:

(a)–(b)–(i)–(e)–(f)–(g)–(l)–(h)–(d)–(k)–(c)–(j)–(a)

112 + 112 + 100 + 112 + 112 + 71 + 112 + 158 + 112 + 112 + 180 + 100 = 1393

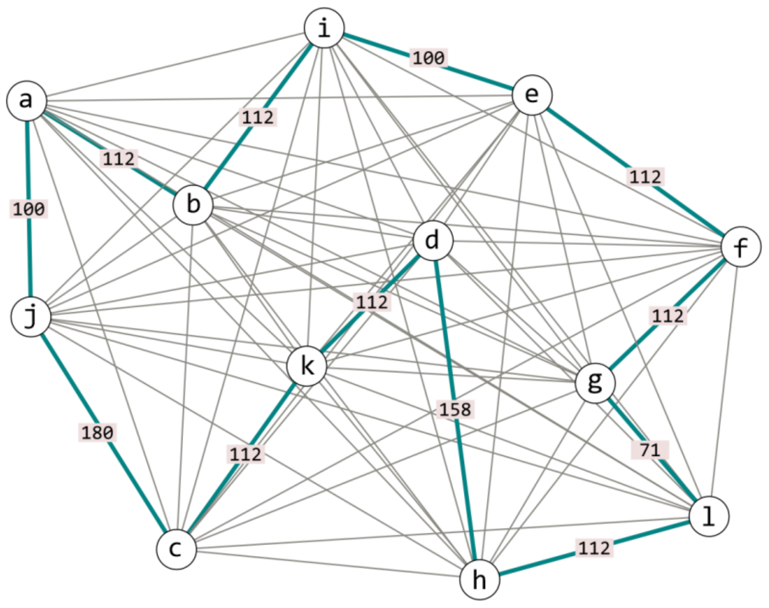

**Figure 2.** Minimal Hamiltonian cycle in graph G (|V| = 12, |E| = 66).

In the next few paragraphs, the implementation of the memetic algorithm will be described. The results obtained from its best performance (among ten of its runs) will also be presented. The values of the control parameters are set as follows: number of iterations: 100; population size: 8.

*Step 1.* Initially, the proposed algorithm randomly generates 8 solutions, labeled from S1 to S8. For that purpose, the permutations of the elements, which index the graph's vertex, are used. Thus, an initial population R0 is formed. Each solution is evaluated according to the length of the produced Hamiltonian cycle. Then, the initialization set R0 contains the following solutions {S1, S2, S3, S4, S5, S6, S7, S8}, and their scores are as follows:

S1: (e)–(i)–(h)–(f)–(l)–(d)–(a)–(b)–(j)–(k)–(c)–(g)–(e) = 2113
S2: (b)–(a)–(j)–(d)–(f)–(e)–(c)–(g)–(l)–(h)–(i)–(k)–(b) = 2043
S3: (k)–(i)–(e)–(c)–(d)–(a)–(b)–(j)–(l)–(h)–(g)–(f)–(k) = 2300
S4: (d)–(f)–(i)–(a)–(j)–(l)–(b)–(e)–(g)–(h)–(k)–(c)–(d) = 2391
S5: (f)–(e)–(g)–(c)–(k)–(b)–(l)–(h)–(d)–(a)–(j)–(i)–(f) = 2167
S6: (k)–(g)–(l)–(h)–(i)–(a)–(b)–(j)–(c)–(d)–(e)–(f)–(k) = 1961
S7: (k)–(g)–(d)–(f)–(e)–(a)–(j)–(h)–(l)–(c)–(b)–(i)–(k) = 2203
S8: (g)–(f)–(i)–(k)–(d)–(l)–(h)–(e)–(b)–(c)–(a)–(j)–(g) = 2259

*Step 2.* All solutions are sorted in ascending order, depending on their scores (calculated earlier). The solution S6 has a score of 1961, which corresponds to the length of the produced Hamiltonian cycle. The next solution, better than S6, is S2 with a score of 2043. The rest of the solutions are also arranged in ascending order: S1, S5, S7, S8, S3, and S4.

S6: (k)–(g)–(l)–(h)–(i)–(a)–(b)–(j)–(c)–(d)–(e)–(f)–(k) = 1961
S2: (b)–(a)–(j)–(d)–(f)–(e)–(c)–(g)–(l)–(h)–(i)–(k)–(b) = 2043
S1: (e)–(i)–(h)–(f)–(l)–(d)–(a)–(b)–(j)–(k)–(c)–(g)–(e) = 2113
S5: (f)–(e)–(g)–(c)–(k)–(b)–(l)–(h)–(d)–(a)–(j)–(i)–(f) = 2167
S7: (k)–(g)–(d)–(f)–(e)–(a)–(j)–(h)–(l)–(c)–(b)–(i)–(k) = 2203
S8: (g)–(f)–(i)–(k)–(d)–(l)–(h)–(e)–(b)–(c)–(a)–(j)–(g) = 2259
S3: (k)–(i)–(e)–(c)–(d)–(a)–(b)–(j)–(l)–(h)–(g)–(f)–(k) = 2300
S4: (d)–(f)–(i)–(a)–(j)–(l)–(b)–(e)–(g)–(h)–(k)–(c)–(d) = 2391

*Step 3.* Half of all solutions are selected in this step. This means the four solutions with the best scores, i.e., S6, S2, S1, and S5, will be used. The remaining solutions, labeled S7, S8, S3, and S4, are removed from the population. They will not participate in the formation of the new population—R1.

S6: (k)–(g)–(l)–(h)–(i)–(a)–(b)–(j)–(c)–(d)–(e)–(f)–(k) = 1961
S2: (b)–(a)–(j)–(d)–(f)–(e)–(c)–(g)–(l)–(h)–(i)–(k)–(b) = 2043
S1: (e)–(i)–(h)–(f)–(l)–(d)–(a)–(b)–(j)–(k)–(c)–(g)–(e) = 2113
S5: (f)–(e)–(g)–(c)–(k)–(b)–(l)–(h)–(d)–(a)–(j)–(i)–(f) = 2167

*Step 4.* The selected solutions of Step 3, S6, S2, S1, and S5, are combined in pairs. Various methods can be used for this purpose. One possible method is based on using the solutions' results in descending order and every two consecutive solutions create a new pair, i.e., S6 with S2 and S1 with S5 are new pairs. Another pairing process is based on the criterion of the largest difference in solutions evaluations. This means S6 with S5 and S2 with S1 are new pairs. Random pairing is also possible. Our method uses the first described pairing process, based on using the solutions' results in descending order. Thus, it forms a new parent pair P1 = {S6, S1} and P2 = {S2, S5}.

