Computing and Informatics, Vol. 40, 2021, 1370–1397, doi: 10.31577/cai_2021_6_1370

MULTIFACTORIAL EVOLUTIONARY ALGORITHM FOR SIMULTANEOUS SOLUTION OF TSP AND TRP

Ha-Bang Ban, Dang-Hai Pham*

School of Information and Communication Technology
Hanoi University of Science and Technology
Hanoi, Vietnam
e-mail: {BangBH, HaiPD}@soict.hust.edu.vn

Abstract. We study two problems called the Traveling Repairman Problem (TRP) and Traveling Salesman Problem (TSP). The TRP wants to minimize the total time for all customers that have to wait before being served, while the TSP aims to minimize the total time to visit all customers. In this sense, the TRP takes a customer-oriented view, whereas the TSP is server-oriented. In the literature, there exist numerous algorithms that are developed for two problems. However, these algorithms are designed to solve each problem independently. Recently, Multifactorial Evolutionary Algorithm (MFEA) has been a variant of Evolutionary Algorithm (EA) aiming to solve multiple optimization tasks simultaneously. The MFEA framework has yet to be fully exploited, but the realm has recently attracted much interest from the research community. This paper proposed a new approach using the MFEA framework to solve these two problems simultaneously. The MFEA has two tasks simultaneously: the first is solving the TRP problem, and the second is solving the TSP. Experiment results show the efficiency of the proposed MFEA: 1. for small instances, the algorithm reaches the optimal solutions of both problems; 2. for large instances, our solutions are better than those of the previous MFEA algorithms.

Keywords: MFEA, TSP, TRP, EA

* Corresponding author

1 INTRODUCTION

Evolutionary Algorithms are optimization techniques that begin from natural evolutionary inspiration. They have been widely applied to solve problems in various fields. Currently, a new approach called MFEA [6, 11, 14, 26, 27, 29] has been developed to incorporate the characteristics of Evolutionary Algorithms into multitasking to handle multiple search spaces of multiple problems at the same time. The advantage of the approach is that the phenomenon of implicit genetic transfer in multitasking can exploit the presence of transferrable knowledge between optimization tasks, thereby facilitating improved performance characteristics for multiple tasks simultaneously. Moreover, instead of solving a pool of similar optimization problems singly, it handles various requests and performs multiple tasks for systems. Therefore, it decreases the response time, that is one of the most important keys of the Cloud service from the clients' perspective.

The TSP [3, 9, 10, 13, 16, 18, 19], and TRP [2, 4, 5, 7, 8, 17, 22, 23] are combinatorial optimization problems that have many practical situations. In this paper, we consider the problem in the metric case, and formulate the TSP and TRP as follows:

Given a complete graph K_n with the vertex set $V = \{v_1, v_2, \dots, v_n\}$ and a symmetric distance matrix $C = \{c(v_i, v_j) \mid i, j = 1, 2, \dots, n\}$, where $c(v_i, v_j)$ is the distance between two vertices v_i and v_j. Suppose that $T = \{v_1, \dots, v_k, \dots, v_n, v_{n+1} \equiv v_1\}$ is a tour in K_n. Denote by $P(v_1, v_k)$ the path from v_1 to v_k on this tour and by $l(P(v_1, v_k))$ its length. The arrival time of a vertex v_k $(1 < k \leq n)$ on T is the length of the path from starting vertex v_1 to v_k:

$$l(P(v_1, v_k)) = \sum_{i=1}^{k-1} c(v_i, v_{i+1}).$$

In the TRP, the cost of the tour T is defined as the sum of arrival times of all vertices:

$$\sum_{k=2}^{n} l(P(v_1, v_k)).$$

In the TSP, the salesman must return to v_1. Therefore, the cost of the tour T is defined as the sum of the length of the path from v_1 to v_{n+1} on this tour: $l(P(v_1, v_{n+1}))$. Note that: $v_{n+1} \equiv v_1$.

The TSP and TRP ask for a tour with minimum cost, which starts at a given vertex v_1 and visits each vertex in the graph exactly once.

The TRP wants to minimize the total time for all customers that have to wait before being served, while the TSP aims to minimize the total time to visit all nodes. Currently, there exist many algorithms that are proposed to solve them. However, these algorithms are designed to solve each problem independently. In this paper, we introduce the algorithm based on the MFEA framework to solve two problems simultaneously. The major contributions of this work are as follows:

- From the algorithmic design, we develop a metaheuristic based on the MFEA framework. Our metaheuristic combines MFEA and Randomized Variable Neighborhood Descent (RVND), in which MFEA ensures diversification while RVND maintains intensification. This combination maintains the right balance between diversification and intensification. It is the first metaheuristic based on the MFEA framework to solve two problems at the same time.
- From the computational perspective, extensive numerical experiments on benchmark instances show that the proposed algorithm reaches good solutions in a short time for two problems simultaneously. Moreover, it obtains better solutions than the previous MFEA algorithms in many cases.

The rest of this paper is organized as follows. Sections 2 and 3 present the literature and preliminaries, respectively. Section 4 describes the proposed algorithm. Computational evaluations are reported in Section 5. Sections 6 and 7 discuss and conclude the paper.

2 LITERATURE

2.1 MFEA

In the literature, MFEA [6, 11, 14, 26, 29] have been known as a framework that can effectively solve many optimization problems. The main advantage of MFEA is to solve multiple problems at the same time. Therefore, it can be applied in a limited computational system. Furthermore, some articles show that the genetic transfer has occurred in relevant multitasking tasks to facilitate finding optimal solutions for multiple problems at the same time [6].

Recently, some variants of MFEA are also introduced. Yuan et al. extended the first study on evolutionary multitasking [29] to develop evolutionary multitasking in permutation-based combinatorial optimization problems. They implement it on some popular combinatorial optimization problems. The experiment results show the potential scalability of evolutionary multitasking to many-task environments.

One of the first articles is introduced by Bali et al. [6] since they proposed an improved version of this algorithm. This study suggested an online *rmp* (*rmp* is the probability of crossover) estimation technique minimizing the negative interactions between optimization tasks. Osaba et al. [20] then proposed a dMFEA-II framework that exploited the complementarities among the tasks, which is often achieved via genetic information transfer. The experiments on some combinatorial optimization problems confirm the good performance of the developed dMFEA-II and concur with the insights drawn in previous studies for continuous optimization. To combine the MFEA framework with the other metaheuristic algorithm, Feng et al. [11] proposed a mechanism for combining MFEA with Particle SWarm Optimization Algorithm (PSO). In the new algorithm, a new assortative mating scheme is proposed. At the same time, the other components, such as unified individual representation, vertical cultural transmission, etc., remain unchanged as in the original MFEA. Xie et al. [26] then combined MFEA and PSO in which PSO plays the role of local search in the Multifactorial Optimization (MFO). Experimental comparisons between their algorithm and the original MFEA show that the particle swarm update operators can effectively accelerate the convergence on some benchmark problems.

2.2 TSP and TRP

The Travelling Salesman Problem (TSP) is a popular NP-hard combinatorial optimization problem studied much in the literature [3, 9, 10, 13, 16, 18, 19]. The problem can be formulated as follows: Given a set of cities and the travel costs between each pair, find the cheapest tour to visit each city exactly once, returning to the point of origin. In the formulation approach, many algorithms were designed to minimize the distance of a single-tour vehicle over the past few decades [12, 19]. In an approximation approach, numerous algorithms [9, 16, 18] were proposed in the literature.

In recent years, the interest in another NP-hard combinatorial optimisation problem known as the Traveling Repairman Problem (TRP) has grown [2, 4, 5, 7, 8, 17, 22, 23]. Given a metric space with n vertices and a tour starting at some vertex and visiting all of the others, the waiting time (or latency) of a vertex is defined to be the total distance traveled before reaching it. The Traveling Repairman Problem (TRP) asks for a tour starting at a root r and visiting all vertices, such that the total waiting time is minimized. The TRP models routing problems in which one wants to minimize the average time each customer has to wait before being served, rather than the total time to visit all vertices, as in the TSP case. In this sense, the TRP takes a customer-oriented view, whereas the TSP is server-oriented. Various works were proposed to solve the problem. We can divide them into three types:

1. The exact algorithms [2, 4, 17] solve the problem with up to 50 vertices;
2. Several approximation algorithms [7, 8] were provided in the literature. The best approximation ratio known is 3.59. However, their algorithms are quite a complexity to implement, and the best ratio is not good enough in practical situations;
3. Heuristic or metaheuristic [5, 22, 23] were used to solve the problem with larger sizes in a short time.

Figures 1 and 2 show the difference between the TRP and TSP in TSPLIB [30]. The eil51 instance includes 51 vertices that places an x-y-coordinate system on a plane. In Figure 1, we show the optimal TRP and TSP tours for this instance. The initial vertex in this instance is vertex 1. In this instance, the optimal TSP tour's total waiting time is 4.48 times as large as that of the optimal TRP tour. In [22], Salehipour et al. show that a good metaheuristic algorithm for the TSP does not produce good solutions for the TRP and vice versa.