P1 = {S6: (k)–(g)–(l)–(h)–(i)–(a)–(b)–(j)–(c)–(d)–(e)–(f)–(k) = 1961
S1: (e)–(i)–(h)–(f)–(l)–(d)–(a)–(b)–(j)–(k)–(c)–(g)–(e) = 2113}
P2 = {S2: (b)–(a)–(j)–(d)–(f)–(e)–(c)–(g)–(l)–(h)–(i)–(k)–(b) = 2043
S5: (f)–(e)–(g)–(c)–(k)–(b)–(l)–(h)–(d)–(a)–(j)–(i)–(f) = 2167}

The ordering of the elements in the formed sets P1 = {S6, S1} and P2 = {S2, S5} is relevant to the crossover operator.

*Step 5.* The combined pairs of solutions (2 in total, P1 and P2) are new parents. After applying the genetic crossover operator, two new solutions are generated from each pair of parents. These descendants are denoted by C1 and C2 (generated by P1), and C3 and C4 (generated by P2). The total number of these solutions should match the total number of removed solutions in Step 3 (i.e., 4 in number).

The crossover operator uses a single crossover point. With it, the two parent solutions are divided in the same position into two halves. In this case, these are the elements S6.1, S6.2, S1.1, and S1.2 without the last vertices of each solution.

S6.1: (k)–(g)–(l)–(h)–(i)–(a)
S6.2: (b)–(j)–(c)–(d)–(e)–(f)
S1.1: (e)–(i)–(h)–(f)–(l)–(d)
S1.2: (a)–(b)–(j)–(k)–(c)–(g)

The new solutions are formed by combining a part of one parent with a part of the other parent. For instance, descendant C1 is generated by combining S6.1 and S1.2, while descendant C2 results from the union of S1.1 and S6.2.

C1: (k)–(g)–(l)–(h)–(i)–(a)–(a)–(b)–(j)–(k)–(c)–(g)

C2: (e)–(i)–(h)–(f)–(l)–(d)–(b)–(j)–(c)–(d)–(e)–(f)

The used approach has undesirable effects on some specific problems. This effect is expressed by the new solutions generated with duplicate elements or completely missing possible solutions. Such decisions are considered invalid and subsequently not used.

In the presented example, solution C1 has three duplicated elements, (a), (g), and (k). The elements (d), (e), and (f) are missing. In solution C2, the duplicate and missing elements are symmetrically swapped, i.e., the situation is just opposite.

To solve the described problem, different approaches to the crossover operator have been proposed [9]. A possible solution is that the first halves of the descendants are identical to those taken from their parents (as is our case), and the remaining half is generated afterward, by filling in missing elements, only in the order of their appearance in the other parent.

The described approach is applied to convert the unacceptable solutions C1 and C2 into acceptable ones. The items, duplicated in both solutions, are marked as shown below:

C1: (k)–(g)–(l)–(h)–(i)–(a)–(a)–(b)–(j)–(k)–(c)–(g)

C2: (e)–(i)–(h)–(f)–(l)–(d)–(b)–(j)–(c)–(d)–(e)–(f)

Solutions C1 and C2 are transformed into acceptable solutions by swapping duplicate elements (from their second halves) in the order they appear. Specifically, in C1, element (a) is exchanged with (d) from C2, (k) with (e), and (g) with (f). This process leads to solutions with acceptable orders of the elements. At last, the first element is added as the last to "close" the Hamiltonian cycle:

C1: (k)–(g)–(l)–(h)–(i)–(a)–(d)–(b)–(j)–(e)–(c)–(f)–(k)

C2: (e)–(i)–(h)–(f)–(l)–(d)–(b)–(j)–(c)–(a)–(k)–(g)–(e)

The situation for the solutions C3 and C4 is similar. With that action, the creation of the 4 new solutions (descendants) is completed.

*Step 6.* Some of the new solutions can be modified by applying the genetic mutation operator. This operator modifies the solution by altering one or more of its elements.

In the considered example, from all the new solutions—C1, C2, C3, and C4—a solution C2 is randomly chosen to mutate.

C2: (e)–(i)–(h)–(f)–(l)–(d)–(b)–(j)–(a)–(c)–(k)–(g)–(e)

After this step, the new solutions have a quantitative measure of their quality—the length of the formed Hamiltonian cycle.

*Step 7.* At this step of the algorithm, the parents and descendants are merged. The algorithm transitions to Step 2 and repeats the processes.

*Step 2′.* All solutions are sorted in ascending order, depending on their score (calculated earlier). The solution S6 has a score of 1961, which corresponds to the length of the formed Hamiltonian cycle, given the order for traversing the vertices of the graph. The next higher-rated solution is solution number C3 with a score of 2007. Next is solution number C2 with a score of 2032 and so on. The remaining solutions with the numbers S2, S1, S5, C4, and C1 are also arranged.

S6: (k)–(g)–(l)–(h)–(i)–(a)–(b)–(j)–(c)–(d)–(e)–(f)–(k) = 1961

C3: (b)–(a)–(j)–(d)–(f)–(e)–(l)–(h)–(c)–(g)–(k)–(i)–(b) = 2007

C2: (e)–(i)–(h)–(f)–(l)–(d)–(b)–(j)–(a)–(c)–(k)–(g)–(e) = 2032

S2: (b)–(a)–(j)–(d)–(f)–(e)–(c)–(g)–(l)–(h)–(i)–(k)–(b) = 2043

S1: (e)–(i)–(h)–(f)–(l)–(d)–(a)–(b)–(j)–(k)–(c)–(g)–(e) = 2113

S5: (f)–(e)–(g)–(c)–(k)–(b)–(l)–(h)–(d)–(a)–(j)–(i)–(f) = 2167

C4: (f)–(e)–(g)–(c)–(k)–(b)–(d)–(a)–(l)–(h)–(i)–(j)–(f) = 2645

C1: (k)–(g)–(l)–(h)–(i)–(a)–(d)–(b)–(j)–(e)–(c)–(f)–(k) = 2657

Applying crossover and mutation to the four newly generated solutions yields C3 and C2, which outperform their parent solutions, S1 and S5. These two solutions will replace the two parent solutions when the algorithm moves to the next iteration cycle (Step 3). Next, half of all solutions are selected. In our case, these are the first four solutions with the best results—S6, C3, C2, and S2.