Recently, in [1], Arellano-Arriaga et al. provided the bi-objective approach that considers a single-vehicle tour and seeks to minimize the travel time and the latency of that tour simultaneously. They called this problem the Minimum Latency-Distance Problem (MLDP). Their approach minimizes only one problem with two objective functions, while the proposed approach minimizes two objective functions simultaneously.

The above algorithms are the state-of-the-art algorithms for the TSP and TRP. However, they solve each problem well independently, but they cannot simultaneously produce good solutions for two problems. That means the proposed approach is the first algorithm that obtains two good solutions for both the TSP and TRP.

2.3 The Proposed Approach

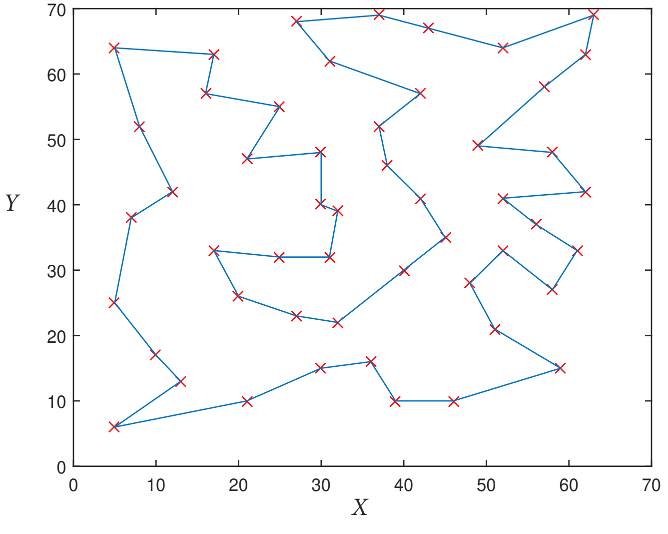

Figure 1. The optimal TSP tour on the eil51 instance

The problem is also NP-hard because it is a generalization case of the TSP and TRP. For NP-hard problems, there are three common approaches to solve the problem, namely,

1. exact algorithms,
2. approximation algorithms,
3. heuristic (or metaheuristic) algorithms:

- The exact algorithms guarantee to find the optimal solution and take exponential time in the worst case.

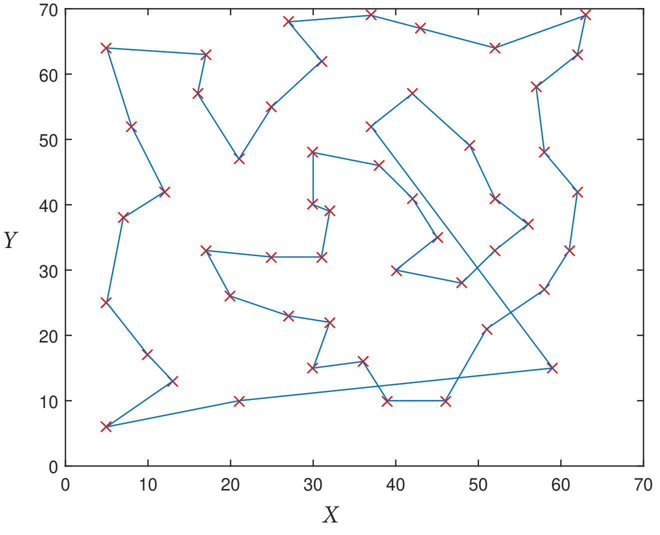

Figure 2. The optimal TRP tour on the eil51 instance

- An α-approximation algorithm produces a solution within some factor of α of the optimal solution.
- Heuristic (metaheuristic) algorithms perform well in practice and validate their empirical performance on an experimental benchmark of interesting instances.

The problem is NP-hard problem; thus, a (heuristic) or metaheuristic algorithm is a suitable approach for a short computation time. Previously, several metaheuristics have been proposed [5, 9, 16, 18, 22, 23]. However, they are designed to solve each problem independently. That means they cannot solve two problems well at the same time.

In this paper, we propose a new approach using the MFEA framework to solve two problems simultaneously. The proposed MFEA has two tasks simultaneously: the first is solving the TRP problem, and the second is solving the TSP. Experiment results show the efficiency of the proposed MFEA:

1. for small instances, the algorithm reaches the optimal solutions of both problems;
2. for large cases, our solutions are better than those of the previous MFEA approaches.

3 PRELIMINARIES

The concept of multifactorial optimization (MFO) is formalized in [6]. After that, we describe how to use the concept of MFO in EA briefly. Assume that, we have k optimization problems needed to be performed simultaneously. Without loss of generality, all tasks are assumed to be minimization problems. The j^{th} task, denoted T_i, has objective function $f_j : X_j \Rightarrow R$, in which X_j is solution space. We need to find k solutions $\{x_1, x_2, \ldots, x_{k-1}, x_k\} = \arg\min\{f_1(x), f_2(x), \ldots, f_{k-1}(x), f_k(x)\}$, where x_j is a feasible solution in X_j. Each f_j is considered as an additional factor that impacts the evolution of a single population of individuals. Therefore, the problem also is called k-factorial problem. For a composite problem, general method for comparing individuals is necessary. Each individual p_i ($i \in \{1, 2, \ldots, |P|\}$) in a population P has a set of properties as follows: Factorial Cost, Factorial Rank, Scalar Fitness, and Skill Factor. These properties allow us to sort, and select individuals in the population.

- Factorial Cost c_j^i of the individual p_i is its fitness value for task T_j ($1 \leq j \leq k$).
- Factorial rank r_j^i of p_i on the task T_j is its index in the list of population individuals sorted in ascending order with respect to c_j^i.
- Scalar-fitness ϕ_i of p_i is given by its best factorial rank over all tasks as $\phi_i = \frac{1}{\min_{j \in 1,\ldots,k} r_j^i}$.
- Skill-factor ρ_i of p_i is the one task, amongst all other tasks, on which the individual is most effective, i.e., $\rho_i = \arg\min_j\{r_j^i\}$ where $j \in \{1, 2, \ldots, k\}$.

According to the scalar-fitness, we can compare population individuals in a multitasking environment. The basic structure of the MFEA (presented in Figure 2) includes main steps: population generation with skill-factor in unified search space, assortative crossover and mutation operator, skill-factor assignment, and elitist. We describe the pseudo-code of the basic MFEA in Algorithm 1 as follows: We initialize Sp (Sp is the size of population) individuals in the unified search space. They then are evaluated by calculating the factorial fitness and skill-factor of each individual. After the initialization, the iteration begins with selecting parents for the crossover and mutation operators. The output of these operators are offsprings. After that, the skill-factors selected randomly among those of the parents are assigned to those of the offsprings. Next, the offspring and the parent are merged to create a new population that has $2 \times Sp$ individuals, and finally the Elitist strategy keeps the Sp best solutions in terms of scalar-fitness for the next generation. The algorithm stops when the stop condition is satisfied.

Figure 3 shows the difference between the traditional GA and MFEA. In the MFEA, the crossover and mutation are also applied like the traditional GA. Unlike the traditional GA, the productions depend on two aspects: 1) the parents' skill factor and random mating probability (notation: *rmp*). More specifically, the offspring is created using crossover from parents that have the same skill factor. Otherwise, the offspring is produced by a crossover with the random mat-

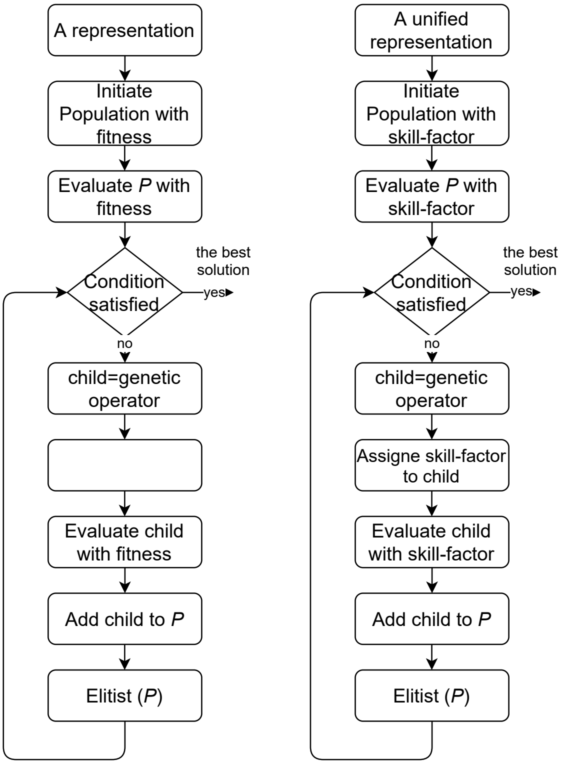

Figure 3. The similarity and difference between EA and MFEA

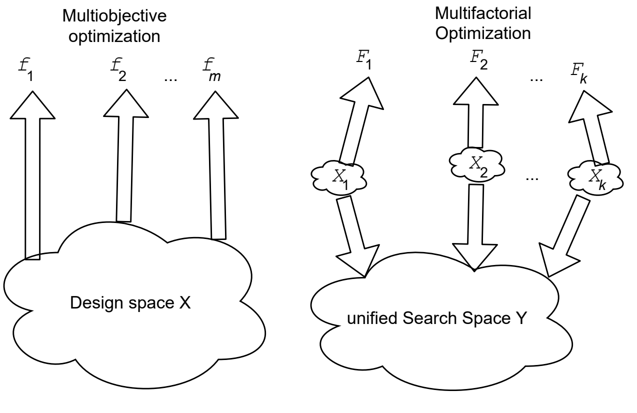

Figure 4. The difference between multiobjective and multifactorial optimization [15]

ing probability or mutation when parents have different skill factors. A large *rmp* creates more knowledge exchanging between two tasks. Also, unlike traditional GA, the offspring is not evaluated directly. The skill factor is assigned to the offspring.

Also, in Figure 4, the MFEA is different from multiobjective optimization. In multiobjective optimization, only one problem with many objective functions is solved, while the MFEA aims to optimize many tasks simultaneously. More clearly, While multiobjective optimization generally uses a single representative space for all objective functions, the MFEA unifies multiple representative spaces for many problems.

4 THE PROPOSED ALGORITHM

In this section, we introduce pseudocode of the proposed MFEA to solve two problems simultaneously. The TSP task is corresponding to a particular task in MFEA, while another is the TRP task. In the initial step, we created Sp individuals in the unified search space. Each individual is evaluated by calculating its factorial fitness and skill factor. After the initialization, the algorithm repeats these following actions until the stop condition is satisfied. The next step, selection operator picks parents to crossover or mutate. After the crossover and mutation operators, the offsprings are generated. The skill-factors among those of the parents are assigned to the obtained offsprings randomly. Next, the offspring and parent are merged to obtain a new population that has $2 \times Sp$ individuals, and the Elitist strategy keeps the Sp best solutions in terms of scalar-fitness for the next generation. In the population, we choose the individual with the best scalar-fitness to implement RVND procedure. The proposed algorithm is shown in Algorithm 1.

4.1 Creating Unified Search Space – USS

For two problems, many representations are proposed in the literature. These works indicate the efficiency of permutation representation in comparison with the others. In this paper, the permutation encoding is used, in which an individual is represented as a list of n vertices $(v_1, v_2, \ldots, v_k, \ldots, v_n)$, where v_1 is root and v_k is the k^{th} vertex to be visited. Figure 2 depicts this encoding for two tasks.

4.2 Initializing Population

Each tour created randomly takes a role as an individual in the population. Therefore, we have Sp individuals in the initial population for the genetic step.

Algorithm 1 MFEA-RVND

Input: K_n, C_{ij}, P, Sp, rmp are the graph, the cost matrix, the population, the size of population, and probability of crossover operator, respectively.
Output: The best solution T^*.

Create Unified Search Space – USS;
Init population;
while (The termination criterion of the GA is not satisfied) **do**
  {Genetic operators step}
  for ($j = 1; j \leqslant Sp; j++$) **do**
    (TP, TM) = Selection(P, NG); {select two parents TM, TP from the population}
    if TM and TP have the same skill-factor or (rand(1) $\leqslant rmp$) **then**
      TC_1, TC_2 = Crossover(TP, TM); {TC_1, TC_2 are the children}
      TC_1, TC_2's skill-factors are set to TP or TM randomly;
    else
      TC_1 = Mutate(TP);
      TC_1 = Mutate(TM);
      TC_1's skill-factor is set to TP;
      TC_2's skill-factor is set to TM;
    end if
    $P = P \cup \{TC_1, TC_2\}$; {Add TC_1, TC_2 to the current population}
  end for
  Update scalar-fitness for all individuals in P;
  P = Elitism-Selection(P); {Keep the best Sp individuals}
  T = Select the best individual from P;
  {RVND step}
  convert a solution from unified representation to representation for each task;
  RVND(T);
end while
return T^*;

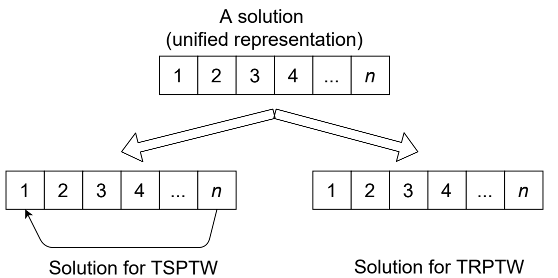

Figure 5. The interpretation of unified representation for each task

Algorithm 2 Crossover(TP, TM)

Input: TP, TM are the parent tours, respectively.
Output: A new child T.

```
type = rand(3);
rnd = rand(3);
if (type==1) then
  {the first type crossover is selected}
  if (rnd==1) then
    TC = PMX(TP, TM);{PMX is chosen}
  else if (rnd==2) then
    TC = CX(TP, TM);{CX is selected}
  else if (rnd==3) then
    TC = POS(TP, TM){POS is selected}
  end if
else if (type==2) then
  {the second type is selected}
  if (rnd==1) then
    TC = EXX(TP, TM);{EXX is selected}
  else if (rnd==2) then
    TC = EAX(TP, TM);{EAX is selected}
  else if (rnd==3) then
    TC = HGreX(TP, TM){HGreX is selected}
  end if
else if (type==3) then
  {type 3 is selected}
  if (rnd==1) then
    TC = SC(TP, TM);{SC is selected}
  else if (rnd==2) then
    TC = MC(TP, TM);{MC is selected}
  else if (rnd==3) then
    TC = ULX(TP, TM){ULX is selected}
  end if
end if
```

4.3 Evaluating for Individuals

The fitness function represents the method for the evaluation of individuals. We calculate skill-factor and scalar-fitness for each individual. The larger the scalar-fitness, the better the individual is.

Algorithm 3 **Mutate**(TC)

Input: TC is the child tour, respectively.
Output: A new child TC.

```
{Choose a mutation operator randomly}
rnd = rand(3);
if (rnd==1) then
    TC = EM(TC){EM is selected}
else if (rnd==2) then
    TC = IM(TC){IM is selected}
end if
```

Algorithm 4 **RVND**(T)

Input: T is a tour.
Output: A new solution T.

```
Initialize the Neighborhood List NL;
while NL ≠ 0 do
    Choose a neighborhood N in NL at random
    T' ← arg min N(T); {Neighborhood search}
    if ((W(T') < W(T*)) then
        T ← T'
        Update NL;
    else
        Remove N from the NL;
    end if
    if (W(T') < W(T*)) then
        T* ← T';
    end if
end while
```

4.4 Selection Operator

The selection phase is the process where the individuals are selected based on their scalar-fitness to mate and produce new offspring. There is no special selection method. In this work, the tournament selection operator is applied [24]. A group of NG individuals with a specified size is selected on a random basis. Then, two individuals that have the best scalar-fitness in the group will be chosen. These individuals will become parental individuals. Increased selection pressure can be provided by simply increasing the size of the group, as the winners from a larger size will, on average, have higher scalar-fitness than the winners of a small size.

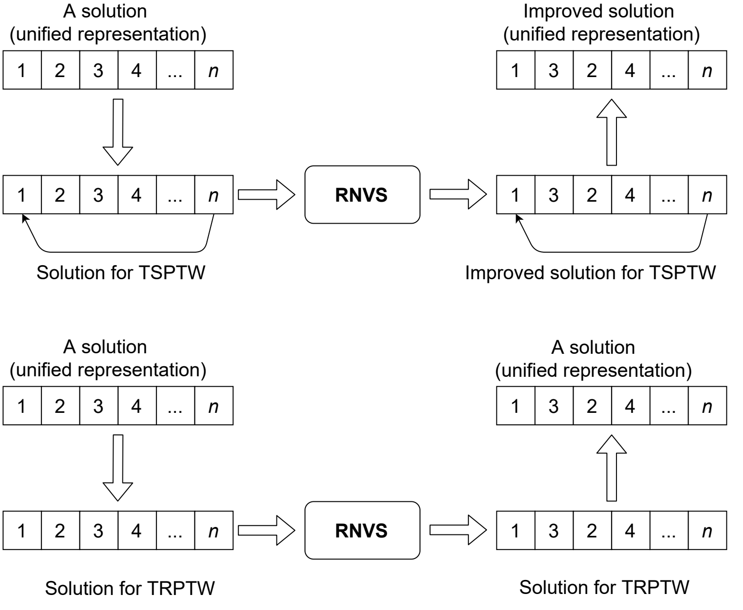

Figure 6. An illustration of converting unified representation to particular representation

4.5 Crossover Operator

The operator is done with the predefined crossover probability (*rmp*) or if parents have the same skill-factor. In [25], they divide the crossover operators into three types. We found no logical explanation of which one should bring a better performance or better overall results. In a pilot study, we found that the algorithm's performance is relatively insensitive to crossover operators, which are used. As testing our algorithm on all operators would have been computationally too expensive, we implement our numerical analysis on some selected operators for each type. In this work, the following operators are selected: type 1 (PMX, CX, and POS), type 2 (EXX, EAX, and HGreX), and type 3 (SC, MC, and ULX) [25]. Using multiple crossovers makes the population more diverse than using only one crossover. Therefore, it can help the algorithm to prevent being trapped in a local optimum. The offspring's skill-factor is set to the one of father or mother randomly. The detail of this step is given in Algorithm 2.