The remaining solutions (S1, S5, C4, and C1) will be removed from the population. They will not participate in the forming of the new population—R2.

After 100 iteration cycles (reproductions), the best solution found by the genetic algorithm and presented in Figure 3 has a value of 1639.

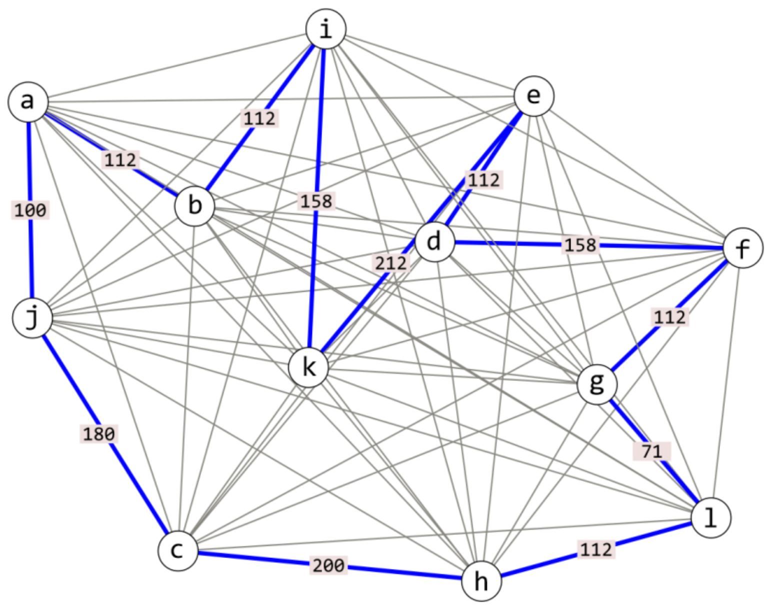

**Figure 3.** The best-found solution by the genetic algorithm after 100 iterations.

(a)–(b)–(i)–(k)–(e)–(d)–(f)–(g)–(l)–(h)–(c)–(j)–(a)

112 + 112 + 158 + 212 + 112 + 158 + 112 + 71 + 112 + 200 + 180 + 100 = 1639

To present the transformation of the genetic algorithm into a memetic one, an example applying a method based on local search will be given. The local search method will be applied for solution S6. That means an additional Step 7.1 is introduced. The local search method was developed by the authors and presented in detail in [9].

We temporarily remove the last element (vertex) from the cycle—(k)—then distribute the remaining elements into three groups—A, B, and C—with 4 elements each.

S6: A = {(k)–(g)–(l)–(h)}, B = {(i)–(a)–(b)–(j)}, C = {(c)–(d)–(e)–(f)}

All the vertices in the first group are moved so that the vertices (g), (l), and (h) are each placed in the first position (in the group). The order with the initial vertex (k) does not need to be checked because solution S6 is identical to this order. The following four (of which three are new) solutions are formed:

S6.A.1: (k)–(g)–(l)–(h)–(i)–(a)–(b)–(j)–(c)–(d)–(e)–(f)–(k) = 1961

S6.A.2: (g)–(l)–(h)–(k)–(i)–(a)–(b)–(j)–(c)–(d)–(e)–(f)–(g) = 1673

S6.A.3: (l)–(h)–(k)–(g)–(i)–(a)–(b)–(j)–(c)–(d)–(e)–(f)–(l) = 1894

S6.A.4: (h)–(k)–(g)–(l)–(i)–(a)–(b)–(j)–(c)–(d)–(e)–(f)–(h) = 1998

The initial application of the local search method yielded a significant improvement with solution S6.A.2. Notably, the starting and ending vertices of the vertex traversal sequence also differed. While the resulting score is still considerably below the optimal value of 1393, this initial step represents substantial progress.

The elements in the groups (tentatively labeled B and C) are shuffled cyclically without changing the best arrangement of group A.

We note that if the local search method is applied as described, the last generated best solution cannot be lost, i.e., either some improvement will be achieved, or the previous best solution will stay unchanged.

The best solution found after 100 generated reproductions, from the genetic algorithm without applying the local search method, has a value of 1639. On the other hand, when the local search method is used with the same values of the control parameters, the best solution found has a value of 1425 (Figure 4).