4.6 Mutation Operator

The operator is done if the above crossover does not occur. Mutation is an operator that serves as the means to keep the diversity of the population. It is one of the easiest operators to implement, so we have quite different mutation methods. Several mutations are used in our algorithm. Firstly, the simplest mutation

(EM) is the basic random exchange of two adjacent customers on tour. Secondly, another simple method is the insertion mutation (IM), which removes the customer from its place and inserts it in a random place on tour in another position. Specific time this mutation is performed, we select one of three operators randomly. After the mutation operator, two offsprings are created from the parents. Their skill-factors are set to those of parents. The detail of this step is given in Algorithm 3.

4.7 Elitism Operator

Let us start with the very popular elitism operator [24]. Elitism is a process that ensures the survival of the best intermediate solutions, so they are not lost through the evolutionary processes. Researchers define the number (usually very small, equal, or below 15 %) of the best solutions that automatically advance to the next generation. Our method chooses Sp individuals to the next generation, in which 15 % of them are the best solutions in the previous generation, and the remaining individuals are chosen randomly from P.

4.8 RVND

Every search algorithm needs to balance the exploration and exploitation of a search space. Exploration is the process of finding new promising space, whilst exploitation is the process of exploiting good regions of visited search space. Our GA is also combined with RVND to keep a balance between exploration and exploitation. The GA helps to explore the promising solution spaces while the RVND exploits these spaces. Before the RVND step, we convert a solution from unified representation to representation for each task. The RVND then applies to this solution. Finally, the output of the RVND is converted into a unified representation for the next step.

For the RVND step, we use several neighborhoods such as remove-insert, reinsertion, swap-adjacent, swap, 2-opt, and or-opt in [28]. A pseudocode of the RVND algorithm is given in Algorithm 4.

The stop condition: After the number of generations (Np), the best solution has not been improved, the GA stops.

5 COMPUTATIONAL EVALUATIONS

We have implemented the algorithm in C# language to evaluate its performance. The experiments are conducted on a personal computer equipped with a Xeon E-2234 CPU and 16 GB bytes RAM memory. Through preliminary experiments, we observed that the values $m = 10$, $Sp = 300$, $NG = 5$, $rmp = 0.7$, and $Np = 20$ resulted in a good trade-off between solution quality and run time. This parameter setting has thus been used in the following experiments.

The instances are gotten from the TSP and TRP benchmark in [22, 30]. We implemented our experiments on selected instances because testing on all instances would have been computationally too expensive. These are instances as follows:

1. The TSPLIB [30] includes many instances from 50 to 200 instances;
2. Three of these sets are generated by [22], where each of them is composed of 20 instances with 20, 30, 40, 50, and 100 customers, respectively.

Note that: TRP-30x and TRP-40x are generated randomly according to the method described in [22]. The aim of creating additional small instances is to obtain their optimal solutions from exact algorithm [4]. All tested instances from [2, 20, 22, 23, 29] are only Euclidean instances, and are available upon request.

The efficiency of the metaheuristic algorithm can be evaluated by comparing the best solution found by our algorithm (notation: *Best.Sol*) to

1. the optimal solution (notation: OPT); and
2. the previous metaheuristic solution (notation: UB) as follows:

$$gap_1[\%] = \frac{Best.Sol - OPT}{OPT} \times 100\,\%, \tag{1}$$

$$gap_2[\%] = \frac{Best.Sol - UB}{UB} \times 100\,\%. \tag{2}$$

In the experiments, our algorithm directly compares with the state-of-the-art metaheuristic algorithms [20, 29] on the same benchmark. These algorithms are developed from the MFEA framework. In Tables, OPT, *Aver.Sol* and *Best.Sol* are the optimal, average, and best solution after ten runs, respectively. Let $Time$ be the running time such that the proposed algorithm reaches the best solution. Moreover, the proposed algorithm also compares to the known-best solutions (or the optimal solutions) for the TSP [30, 31] and TRP [2, 23] in the literature. Yuan et al. support the source code of their algorithm in [29] while the dMFEA-II algorithm [20] was implemented by us. All algorithms are run on the same instances. Tables 1, 2, 3 and 4 compare between the MFEA + RVND and OA [20], and YA [29]. In the TSP, the optimal solutions of the TSPLIB-instances are extracted [30] while the optimal or best solutions are obtained by running Concord tool [31] for the other instances. In the TRP, the optimal or best solutions are obtained from [2, 23].

5.1 Comparison with the Previous MFEA Algorithms

Tables 1, 2, 3 and 4 compare the proposed algorithm to two algorithms [20, 29] for one hundred instances of both problems. The values in Table 5 are the average ones calculated from Tables 1, 2, 3 and 4. Tables 5, 6 and 7 show the results that the proposed algorithm reaches better solutions than the others in terms of the average gap in both problems. In addition, to more clearly illustrate the proposed algorithm's

Instances	OPT		YA				OA				MFEA							
	TSP	TRP	TSP		TRP		TSP		TRP		TSP				TRP			
			best. sol	*gap*	*best. sol*	*gap*	*best. sol*	gap_1	*best. sol*	gap_1	*best. sol*	*aver. sol*	gap_1	*Time Time*	*best. sol*	*aver. sol*	gap_1	*Time*
TRP-20-1	357	3 175	357	0	3 175	0	357	0	3 175	0	357	357	0	0.70	3 175	3 175	0	0.74
TRP-20-2	375	3 248	375	0	3 248	0	375	0	3 248	0	375	375	0	0.51	3 248	3 248	0	0.77
TRP-20-3	391	3 570	391	0	3 570	0	391	0	3 570	0	391	391	0	0.75	3 570	3 570	0	0.54
TRP-20-4	343	2 983	343	0	2 983	0	343	0	2 983	0	343	343	0	0.78	2 983	2 983	0	0.77
TRP-20-5	341	3 248	341	0	3 248	0	341	0	3 248	0	341	341	0	0.70	3 248	3 248	0	0.69
TRP-20-6	379	3 328	379	0	3 328	0	379	0	3 328	0	379	379	0	0.73	3 328	3 328	0	0.53
TRP-20-7	355	2 809	355	0	2 809	0	355	0	2 809	0	355	355	0	0.72	2 809	2 809	0	0.58
TRP-20-8	367	3 461	367	0	3 461	0	367	0	3 461	0	367	367	0	0.62	3 461	3 461	0	0.66
TRP-20-9	403	3 475	403	0	3 475	0	403	0	3 475	0	403	403	0	0.70	3 475	3 475	0	0.79
TRP-20-10	432	3 359	432	0	3 359	0	432	0	3 359	0	432	432	0	0.55	3 359	3 359	0	0.79
TRP-20-11	338	2 916	338	0	2 916	0	338	0	2 916	0	338	338	0	0.71	2 916	2 916	0	0.55
TRP-20-12	373	3 314	373	0	3 314	0	373	0	3 314	0	373	373	0	0.51	3 314	3 314	0	0.79
TRP-20-13	392	3 412	392	0	3 412	0	392	0	3 412	0	392	392	0	0.58	3 412	3 412	0	0.79
TRP-20-14	380	3 297	380	0	3 297	0	380	0	3 297	0	380	380	0	0.51	3 297	3 297	0	0.65
TRP-20-15	341	2 862	341	0	2 862	0	341	0	2 862	0	341	341	0	0.53	2 862	2 862	0	0.74
TRP-20-16	385	3 433	385	0	3 433	0	385	0	3 433	0	385	385	0	0.75	3 433	3 433	0	0.54
TRP-20-17	342	2 913	342	0	2 913	0	342	0	2 913	0	342	342	0	0.71	2 913	2 913	0	0.63
TRP-20-18	402	3 124	402	0	3 124	0	402	0	3 124	0	402	402	0	0.60	3 124	3 124	0	0.77
TRP-20-19	373	3 299	373	0	3 299	0	373	0	3 299	0	373	373	0	0.79	3 299	3 299	0	0.74
TRP-20-20	388	2 796	388	0	2 796	0	388	0	2 796	0	388	388	0	0.51	2 796	2 796	0	0.79
aver				0		0		0		0			0	0.65			0	0.69