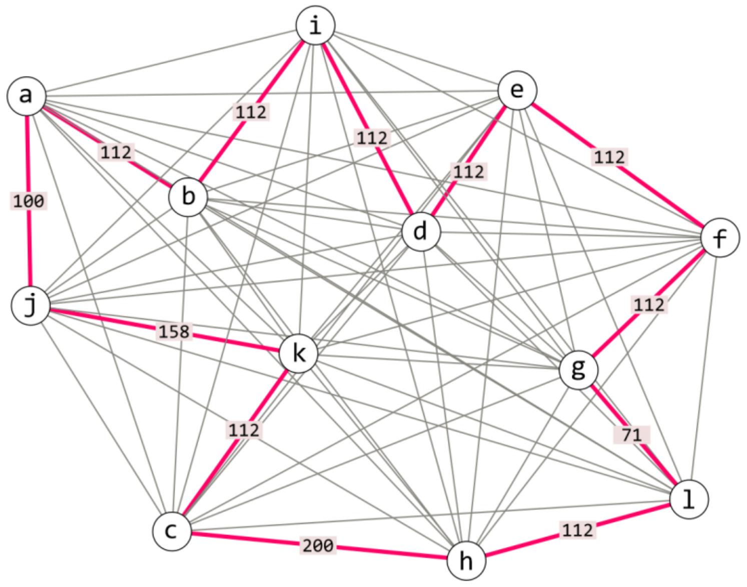

**Figure 4.** The best-found solution by the memetic algorithm after 100 iterations.

(b)—(i)—(d)—(e)—(f)—(g)—(l)—(h)—(c)—(k)—(j)—(a)—(b)
112 + 112 + 112 + 112 + 112 + 71 + 112 + 200 + 112 + 158 + 100 + 112 = 1425
Expanding the population size to 64 while maintaining 100 reproductions enhanced the performance of both algorithms. Under these conditions, the genetic algorithm produced a solution with a value of 1518. Conversely, the memetic algorithm achieved the optimal solution, scoring 1393, a result that matches the performance of the same algorithm using the backtracking method. When increasing the number of solutions in the population, the memetic algorithm finds the optimal solution very quickly: in only 47 ms with 4596 solutions generated. In contrast, the exact algorithm, which uses the backtracking method, found the best solution after 2625 ms, after testing a total of 48,280,595 solutions. The quality of the solutions generated by the memetic algorithm is a function of the population size.

## 3. Results

The two approximate algorithms—genetic, using the evolutionary techniques of crossover and mutation, and memetic, based on the same evolutionary techniques but supplemented with a local search method—are the object of this paper. This research aims to conduct a comparative analysis of approximate algorithms, evaluating their performance on identical input data while generating a comparable number of solutions. Additionally, the study addresses the impact of local search methods and genetic crossover operators on the memetic algorithm's solution quality.

If the number of vertices in each complete undirected graph $G$ is denoted by $n$, then the number of edges $m$ in this graph can be calculated using the following formula:

$$m = \frac{n(n-1)}{2} \tag{1}$$

All vertices have the same degree ($d = n - 1$) since each vertex is connected to exactly $n - 1$ edges. The total number of Hamiltonian cycles in a complete undirected graph can be calculated by the following formula:

$$Hamiltonian\ cycles = \frac{(n-1)!}{2} \tag{2}$$

To experiment, 17 complete undirected graphs (G_n_m) were created, G_12_66, G_13_78, . . ., G_28_378, and the coordinates (in pixels) of all vertices were randomly generated. Each graph was created by adding one more vertex to the available ones, as well as adding edges that connected that vertex to the others.

The experiments were performed on a standard computer configuration with the Win 11 Pro 64-bit operating system installed and the following hardware configuration: CPU: Intel® Core™ i7-4712MQ CPU @ 2.30 GHz 2.30 GHz; RAM: 8.00 GB.

The results obtained from the experiments will be presented in the following paragraphs. The aim is to investigate the behavior of the approximate algorithms with the same input data. The main parameters that need to be analyzed are the population size (i.e., the number of solutions in the set of generated solutions) and the number of reproductions (i.e., the number of iterations performed by both algorithms—genetic and memetic).

The results of the experiments with the approximate algorithms will be analyzed in three stages. First, the results obtained after running the genetic algorithm on all test graphs, from G_12_66 to G_28_378, are analyzed. Second, the results from the execution of the memetic algorithm on the same graphs are analyzed. Third, a comparative analysis will be made between the genetic and memetic algorithms by comparing the solutions generated by the genetic mutation operators, total solutions generated, execution time, best result obtained, and best run out of total number of runs.

The number of crossover solutions is determined by the number of reproductions and the population size, while the number of mutated solutions is set as a percentage of the crossover solutions.

Tables 2 and 3 present the summarized results of executing the genetic and memetic algorithms.

**Table 2.** Summary results of the genetic algorithm.

| Graph | Crossover Solutions | Mutation Solutions | Total Solutions | Time (ms) | Solutions Per ms | Mutated Solutions (%) |
|---|---|---|---|---|---|---|
| G_12_66 | 662,640 | 248,738 | 911,378 | 5125 | 178 | 37.5% |
| G_13_78 | 783,120 | 352,439 | 1,135,559 | 6451 | 176 | 45.0% |
| G_14_91 | 843,360 | 488,904 | 1,332,264 | 7611 | 175 | 58.0% |
| G_15_105 | 978,900 | 644,861 | 1,623,761 | 8969 | 181 | 65.9% |
| G_16_120 | 1,124,480 | 857,726 | 1,982,206 | 10,887 | 182 | 76.3% |
| G_17_136 | 1,280,100 | 896,428 | 2,176,528 | 12,423 | 175 | 70.0% |
| G_18_153 | 1,355,400 | 863,862 | 2,219,262 | 11,755 | 189 | 63.7% |
| G_19_171 | 1,526,080 | 849,710 | 2,375,790 | 12,932 | 184 | 55.7% |
| G_20_190 | 1,706,800 | 853,826 | 2,560,626 | 14,029 | 183 | 50.0% |
| G_21_210 | 1,897,560 | 1,080,615 | 2,978,175 | 16,629 | 179 | 56.9% |
| G_22_231 | 1,987,920 | 1,244,005 | 3,231,925 | 17,316 | 187 | 62.6% |
| G_23_253 | 2,193,740 | 1,268,791 | 3,462,531 | 19,942 | 174 | 57.8% |
| G_24_276 | 2,289,120 | 1,294,092 | 3,583,212 | 20,538 | 174 | 56.5% |
| G_25_300 | 2,635,500 | 1,815,082 | 4,450,582 | 23,664 | 188 | 68.9% |
| G_26_325 | 2,740,920 | 1,798,503 | 4,539,423 | 24,512 | 185 | 65.6% |
| G_27_351 | 2,981,880 | 1,881,312 | 4,863,192 | 25,896 | 188 | 63.1% |
| G_28_378 | 3,232,880 | 1,900,576 | 5,133,456 | 28,049 | 183 | 58.8% |

The obtained values show that for all test graphs, the number of solutions generated within 1 ms is comparable with the deviation from the column average (equal to 181 ms). Two conclusions can be made from the obtained values. First, in 14 out of a total of 17 cases of test graphs, the number of solutions that have mutated exceeds 50% of the number of solutions generated by the crossover operator. This means that the populations contain many identical solutions, which leads to a faster onset of the convergence level. Second, the results show that the deviation from the column average (equal to 59.6%) is in the range [–16.68%, 22.06%], which indicates the stochastic nature of the crossover operator in genetic algorithms.

**Table 3.** Summary results of the memetic algorithm.

| Graph | Crossover and Local Search Solutions | Mutation Solutions | Total Solutions | Time (ms) | Solutions Per ms | Mutated Solutions (%) |
|---|---|---|---|---|---|---|
| G_12_66 | 662,640 | 82,167 | 744,807 | 5470 | 136 | 12.4% |
| G_13_78 | 783,120 | 117,468 | 900,588 | 6496 | 139 | 15.0% |
| G_14_91 | 843,360 | 102,889 | 946,249 | 6986 | 135 | 12.2% |
| G_15_105 | 978,900 | 74,396 | 1,053,296 | 8366 | 126 | 7.6% |
| G_16_120 | 1,124,480 | 94,456 | 1,218,936 | 9247 | 132 | 8.4% |
| G_17_136 | 1,280,100 | 120,329 | 1,400,429 | 11,327 | 124 | 9.4% |
| G_18_153 | 1,355,400 | 121,986 | 1,477,386 | 12,159 | 122 | 9.0% |
| G_19_171 | 1,526,080 | 131,242 | 1,657,322 | 13,539 | 122 | 8.6% |
| G_20_190 | 1,706,800 | 150,198 | 1,856,998 | 14,472 | 128 | 8.8% |
| G_21_210 | 1,897,560 | 170,780 | 2,068,340 | 16,665 | 124 | 9.0% |
| G_22_231 | 1,987,920 | 182,888 | 2,170,808 | 16,475 | 132 | 9.2% |
| G_23_253 | 2,193,740 | 206,211 | 2,399,951 | 19,954 | 120 | 9.4% |
| G_24_276 | 2,289,120 | 226,502 | 2,515,622 | 20,938 | 120 | 9.9% |
| G_25_300 | 2,635,500 | 253,008 | 2,888,508 | 23,041 | 125 | 9.6% |
| G_26_325 | 2,740,920 | 268,610 | 3,009,530 | 24,056 | 125 | 9.8% |
| G_27_351 | 2,981,880 | 250,477 | 3,232,357 | 24,758 | 131 | 8.4% |
| G_28_378 | 3,232,880 | 278,027 | 3,510,907 | 28,734 | 122 | 8.6% |

The obtained values show that for all test graphs, the number of solutions generated within 1 ms is comparable with the deviation from the column average (equal to 127 ms). From the obtained values, it can be seen that only in 3 cases (out of a total of 17), the number of solutions that have mutated exceeds 10% of the number of solutions generated by the crossover operator, summed with those of the local search method. This means that the populations of the memetic algorithm contain many unique solutions.

Comparative analyses between the generated solutions, the execution time, and the number of starts of the genetic and memetic algorithms are presented in Table 4.

**Table 4.** Summary results between genetic and memetic algorithms.

| Graph | GA Result (px) | GA Time (ms) | GA Runs | MA Result (px) | MA Time (ms) | MA Runs |
|---|---|---|---|---|---|---|
| G_12_66 | 1320 | 5125 | 4/4 | 1320 | 5470 | 1/1 |
| G_13_78 | 1388 | 6451 | 3/3 | 1388 | 6496 | 1/1 |
| G_14_91 | 1461 | 7611 | 7/7 | 1461 | 6986 | 1/1 |
| G_15_105 | 1512 | 8969 | 1/1 | 1512 | 8366 | 1/1 |
| G_16_120 | 1568 | 10,887 | 5/5 | 1568 | 9247 | 1/1 |
| G_17_136 | 1681 | 12,423 | 8/8 | 1681 | 11,327 | 4/4 |
| G_18_153 | 1718 | 11,755 | 2/2 | 1718 | 12,159 | 7/7 |
| G_19_171 | 1963 | 12,932 | 6/10 | 1941 | 13,539 | 3/10 |
| G_20_190 | 2042 | 14,029 | 4/4 | 2042 | 14,472 | 8/8 |
| G_21_210 | 2115 | 16,629 | 5/10 | 2085 | 16,665 | 6/10 |
| G_22_231 | 2170 | 17,316 | 7/10 | 2158 | 16,475 | 4/10 |
| G_23_253 | 2353 | 19,942 | 3/10 | 2296 | 19,954 | 2/10 |
| G_24_276 | 2434 | 20,538 | 4/10 | 2357 | 20,938 | 3/10 |
| G_25_300 | 2359 | 23,664 | 2/10 | 2359 | 23,041 | 8/10 |
| G_26_325 | 2490 | 24,512 | 8/10 | 2452 | 24,056 | 5/10 |
| G_27_351 | 2549 | 25,896 | 3/10 | 2519 | 24,758 | 7/10 |
| G_28_378 | 2590 | 28,049 | 7/10 | 2543 | 28,734 | 4/10 |

Table 4 shows that the genetic algorithm found 9 out of a total of 11 known optimal solutions (for the first columns up to G_21_210 inclusive). In only two cases (in graphs G_19_171 and G_21_210), the genetic algorithm failed to find the optimal solutions but found solutions that were close to them (in value). In contrast to the genetic algorithm, the memetic algorithm found all 11 known optimal solutions, and in 5 out of 11 cases, this happened on its first run. For the remaining seven graphs (from G_22_231 to G_28_378), the memetic algorithm found solutions that in six out of seven cases were better than those found by the genetic algorithm, and in only one of the cases—for graph G_25_300—the found solution and from both algorithms is the same (i.e., the Hamiltonian cycle found for the graph G_25_300 from both algorithms is the same).

From these results, it can be concluded that when the number of possible Hamiltonian cycles increases, it becomes increasingly more "difficult" for the genetic algorithm to find the optimal solutions. This indicates that the population size (which is many times larger than that of the memetic algorithm) must be increased or the reproductions generated (which are also many times larger than those performed by the memetic algorithm) must be increased.

The execution time of both algorithms is comparable, although the total number of solutions generated by the genetic algorithm is more than that generated by the memetic algorithm. This trend is presented in Figure 5.