Table 1. Comparison of all algorithms for TRP-20-x

Instances	OPT		YA				OA				MFEA							
	TSP	TRP	TSP		TRP		TSP		TRP		TSP				TRP			
			best. sol	gap_1	*best. sol*	gap_1	*best. sol*	gap_1	*best. sol*	gap_1	*best. sol*	*aver. sol*	gap_1	*Time*	*best. sol*	*aver. sol*	gap_1	*Time*
TRP-50-1	602	12 198	641	6.48	13 253	8.65	634	5.32	13 281	8.88	610	610	1.33	15.53	12 330	12 330	1.08	15.81
TRP-50-2	549	11 621	583	6.22	12 958	11.51	560	2.09	12 543	7.93	560	560	2.00	19.81	11 710	11 710	0.77	18.97
TRP-50-3	584	12 139	596	2.07	13 482	11.06	596	2.07	13 127	8.14	592	592	1.37	15.02	12 312	12 312	1.43	16.56
TRP-50-4	603	13 071	666	10.46	14 131	8.11	613	1.60	15 477	18.41	610	610	1.16	18.87	13 575	13 575	3.86	17.64
TRP-50-5	557	12 126	579	3.90	13 377	10.32	578	3.72	14 449	19.16	557	557	0.00	19.09	12 657	12 657	4.38	15.83
TRP-50-6	577	12 684	602	4.30	13 807	8.85	600	3.92	13 601	7.23	588	588	1.91	19.34	13 070	13 070	3.04	18.01
TRP-50-7	534	11 176	563	5.46	11 984	7.23	555	4.00	12 825	14.75	547	547	2.43	15.42	11 793	11 793	5.52	16.31
TRP-50-8	569	12 910	629	10.56	14 043	8.78	609	7.04	13 198	2.23	572	572	0.53	17.00	13 198	13 198	2.23	18.27
TRP-50-9	575	13 149	631	9.67	14 687	11.70	597	3.76	13 459	2.36	576	576	0.17	16.30	13 459	13 459	2.36	18.45
TRP-50-10	583	12 892	604	3.64	14 104	9.40	602	3.30	13 638	5.79	590	590	1.20	19.00	13 267	13 267	2.91	18.74
TRP-50-11	578	12 103	607	5.02	13 878	14.67	585	1.22	12 124	0.17	585	585	1.21	17.16	12 124	12 124	0.17	17.25
TRP-50-12	500	10 633	521	4.24	11 985	12.72	508	1.64	11 777	10.76	604	604	20.80	19.55	11 305	11 305	6.32	15.42
TRP-50-13	579	12 115	615	6.23	13 885	14.61	601	3.88	13 689	12.99	587	587	1.38	15.91	12 559	12 559	3.66	16.14
TRP-50-14	563	13 117	612	8.64	14 276	8.84	606	7.64	14 049	7.11	571	571	1.42	16.32	13 431	13 431	2.39	19.57
TRP-50-15	526	11 986	526	-0.08	12 546	4.67	526	-0.08	12 429	3.70	526	526	0.00	15.73	12 429	12 429	3.70	15.76
TRP-50-16	551	12 138	577	4.79	13 211	8.84	564	2.41	12 635	4.09	551	551	0.00	15.68	12 417	12 417	2.30	19.13
TRP-50-17	550	12 176	601	9.21	13 622	11.88	585	6.30	13 342	9.58	564	564	2.55	19.35	12 475	12 475	2.46	17.69
TRP-50-18	603	13 357	629	4.30	14 750	10.43	625	3.64	14 108	5.62	603	603	0.00	17.90	13 683	13 683	2.44	19.98
TRP-50-19	529	11 430	595	12.53	12 609	10.31	594	12.34	12 899	12.85	539	539	1.89	17.75	11 659	11 659	2.00	15.39
TRP-50-20	539	11 935	585	8.53	13 603	13.98	575	6.68	12 458	4.38	539	539	0.00	15.72	12 107	12 107	1.44	17.21
aver				6.31		10.33		4.12		8.31			2.05	17.32			2.72	17.41

Table 2. Comparison of all algorithms for TRP-50-x

Instances	UB		YA				OA				MFEA							
	TSP	TRP	TSP		TRP		TSP		TRP		TSP				TRP			
			best. sol	gap_2	*best. sol*	gap_2	*best. sol*	gap_2	*best. sol*	gap_2	*best. sol*	*aver. sol*	gap_2	*Time*	*best. sol*	*aver. sol*	gap_2	*Time*
TRP-100-1	762	32 779	830	8.92	36 012	9.86	806	5.82	36 869	12.48	791	791	3.81	130.55	35 785	35 785	9.17	142.54
TRP-100-2	771	33 435	800	3.80	39 019	16.70	817	5.97	37 297	11.55	782	782	1.43	144.92	35 546	35 546	6.31	127.65
TRP-100-3	746	32 390	865	15.95	38 998	20.40	849	13.84	34 324	5.97	767	767	2.82	137.56	34 324	34 324	5.97	135.18
TRP-100-4	776	34 733	929	19.72	41 705	20.07	897	15.59	38 733	11.52	810	810	4.38	136.49	37 348	37 348	7.53	140.97
TRP-100-5	749	32 598	793	5.89	40 063	22.90	899	20.03	37 191	14.09	774	774	3.34	147.52	34 957	34 957	7.24	146.73
TRP-100-6	807	34 159	905	12.14	40 249	17.83	886	9.82	40 588	18.82	854	854	5.82	128.58	36 689	36 689	7.41	148.78
TRP-100-7	767	33 375	780	1.69	38 794	16.24	849	10.69	39 430	18.14	780	780	1.69	142.72	35 330	35 330	5.86	136.42
TRP-100-8	744	31 780	824	10.75	38 155	20.06	845	13.58	35 581	11.96	763	763	2.55	142.61	34 342	34 342	8.06	124.16
TRP-100-9	786	34 167	863	9.80	39 189	14.70	858	9.16	41 103	20.30	809	809	2.93	131.41	35 990	35 990	5.34	124.48
TRP-100-10	751	31 605	878	16.91	36 191	14.51	831	10.65	37 958	20.10	788	788	4.93	137.03	33 737	33 737	6.75	127.73
TRP-100-11	776	34 188	831	7.09	39 750	16.27	876	12.89	41 153	20.37	814	814	4.90	122.28	36 988	36 988	8.19	145.22
TRP-100-12	797	32 146	855	7.29	39 422	22.63	855	7.29	40 081	24.68	823	823	3.26	121.62	34 103	34 103	6.09	127.63
TRP-100-13	753	32 604	772	2.46	37 004	13.50	772	2.46	40 172	23.21	771	771	2.39	135.92	35 011	35 011	7.38	144.43
TRP-100-14	770	32 433	810	5.19	40 432	24.66	810	5.19	36 134	11.41	800	800	3.90	143.38	34 576	34 576	6.61	127.31
TRP-100-15	776	32 574	953	22.81	38 369	17.79	878	13.17	38 450	18.04	810	810	4.38	148.02	35 653	35 653	9.45	147.88
TRP-100-16	775	33 566	838	8.10	40 759	21.43	835	7.74	38 549	14.85	808	808	4.26	123.90	36 188	36 188	7.81	130.50
TRP-100-17	805	34 198	939	16.65	39 582	15.74	881	9.44	42 155	23.27	838	838	4.10	137.06	36 969	36 969	8.10	125.90
TRP-100-18	785	31 929	876	11.59	38 906	21.85	836	6.50	37 856	18.56	814	814	3.69	134.08	34 154	34 154	6.97	127.53
TRP-100-19	780	33 463	899	15.26	39 865	19.13	881	12.95	40 379	20.67	797	797	2.18	120.36	35 669	35 669	6.59	138.48
TRP-100-20	775	33 632	816	5.30	41 133	22.30	905	16.77	40 619	20.77	808	808	4.26	130.11	35 532	35 532	5.65	134.20
aver				10.37		18.43		10.48		17.04			3.55	134.81			7.12	135.18