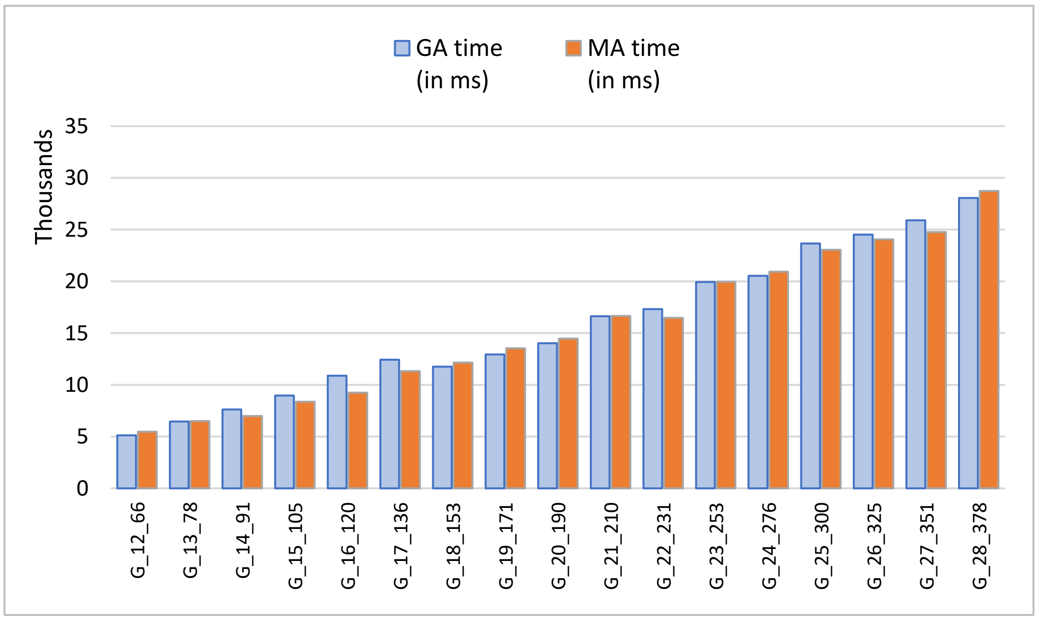

**Figure 5.** Comparison between execution times for genetic algorithm and memetic algorithm.

Figure 5 shows that the time for executing the memetic and the genetic algorithms when searching for solutions of the same graphs is comparable, even if the genetic algorithm generates more solutions than the memetic one. Since the number of solutions generated by the genetic algorithm in 1 millisecond is greater than the number of solutions generated by the memetic algorithm for the same time (between 35% and 42%), the memetic algorithm is more efficient.

The results show that the memetic algorithm always finds a better or equal solution to what the genetic algorithm found. This trend can be seen in Figure 6.

Although the differences in the values of the solutions found by the two algorithms are not large (except for graph G_25_300, where they are equal), finding better solutions for the concrete NP-hard problem is a significant success. To solve this problem, the memetic algorithm can be used because it finds optimal or near-optimal solutions. Moreover, this algorithm finds solutions in an acceptable time.

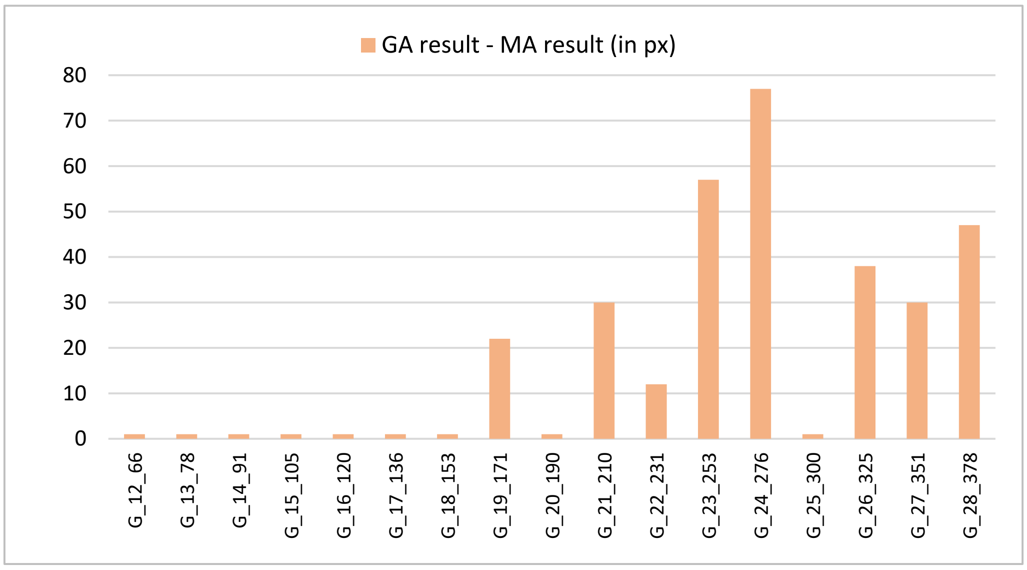

**Figure 6.** Differences between results found by the genetic algorithm and memetic algorithm.

## 4. Conclusions

When the number of Hamiltonian cycles increases, the memetic algorithm can find the optimal solutions (or close to them) faster than the genetic algorithm. Although the population using the genetic algorithm is larger than that of the memetic algorithm, the execution time of both algorithms is comparable. In most of the examined graphs, the number of solutions that mutated when executing the genetic algorithm exceeded 50% of all solutions generated by the crossover operator. This means that most populations contain many identical solutions, which leads to a faster convergence. Unlike the genetic algorithm, the number of mutated solutions by the memetic algorithm rarely exceeds 10% of the number of all solutions generated by the crossover operator, summed with those from the local search method. This means that the populations generated by the memetic algorithm contain many unique solutions.

The results show that combining evolutionary algorithms with local search techniques provides a powerful tool for solving combinatorial optimization problems. Exact algorithms based on various techniques, such as backtracking and its optimized variants, such as the "branch and bound" method, make it possible to find optimal solutions (in an acceptable time), but only to problems with a small input data size. In contrast, approximate algorithms often have linear complexity or complexity described by a low-degree polynomial. When analyzing the computational complexity of these algorithms (genetic and/or memetic), it is necessary to consider the influence of the "population size" and "number of generated reproductions" parameters. Fine-tuning them can significantly affect the overall execution time of both algorithms. Therefore, finding methods to determine the optimal values of the control parameters (in terms of total execution time) is essential. The guidelines for research development include a comparative analysis of the proposed method with other known methods and common datasets (benchmarks).

**Author Contributions:** Conceptualization, R.K. and V.K.; methodology, R.K.; software, R.K. and V.K.; formal analysis, R.K.; investigation, R.K.; data curation, V.K.; writing—original draft preparation, R.K. and V.K.; writing—review and editing, R.K.; visualization, V.K. All authors have read and agreed to the published version of the manuscript.

**Funding:** The APC was funded by MDPI under grant a5b667bc1618fac8 to R.K.

**Data Availability Statement:** The data involved in the paper are available upon reasonable request to the corresponding author.

**Acknowledgments:** The authors would like to thank the reviewers for their constructive comments.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## References

1. Senbagamalar, L.; Logeswari, S. Genetic Clustering Algorithm-Based Feature Selection and Divergent Random Forest for Multiclass Cancer Classification Using Gene Expression Data. *Int. J. Comput. Intell. Syst.* **2024**, *17*, 23. [CrossRef]
2. Wang, Y. A genetic algorithm with the mixed heuristics for traveling salesman problem. *Int. J. Comput. Intell. Appl.* **2015**, *14*, 1550003. [CrossRef]
3. Pyrih, Y.; Klymash, M.; Kaidan, M.; Strykhalvuk, B. Investigating the Efficiency of Tournament Selection Operator in Genetic Algorithm for Solving TSP. In Proceedings of the 5th IEEE International Conference on Advanced Information and Communication Technologies, AICT 2023, Lviv, Ukraine, 21–25 November 2023; pp. 170–173. [CrossRef]
4. Yamada, M. 1/f Noise in the Simple Genetic Algorithm Applied to a Traveling Salesman Problem. *Fluct. Noise Lett.* **2017**, *16*, 1750026. [CrossRef]
5. Meneses, S.; Cueva, R.; Tupia, M.; Guanira, M. A genetic algorithm to solve 3D traveling salesman problem with initial population based on a GRASP algorithm. *J. Comput. Methods Sci. Eng.* **2017**, *17*, S1–S10. [CrossRef]
6. Zhang, T.; Zhou, Y.; Zhou, G.; Deng, W.; Luo, Q. Discrete Mayfly Algorithm for spherical asymmetric traveling salesman problem. *Expert Syst. Appl.* **2023**, *221*, 119765. [CrossRef]
7. Maity, S.; Roy, A.; Maiti, M. A Modified Genetic Algorithm for solving uncertain Constrained Solid Travelling Salesman Problems. *Comput. Ind. Eng.* **2015**, *83*, 273–296. [CrossRef]
8. Victer Paul, P.; Moganarangan, N.; Kumar, S.S.; Raju, R.; Vengattaraman, T.; Dhavachelvan, P. Performance analyses over population seeding techniques of the permutation-coded genetic algorithm: An empirical study based on traveling salesman problems. *Appl. Soft Comput. J.* **2015**, *32*, 383–402. [CrossRef]
9. Kralev, V.; Kraleva, R.; Kumar, S. A modified event grouping based algorithm for the university course timetabling problem. *Int. J. Adv. Sci. Eng. Inf. Technol.* **2019**, *9*, 229–235. [CrossRef]
10. Romaguera, D.; Plender-Nabas, J.; Matias, J.; Austero, L. Development of a Web-based Course Timetabling System based on an Enhanced Genetic Algorithm. *Procedia Comput. Sci.* **2024**, *234*, 1714–1721. [CrossRef]
11. Plante, J.-F.; Larocque, M.; Adès, M. Objective model selection with parallel genetic algorithms using an eradication strategy. *Can. J. Stat.* **2024**, *52*, 636–654. [CrossRef]
12. Wang, Z. Optimal Scheduling of Flow Shop Based on Genetic Algorithm. *J. Adv. Manuf. Syst.* **2022**, *21*, 111–123. [CrossRef]
13. Alekseev, V.E.; Boliac, R.; Korobitsyn, D.V.; Lozin, V.V. NP-hard graph problems and boundary classes of graphs. *Theor. Comput. Sci.* **2007**, *389*, 219–236. [CrossRef]
14. Ilin, V.; Simić, D.; Simić, S.D.; Simić, S.; Saulić, N.; Calvo-Rolle, J.L. A hybrid genetic algorithm, list-based simulated annealing algorithm, and different heuristic algorithms for travelling salesman problem. *Log. J. IGPL* **2023**, *31*, 602–617. [CrossRef]
15. Yuan, S.; Skinner, B.; Huang, S.; Liu, D. A new crossover approach for solving the multiple travelling salesmen problem using genetic algorithms. *Eur. J. Oper. Res.* **2013**, *228*, 72–82. [CrossRef]
16. Tsai, C.W.; Tseng, S.P.; Chiang, M.C.; Yang, C.S.; Hong, T.P. A high-performance genetic algorithm: Using traveling salesman problem as a case. *Sci. World J.* **2014**, *2014*, 178621. [CrossRef]
17. Alkafaween, E.; Hassanat, A.; Essa, E.; Elmougy, S. An Efficiency Boost for Genetic Algorithms: Initializing the GA with the Iterative Approximate Method for Optimizing the Traveling Salesman Problem—Experimental Insights. *Appl. Sci.* **2024**, *14*, 3151. [CrossRef]
18. Skorpil, V.; Oujezsky, V. Parallel Genetic Algorithms' Implementation Using a Scalable Concurrent Operation in Python. *Sensors* **2022**, *22*, 2389. [CrossRef]
19. Moscato, P.; Norman, M.G. A memetic approach for the traveling salesman problem implementation of a computational ecology for combinatorial optimization on message-passing systems. In *Book Parallel Computing and Transputer Applications*; CIMNE: Barcelona, Spain, 1992; pp. 177–186.
20. Radcliffe, N.J.; Surry, P.D. Formal memetic algorithms. In *AISB Workshop on Evolutionary Computing*; Springer: Berlin/Heidelberg, Germany, 1994; Volume 11, pp. 1–16.
21. Badillo, A.R.; Cotta, C.; Fernández-Leiva, A.J. Towards user-centric memetic algorithms: Experiences with the TSP. *Lect. Notes Comput. Sci. (Lect. Notes Artif. Intell. Lect. Notes Bioinform.)* **2011**, *6692*, 284–291. [CrossRef]
22. Luo, Z.; Zou, G.; Li, Z.; Chen, S.; Xu, J. A memetic ready-mixed concrete scheduling method based on bidirectional collaborative optimisation for highway construction. *Int. J. Embed. Syst.* **2023**, *15*, 505–515. [CrossRef]
23. Wang, Y.; Chen, Y.; Lin, Y. Memetic algorithm based on sequential variable neighborhood descent for the minmax multiple traveling salesman problem. *Comput. Ind. Eng.* **2017**, *106*, 105–122. [CrossRef]
24. Samanlioglu, F.; Ferrell, W.G.; Kurz, M.E. An interactive memetic algorithm for production and manufacturing problems modelled as a multi-objective travelling salesman problem. *Int. J. Prod. Res.* **2012**, *50*, 5671–5682. [CrossRef]
25. Candeias, J.; de Araújo, D.R.B.; Miranda, P.; Bastos-Filho, C.J.A. Memetic evolutionary algorithms to design optical networks with a local search that improves diversity. *Expert Syst. Appl.* **2023**, *232*, 120805. [CrossRef]
26. Koh, K.M.; Dong, F.; Tay, E.G. *Introduction to Graph Theory: With Solutions to Selected Problems*; World Scientific Publishing: Singapore, 2023; pp. 1–294. [CrossRef]