Table 3. Comparison of all algorithms for TRP-100-x

Instances	*UB*		YA				OA				MFEA							
	TSP	TRP	TSP		TRP		TSP		TRP		TSP				TRP			
			best.sol	$gap_{1(2)}$	*best.sol*	$gap_{1(2)}$	*best. sol*	$gap_{1(2)}$	*best. sol*	$gap_{1(2)}$	*best. sol*	*aver. sol*	$gap_{1(2)}$	*Time*	*best. sol*	*aver. sol*	$gap_{1(2)}$	*Time*
eil51	426*	10 178*	446	4.69	10 834	6.45	450	5.63	10 834	6.45	426	426	0.00	16.87	10 178	10 178	0.00	15.64
berlin52	7 542*	143 721*	7 922	5.04	152 886	6.38	8 276	9.73	152 886	6.38	7 542	7 542	0.00	16.45	143 721	143 721	0.00	14.56
st70	675*	20 557*	713	5.63	22 283	8.40	772	14.37	22 799	10.91	680	680	0.74	21.20	22 283	22 283	8.40	22.32
eil76	538*	17 976*	560	4.09	18 777	4.46	589	9.48	18 008	0.18	559	559	3.90	30.12	18 008	18 008	0.18	29.54
pr76	108 159*	3 455 242*	113 017	4.49	3 493 048	1.09	117 287	8.44	3 493 048	1.09	108 159	108 159	0.00	28.95	3 455 242	3 455 242	0.00	28.98
pr107	44 303*	2 026 626*	45 737	3.24	2 135 492	5.37	46 338	4.59	2 135 492	5.37	45 187	45 187	2.00	151.20	2 052 224	2 052 224	1.26	159.32
pr124	59 030*	3 284 743*	62 460	5.81	3 544 105	7.90	63 673	7.87	3 544 105	7.90	60 863	61 294.8	3.11	156.30	3 304 197	3 417 630.25	0.59	170.32
pr136	96 772*	6 618 288*	102 177	5.59	7 028 709	6.20	102 177	5.59	7 028 709	6.20	96 772	96 772	0.00	185.65	6 618 288	6 618 288	0.00	180.56
rat99	1 211*	58 288*	1 316	8.67	60 134	3.17	1 369	13.05	60 134	3.17	1 280	1 280	5.70	148.65	58 971	59 552.5	1.17	140.65
kroA100	21 282*	983 128*	22 233	4.47	1 043 868	6.18	22 233	4.47	1 043 868	6.18	21 878	21 878	2.80	218.78	1 009 986	1 033 437.75	2.73	198.56
kroB100	22 141*	986 008*	23 144	4.53	1 118 869	13.47	24 337	9.92	1 118 869	13.47	23 039	23 039	4.06	134.6 776	1 003 107	1 044 585.25	1.73	140.32
kroC100	20 749*	961 324*	22 395	7.93	1 026 908	6.82	23 251	12.06	1 026 908	6.82	21 541	21 541	3.82	130.1 316	1 007 154	1 018 906.5	4.77	135.62
kroD100	21 294*	976 965*	22 467	5.51	1 069 309	9.45	23 833	11.92	1 069 309	9.45	22 430	22 430	5.33	147.0 016	1 019 821	1 044 565	4.39	150.32
kroE100	22 068*	971 266*	22 960	4.04	1 056 228	8.75	23 622	7.04	1 056 228	8.75	22 964	22 964	4.06	131.0 774	1 034 760	1 044 693.75	6.54	141.96
rd100	7 910*	340 047*	8 381	5.95	380 310	11.84	8 778	10.97	365 805	7.57	8 333	8 333	5.35	123.3 361	354 762	369 679.2 857	4.33	132.12
eil101	629*	27 519*	681	8.27	28 398	3.19	695	10.49	28 398	3.19	662	662	5.25	134.7 259	27 741	27 741	0.81	128.45
ch130	6 110*	359 952*	6 512	6.58	378 043	5.03	6 601	8.04	378 043	5.03	6 334	6 334	3.67	186.32	364 355	373 350.6 667	1.22	172.32
ts225	126 643	13 577 376	130 144	2.76	14 241 621	4.89	137 698	8.73	14 241 621	4.89	129 680	129 680	2.40	250.12	13 746 453	13 929 066.8	1.25	255.63
tsp225	3 919	401 012	4 204	7.27	451 445	12.58	4 316	10.13	451 445	12.58	4 270	4 294	8.96	241.21	448 108	449 776.5	11.74	243.21
aver				5.50		6.93		9.08		6.61			3.22	129.09			2.69	129.49

Table 4. Comparison of all algorithms for TSPLIB

Instances	YA		OA		MFEA			
	TSP	TRP	TSP	TRP	TSP		TRP	
	$gap_{1(2)}$	$gap_{1(2)}$	$gap_{1(2)}$	$gap_{1(2)}$	$gap_{1(2)}$	$Time$	$gap_{1(2)}$	$Time$
TRP-20-x	0	0	0	0	**0**	0.65	**0**	0.69
TRP-50-x	6.31	10.33	4.12	8.31	**2.05**	17.32	**2.72**	17.41
TRP-100-x	10.37	18.43	10.48	17.04	**3.22**	134.8	**7.12**	135.18
TSPLIB	5.5	6.93	9.08	6.61	**3.69**	129.0	**2.69**	129.49

The better average values are highlighted in boldface

Table 5. Average results for all datasets

Instances	R+	R-	p-value
TSP	1 986	−157	0.0001
TRP	1 963	−27	0.0001

Table 6. Wilcoxon signed ranks test results between the proposed MFEA and the others with a level of significance $\alpha = 0.05$

efficiency, a Wilcoxon Rank-Sum test has been applied to verify the statistical results. Because we cannot make an assumption about probability distribution of the outcomes, the non-parametric test like Wilcoxon Rank-Sum Test is a suitable choice. We have compared the outcomes obtained for each instance and problem separately for properly performing this Wilcoxon Rank-Sum test, establishing the confidence interval at 95 %. The results of Wilcoxon Rank-Sum test is shown in Table 6. In two problems, the results in Table 6 indicate that the MFEA + RVND shows a significant improvement over the state-of-art MFEA algorithms with a level of significance $\alpha = 0.05$.

Instances	Optimal TSP Using TRP Objective Function	Optimal TRP	diff [%]	Optimal TRP Using TSP Objective Function	Optimal TSP	diff [%]
st70	22 865	20 557	10	847	675	20
eil51	10 963	10 178	7	482	426	12
berlin52	207 280	143 721	31	8 961	7 542	16
eil101	30 849	27 519	11	751	629	16
pr76	4 019 567	3 455 242	14	131 473	108 159	18
KroA100	1 087 955	983 128	10	23 249	21 282	8
rat99	60 415	56 994	6	1 391	1 211	13
lin105	904 993	585 823	54	17 159	14 379	16
aver			18			15

Table 7. The difference between the optimal TSP using TRP objective function and vice versa

5.2 Comparison with the Previous TSP and TRP's Solutions

In Table 5, the average *gap* for the TSP and TRP is below 3 %. It shows that our solutions are close to the optimal solutions for both problems. The improvement is significant since it can be observed that our algorithm is capable of finding good solutions fast for two problems at the same time. In comparison with the state-of-the-art solutions for the TSP and TRP, our solutions reach the optimal solutions in 65 out of 119 cases for the TSP and 64 out of 119 cases for the TRP in a short computation time.

It is unrealistic to expect that the proposed MFEA gives better solutions than those of state-of-the-art metaheuristic algorithms for the TSP or TRP in the literature because their algorithms are designed to independently solve each problem. In Table 7, the efficient algorithm for the TSP may not be good for the TRP on the same instances and vice versa. On average, the optimal solution for the TSP using the TRP objective function is 18 % worse than the optimal solution for the TRP. Similarly, the optimal solution for the TRP using the TSP objective function is 15 % worse than the optimal solution for the TSP. We conclude that if the proposed MFEA simultaneously reaches the good solutions that are near to the optimal solutions for both problems (on average, our solutions are below 3 % for the TSP and TRP in comparison with the optimal solutions), we can still say that the proposed MFEA + RVND for multitasking is beneficial.

5.3 Convergence Trend

The normalized objective function is used for analyzing the proposed MFEA algorithm's convergence trends. It calculated as follows:

$$\overline{f_j} = \frac{(f_j - f_j^{min})}{(f_j^{max} - f_j^{min})}$$

where $j = 1, 2$ and f_j^{min}, f_j^{max} are the minimum and maximum function cost values across all test runs.

This section analyzes the proposed MFEA algorithm's convergence trends from two aspects:

1. single-tasking strategy;
2. multitasking strategy.

In the single-task, we run the MEFA for each task independently. Otherwise, two tasks are run simultaneously. We select two instances eil51, and TRP-50-R1 in this experiment. Furthermore, to evaluate the diversification contribution of the MFEA in the proposed algorithm, in this experiment, we run the algorithm without the RVND.

The convergence trend of the single-tasking and multitasking is described in Figures 7, 8, 9 and 10 for eil51 and TRP-50-R1. In these figures, single-tasking (ST)

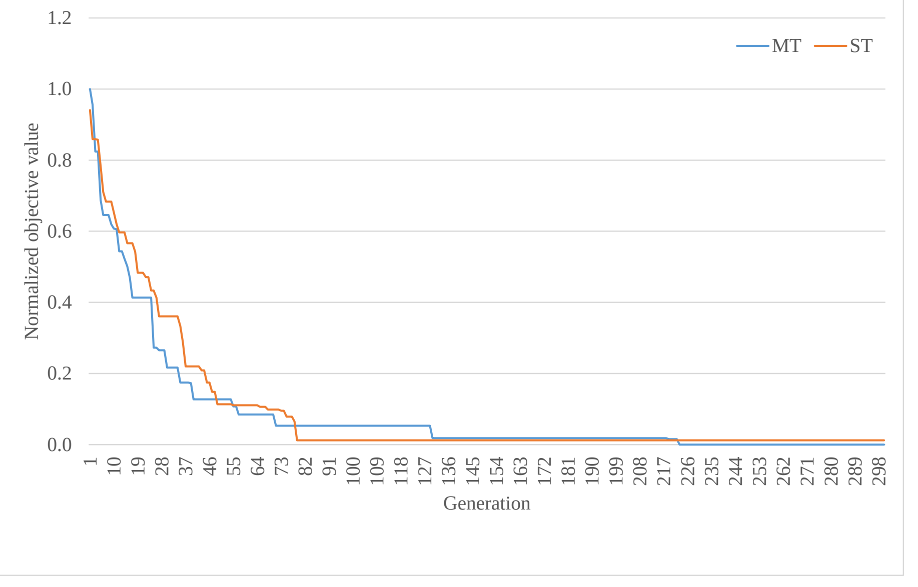

Figure 7. Comparing convergence trends of f_1 and f_2 in multi-tasking and single-tasking for the TSP in eil51 instance