27. Bacheti, G.G.; Camargo, R.S.; Amorim, T.S.; Yahyaoui, I.; Encarnação, L.F. Model-Based Predictive Control with Graph Theory Approach Applied to Multilevel Back-to-Back Cascaded H-Bridge Converters. *Electronics* **2022**, *11*, 1711. [CrossRef]
28. El-Samak, A.F.; Ashour, W. Optimization of traveling salesman problem using affinity propagation clustering and genetic algorithm. *J. Artif. Intell. Soft Comput. Res.* **2015**, *5*, 239–245. [CrossRef]
29. Maskooki, A.; Kallio, M. A bi-criteria moving-target travelling salesman problem under uncertainty. *Eur. J. Oper. Res.* **2023**, *309*, 271–285. [CrossRef]
30. Kuchaki Rafsanjani, M.; Eskandari, S.; Borumand Saeid, A. A similarity-based mechanism to control genetic algorithm and local search hybridization to solve traveling salesman problem. *Neural Comput. Appl.* **2015**, *26*, 213–222. [CrossRef]
31. Kralev, V. An Analysis of a Recursive and an Iterative Algorithm for Generating Permutations Modified for Travelling Salesman Problem. *Int. J. Adv. Sci. Eng. Inf. Technol.* **2017**, *7*, 1685–1692. [CrossRef]
32. El Idrissi, A.L.; Tajani, C. Genetic algorithm with immigration strategy to solve the fixed charge transportation problem. *Indones. J. Electr. Eng. Comput. Sci.* **2023**, *31*, 313–320. [CrossRef]
33. He, M.; Wu, Q.; Benlic, U.; Lu, Y.; Chen, Y. An effective multi-level memetic search with neighborhood reduction for the clustered team orienteering problem. *Eur. J. Oper. Res.* **2024**, *318*, 778–801. [CrossRef]
34. Dang, X.; Gong, D.; Yao, X.; Tian, T.; Liu, H. Enhancement of Mutation Testing via Fuzzy Clustering and Multi-Population Genetic Algorithm. *IEEE Trans. Softw. Eng.* **2022**, *48*, 2141–2156. [CrossRef]
35. Applegate, D.L.; Bixby, R.E.; Chvátal, V.; Cook, W.J. *The Traveling Salesman Problem: A Computational Study*; Princeton University Press: Princeton, NJ, USA, 2011; pp. 1–593.
36. Zhukova, G.N.; Ul'yanov, M.V.; Fomichev, M.I. A Hybrid Exact Algorithm for the Asymmetric Traveling Salesman Problem: Construction and a Statistical Study of Computational Efficiency. *Autom. Remote Control* **2019**, *80*, 2054–2067. [CrossRef]
37. de Oliveira, S.M.; Bezerra, L.C.T.; Stützle, T.; Dorigo, M.; Wanner, E.F.; de Souza, S.R. A computational study on ant colony optimization for the traveling salesman problem with dynamic demands. *Comput. Oper. Res.* **2021**, *135*, 105359. [CrossRef]
38. Pop, P.C.; Cosma, O.; Sabo, C.; Sitar, C.P. A comprehensive survey on the generalized traveling salesman problem. *Eur. J. Oper. Res.* **2024**, *314*, 819–835. [CrossRef]
39. Battarra, M.; Pessoa, A.A.; Subramanian, A.; Uchoa, E. Exact algorithms for the traveling salesman problem with draft limits. *Eur. J. Oper. Res.* **2014**, *235*, 115–128. [CrossRef]
40. Kinable, J.; Smeulders, B.; Delcour, E.; Spieksma, F.C.R. Exact algorithms for the Equitable Traveling Salesman Problem. *Eur. J. Oper. Res.* **2017**, *261*, 475–485. [CrossRef]
41. Hussain, A.; Muhammad, Y.S.; Nauman Sajid, M.; Hussain, I.; Mohamd Shoukry, A.; Gani, S. Genetic Algorithm for Traveling Salesman Problem with Modified Cycle Crossover Operator. *Comput. Intell. Neurosci.* **2017**, *2017*, 7430125. [CrossRef]
42. Wollmann, J.; Muschalski, L.; Wang, Z.; Zichner, M.; Winkler, A.; Modler, N. Application of genetic algorithm for the synthesis of path-generating compliant mechanisms. *Smart Mater. Struct.* **2024**, *33*, 015023. [CrossRef]
43. Wang, Z.; Shen, Y.; Li, S.; Wang, S. A fine-grained fast parallel genetic algorithm based on a ternary optical computer for solving traveling salesman problem. *J. Supercomput.* **2023**, *79*, 4760–4790. [CrossRef]