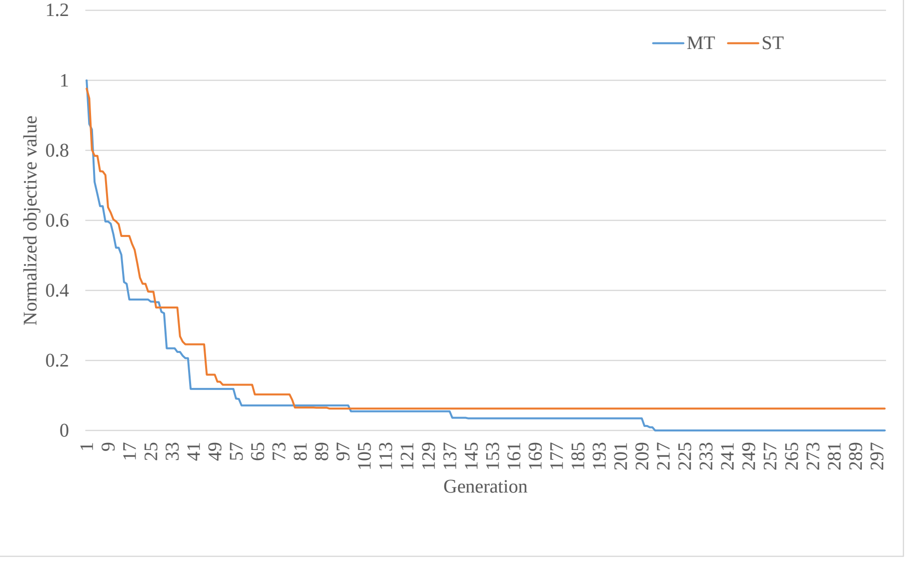

Figure 8. Comparing convergence trends of f_1 and f_2 in multi-tasking and single-tasking for the TRP in eil51 instance

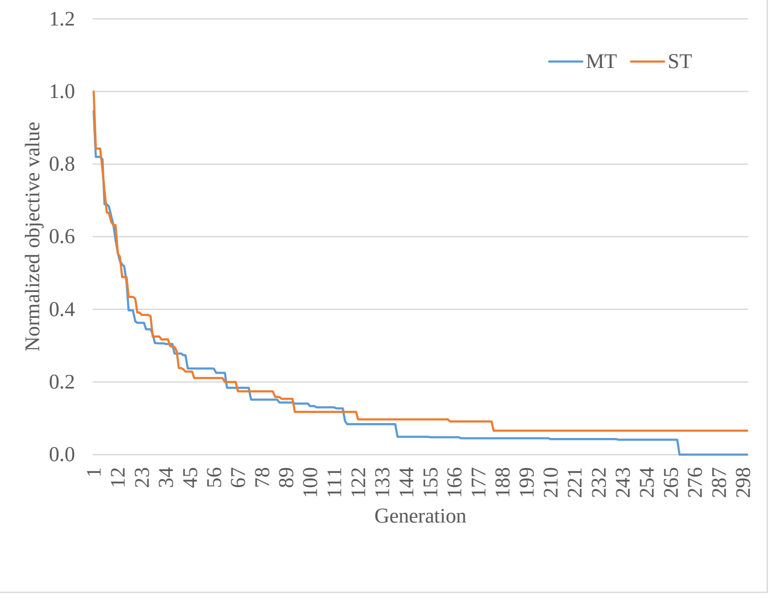

Figure 9. Comparing convergence trends of f_1 and f_2 in multi-tasking and single-tasking for the TSP in TRP-50-1 instance

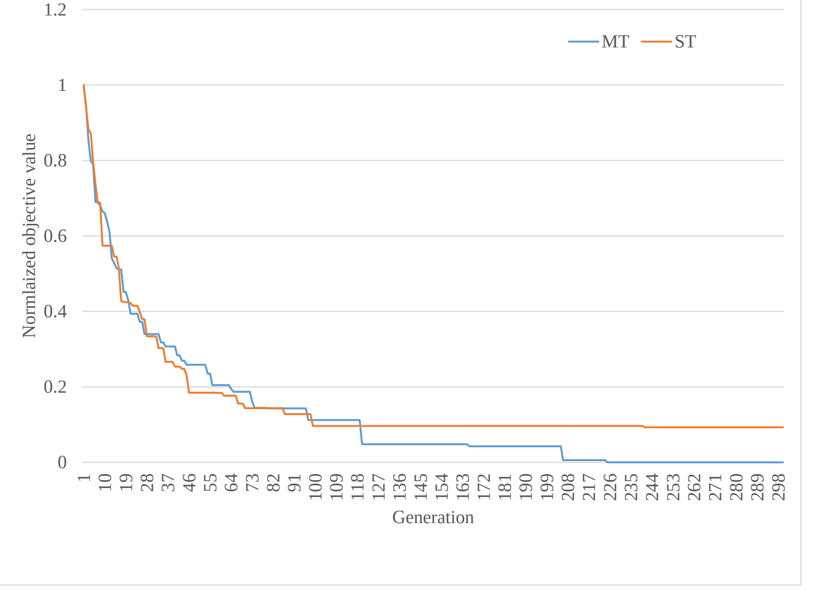

Figure 10. Comparing convergence trends of f_1 and f_2 in multi-tasking and single-tasking for the TRP in TRP-50-1 instance

converges faster than multitasking (MT). Therefore, the multitasking can generally converge to a better objective value.

In summary, the performance of the multitasking strategy is better than the one of single-tasking. With the same instances and algorithm, the improvement can be obtained by exploiting multiple function landscapes via implicit genetic transfer. Obviously, the process of transferring knowledge during multitasking leads to the clear dominance of multitasking in comparison with single-tasking. It shows the advantage of the evolutionary multitasking paradigm.

The proposed algorithm's average running time does not consume in tables than the others in [2, 30]. Moreover, it grows quite moderate with the single-tasking because it runs two problems at the same time.

6 DISCUSSIONS

The MFEA framework [6, 11, 14, 26, 29] has been developed to incorporate Evolutionary Algorithms into multitasking to handle multiple problems at the same time. Instead of solving a pool of similar optimization problems singly, it handles various requests and performs multiple tasks for systems. The advantage of the approach is that the phenomenon of implicit genetic transfer in multitasking can exploit transferrable knowledge between optimization tasks. The approach is good for the problems that have the same representative solution space.

The TSP [3, 9, 10, 13, 16, 18, 19], and TRP [2, 4, 5, 7, 8, 17, 22, 23] are combinatorial optimization problems that have many practical situations. Currently, there exist many algorithms that are proposed to solve them. However, these algorithms are designed to solve each problem independently. That means each problem is solved separately. This paper introduces the first algorithm that combines the MFEA framework and RVND for solving two problems simultaneously. The reason behind the combination is to maintain the right balance between diversification from the MFEA and intensification from RVND. In the literature, the idea of the combination between the MFEA and local search (such as 2-opt) is proposed in [29]. However, 2-opt may not exploit well promising solution space. Conversely, the RVND is a powerful framework that uses many neighborhood search heuristics. It often exploits promising solution space better than 2-opt. In comparison with the previous schemes, our scheme includes new features as follows:

- Multiple crossover and mutation operator schemes are used in our MFEA. These schemes help our algorithm to maintain good diversity.
- The combination between the MFEA with the RVND to keep a right balance between exploration and exploitation. We use more neighborhoods; therefore, the explored neighborhood is extended, and the chance to obtain a better solution is higher.

Summarily, this work's main contributions can be summarized as follows:

1. From the algorithmic perspective, the proposed algorithm brings the advantages of the MFEA with multiple crossover and mutation operators and RVND. The hybrid consists of new features compared with the previous schemes;
2. From the computational perspective, extensive numerical experiments on benchmark instances show that our algorithm solves two problems well simultaneously.

Moreover, it reaches better solutions than the previous MFEA framework in many cases.

7 CONCLUSIONS

In the paper, we present an effective MFEA framework for solving the TSP and TRP simultaneously, which combines the MFEA, and RVND. In the proposed algorithm, the MFEA is used to explore the promising solution areas while the RVND exploits them. Thus, the combination maintains the balance between exploration and exploitation. Extensive computational experiments on benchmark instances show that our solutions reach the optimal solutions in 65 out of 119 cases for the TSP and 64 out of 119 cases for the TRP in a short computation time. Furthermore, it shows that the proposed algorithm can solve well two problems at the same time. In comparison with the state-of-the-art MFEA for solving TSP and TRP, our algorithm finds either the better solutions in many cases or at least as well as for the others. However, the running time does not meet real situations. Therefore, enhancing it is our aim in the future.

REFERENCES

[1] Arellano-Arriaga, N. A.—Molina, J.—Schaeffer, S. E.—Álvarez-Socarrás, A. M.—Martínez-Salazar, I. A.: A Bi-Objective Study of the Minimum Latency Problem. Journal of Heuristics, Vol. 25, 2019, No. 1, pp. 431–454, doi: 10.1007/s10732-019-09405-0.

[2] Abeledo, H.—Fukasawa, R.—Pessoa, A.—Uchoa, E.: The Time Dependent Traveling Salesman Problem: Polyhedra and Algorithm. Mathematical Programming Computation, Vol. 5, 2013, No. 1, pp. 27–55, doi: 10.1007/s12532-012-0047-y.

[3] Applegate, D. L.—Bixby, R. E.—Chvatal, V.—Cook, W. J.: The Traveling Salesman Problem: A Computational Study. Princeton University Press, Princeton, 2007.

[4] Ban, H. B.—Nguyen, K.—Ngo, M. C.—Nguyen, D. N.: An Efficient Exact Algorithm for the Minimum Latency Problem. Progress in Informatics, Vol. 10, 2013, pp. 167–174, doi: 10.2201/NiiPi.2013.10.10.

[5] Ban, H. B.: An Efficient Two-Phase Metaheuristic Algorithm for The Time Dependent Traveling Salesman Problem. RAIRO-Operations Research, Vol. 53, 2019, No. 3, pp. 917–935, doi: 10.1051/ro/2019006.

[6] Bali, K. K.—Ong, Y. S.—Gupta, A.—Tan, P. S.: Multifactorial Evolutionary Algorithm with Online Transfer Parameter Estimation: MFEA-II. IEEE Transactions on Evolutionary Computation, Vol. 24, 2020, No. 1, pp. 69–83, doi: 10.1109/TEVC.2019.2906927.

[7] Blum, A.—Chalasani, P.—Coppersmith, D.—Pulleyblank, W.—Raghavan, P.—Sudan, M.: The Minimum Latency Problem. Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing (STOC '94), 1994, pp. 163–171, doi: 10.1145/195058.195125.

[8] Chaudhuri, K.—Godfrey, B.—Rao, S.—Talwar, K.: Paths, Trees, and Minimum Latency Tours. Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, 2003, pp. 36–45, doi: 10.1109/SFCS.2003.1238179.

[9] Chvátal, V.—Cook, W.—Dantzig, G. B.—Fulkerson, D. R.—Johnson, S. M.: Solution of a Large-Scale Traveling-Salesman Problem. In: Jünger, M., Liebling, T. M., Naddef, D., Nemhauser, G. L., Pulleyblank, W. R., Reinelt, G., Rinaldi, G., Wolsey, L. A. (Eds.): 50 Years of Integer Programming 1958–2008: From the Early Years to the State-of-the-Art. Chapter 1. Springer, Berlin, Heidelberg, pp. 7–28, doi: 10.1007/978-3-540-68279-0_1.

[10] Dorigo, M.—Gambardella, L. M.: Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. IEEE Transactions on Evolutionary Computation, Vol. 1, 1997, No. 1, pp. 53–66, doi: 10.1109/4235.585892.

[11] Feng, L.—Zhou, W.—Zhou, L.—Jiang, S.—Zhong, J.—Da, B.—Zhu, Z.—Wang, Y.: An Empirical Study of Multifactorial PSO and Multifactorial DE. 2017 IEEE Congress on Evolutionary Computation (CEC), 2017, pp. 921–928, doi: 10.1109/CEC.2017.7969407.

[12] Fischetti, M.—Laporte, G.—Martello, S.: The Delivery Man Problem and Cumulative Matroids. Operations Research, Vol. 41, 1993, No. 6, pp. 1055–1064, doi: 10.1287/opre.41.6.1055.

[13] Gutin, G.—Punnen, A. P. (Eds.): The Traveling Salesman Problem and Its Variations. Springer, New York, Combinatorial Optimization, Vol. 12, 2006, doi: 10.1007/b101971.

[14] Gupta, A.—Ong, Y. S.—Feng, L.: Multifactorial Evolution: Toward Evolutionary Multitasking. IEEE Transactions on Evolutionary Computation, Vol. 20, 2016, No. 3, pp. 343–357, doi: 10.1109/TEVC.2015.2458037.

[15] Gupta, A.—Mańdziuk, J.—Ong, Y. S.: Evolutionary Multitasking in Bi-Level Optimization. Complex and Intelligent Systems, Vol. 1, 2015, No. 1–4, pp. 83–95, doi: 10.1007/s40747-016-0011-y.

[16] Laporte, G.: The Traveling Salesman Problem: An Overview of Exact and Approximate Algorithms. European Journal of Operational Research, Vol. 59, 1992, No. 2, pp. 231–247, doi: 10.1016/0377-2217(92)90138-Y.

[17] Lucena, A.: Time-Dependent Traveling Salesman Problem – The Deliveryman Case. Networks, Vol. 20, 1990, No. 6, pp. 753–763, doi: 10.1002/net.3230200605.

[18] Lin, S.—Kernighan, B. W.: An Effective Heuristic Algorithm for the Traveling-Salesman Problem. Operations Research, Vol. 21, 1973, No. 2, pp. 498–516, doi: 10.1287/opre.21.2.498.

[19] Miller, C. E.—Tucker, A. W.—Zemlin, R. A.: Integer Programming Formulation of Traveling Salesman Problems. Journal of the ACM, Vol. 7, 1960, No. 4, pp. 326–329, doi: 10.1145/321043.321046.

[20] Osaba, E.—Martinez, A. D.—Galvez, A.—Iglesias, A.—Del Ser, J.: dMFEA-II: An Adaptive Multifactorial Evolutionary Algorithm for Permutation-Based Discrete Optimization Problems. Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion (GECCO '20), 2020, pp. 1690–1696, doi: 10.1145/3377929.3398084.

[21] Orman, A. J.—Williams, H. J.: A Survey of Different Integer Programming Formulations of the Travelling Salesman Problem. In: Kontoghiorghes, E. J., Gatu, C. (Eds.): Optimisation, Econometric and Financial Analysis. Springer, Berlin, Heidelberg, Advances in Computational Management Science, Vol. 9, 2007, pp. 91–104, doi: 10.1007/3-540-36626-1_5.

[22] Salehipour, A.—Sorensen, K.—Goos, P.—Braysy, O.: Efficient GRASP + VND and GRASP+VNS Metaheuristics for the Traveling Repairman Problem. 4OR – A Quarterly Journal of Operations Research, Vol. 9, 2011, No. 2, pp. 189–209, doi: 10.1007/s10288-011-0153-0.

[23] Silva, M. M.—Subramanian, A.—Vidal, T.—Ochi, L. S.: A Simple and Effective Metaheuristic for the Minimum Latency Problem. European Journal of Operational Research, Vol. 221, 2012, No. 3, pp. 513–520, doi: 10.1016/j.ejor.2012.03.044.

[24] Razali, N. M.—Geraghty, J.: Genetic Algorithm Performance with Different Selection Strategies in Solving TSP. International Conference of Computational Intelligence and Intelligent Systems (ICCIIS '11), 2011, pp. 1134–1139.

[25] Abdoun, O.—Abouchabaka, J.: A Comparative Study of Adaptive Crossover Operators for Genetic Algorithms to Resolve the Traveling Salesman Problem. International Journal of Computer Applications, Vol. 31, 2011, No. 11, pp. 49–57.

[26] Xie, T.—Gong, M.—Tang, Z.—Lei, Y.—Liu, J.—Wang, Z.: Enhancing Evolutionary Multifactorial Optimization Based on Particle Swarm Optimization. 2016 IEEE Congress on Evolutionary Computation (CEC), 2016, pp. 1658–1665, doi: 10.1109/CEC.2016.7743987.

[27] Xu, Q.—Wang, N.—Wang, L.—Li, W.—Sun, Q.: Multi-Task Optimization and Multi-Task Evolutionary Computation in the Past Five Years: A Brief Review. Mathematics, Vol. 9, 2021, No. 8, Art. No. 864, doi: 10.3390/math9080864.

[28] Vander Wiel, R. J.—Sahinidis, N. V.: Heuristics Bounds and Test Problem Generation for the Time-Dependent Traveling Salesman Problem. Transportation Science, Vol. 29, 1995, No. 2, pp. 167–183, doi: 10.1287/trsc.29.2.167.

[29] Yuan, Y.—Ong, Y. S.—Gupta, A.—Tan, P. S.—Xu, H.: Evolutionary Multitasking in Permutation-Based Combinatorial Optimization Problems: Realization with TSP, QAP, LOP, and JSP. Proceedings of the 2016 IEEE Region 10 Conference (TENCON), 2016, pp. 3157–3164, doi: 10.1109/TENCON.2016.7848632.

[30] `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`.

[31] `http://www.math.uwaterloo.ca/tsp/concorde.html`.

Ha-Bang Ban received his Bc.E. in information technology and his Ph.D. in computer science from the Hanoi University of Science and Technology (HUST), Vietnam, in 2006 and 2015, respectively. He is currently Lecturer at the School of Information and Communication Technology (SoICT), HUST, Vietnam. His research interests include algorithms, graphs, optimization, logistics, etc. He has published many publications in peer-reviewed international journals and conferences.

Hai-Dang Pham received the engineering diploma in information technology from the Hanoi University of Science and Technology (HUST), Vietnam, in 1995 and his Ph.D. in computer science from Ecole Pratique des Hautes Etudes (EPHE), France, in 2011. He is currently Senior Lecturer at the School of Information and Communication Technology (SoICT), HUST, Vietnam. His current research interests include algorithms, parallel and distributed simulation, multi-agent based simulation and high performance computing.