Solving optimization problems simultaneously: the variants of the traveling salesman problem with time windows using multifactorial evolutionary algorithm

Ha-Bang Ban Corresp., 1 , **Dang-Hai Pham** Corresp. 1

[1] Computer Science, Hanoi University of Science and Technology, Hanoi, Vietnam

Corresponding Authors: Ha-Bang Ban, Dang-Hai Pham
Email address: BANGBH@SOICT.HUST.EDU.VN, HaiPD@soict.hust.edu.vn

We study two problems called the Traveling Repairman Problem (TRPTW) and Traveling Salesman Problem (TSPTW) with time windows. The TRPTW wants to minimize the sum of travel durations between a depot and customer locations, while the TSPTW aims to minimize the total time to visit all customers. In two problems, the deliveries are made during a specific time window given by the customers. The difference between the TRPTW and TSPTW is that the TRPTW takes a customer-oriented view, whereas the TSPTW is server-oriented. Existing algorithms have been developed for solving two problems independently in the literature. However, the literature does not have an algorithm that simultaneously solves two problems. Nowadays, Multifactorial Evolutionary Algorithm (MFEA) is a variant of the Evolutionary Algorithm (EA), aiming to solve multiple factorial tasks simultaneously. The main advantage of the approach is to allow transferrable knowledge between tasks. Therefore, it can improve the solution quality for multitasks. This paper presents an efficient algorithm that combines the MFEA framework and Randomized Variable Neighborhood Search (RVNS) to solve two problems simultaneously. The proposed algorithm has transferrable knowledge between tasks from the MFEA and the ability to exploit good solution space from RVNS. The proposed algorithm is compared directly to the state-of-the-art MFEA on numerous datasets. Experimental results show the proposed algorithm outperforms the state-of-the-art MFEA in many cases. In addition, it finds several new best-known solutions.

Solving Optimization Problems Simultaneously: The variants of the Traveling Salesman Problem with Time Windows using Multifactorial Evolutionary Algorithm

Ha-Bang Ban, Dang-Hai Pham

School of Information and Communication Technology, Hanoi University of Science and Technology, Hanoi, Vietnam

Corresponding author:
Ha-Bang Ban, and Dang-Hai Pham

Email address: BangBH@soict.hust.edu.vn; HaiPD@soict.hust.edu.vn

ABSTRACT

We study two problems called the Traveling Repairman Problem (TRPTW) and Traveling Salesman Problem (TSPTW) with time windows. The TRPTW wants to minimize the sum of travel durations between a depot and customer locations, while the TSPTW aims to minimize the total time to visit all customers. In two problems, the deliveries are made during a specific time window given by the customers. The difference between the TRPTW and TSPTW is that the TRPTW takes a customer-oriented view, whereas the TSPTW is server-oriented. Existing algorithms have been developed for solving two problems independently in the literature. However, the literature does not have an algorithm that simultaneously solves two problems. Nowadays, Multifactorial Evolutionary Algorithm (MFEA) is a variant of the Evolutionary Algorithm (EA), aiming to solve multiple factorial tasks simultaneously. The main advantage of the approach is to allow transferrable knowledge between tasks. Therefore, it can improve the solution quality for multitasks. This paper presents an efficient algorithm that combines the MFEA framework and Randomized Variable Neighborhood Search (RVNS) to solve two problems simultaneously. The proposed algorithm has transferrable knowledge between tasks from the MFEA and the ability to exploit good solution space from RVNS. The proposed algorithm is compared directly to the state-of-the-art MFEA on numerous datasets. Experimental results show the proposed algorithm outperforms the state-of-the-art MFEA in many cases. In addition, it finds several new best-known solutions.

1 INTRODUCTION

1.1 The TSPTW and TRPTW literature

The TSPTW (S. Dash et al.,2012; M. Gendreau et al.,1998; A. Langevin et al.,1993; R. F. Silva et al.,2010; J. N. Tsitsiklis et al.,1992; J. W. Ohlmann et al.,2007), and TRPTW (H. Abeledo et al.,2013; H.B. Ban et al.,2017; H.B. Ban et al.,2021; G. Heilporna et al.,2010; J. N. Tsitsiklis et al.,1992) are combinatorial optimization problems that have many practical situations. The TRPTW wants to minimize the sum of travel durations between a depot and customer locations, while the TSPTW aims to minimize the total time to visit all customers. In two problems, the deliveries are made during a specific time window given by the customers. Due to time window constraints, the TSPTW and TRPTW are much harder than the traditional TSP and TRP.

The Travelling Salesman Problem with Time Windows (TRPTW) is a popular NP-hard combinatorial optimization problem studied much in the literature (Y. Dumas et al.,1995). The algorithms include exact and metaheuristic approaches. Langevin et al. (A. Langevin et al.,1993) introduced a two-commodity flow formulation to solve the problem. Dumas et al. (Y. Dumas et al.,1995) then used a dynamic programming approach. Similarly, Focacci et al. (M. Gendreau et al.,1998) brought constraint programming and optimization techniques together. Most recently, Dash et al. (S. Dash et al.,2012) propose a method using an IP model based on the discretization of time. The results are extremely good: several benchmark instances are solved first. Gendreau et al. (M. Gendreau et al.,1998) then proposed an insertion heuristic that generated the solution in the first step and improved it in a post-phase using removal and reinsertion of vertices. Ohlmann et al. (J. W. Ohlmann et al.,2007) developed simulated annealing relaxing the time windows constraints by integrating a variable penalty method within a stochastic search procedure. In this work, they developed a two-phase heuristic. In the first phase, a feasible solution was created by using a Variable Neighborhood Search (VNS). In the post phase, this solution was improved by using a General VNS. Generally speaking, the results from this approach are very promising.

In the Travelling Repairman Problem with Time Windows (TRPTW), there are an exact algorithm and three metaheuristics algorithms in the literature: 1) Tsitsiklis (J. N. Tsitsiklis et al.,1992) proposed a polynomial algorithm when several customers are bounded; 2) Heilporn et al. (G. Heilporna et al.,2010) then developed an exact algorithm and heuristic algorithm to solve the problem; 3) Ban et al. (H.B. Ban et al.,2017; H.B. Ban e proposed two metaheuristic algorithms based on Variable Neighborhood Search (VNS) scheme. Their experimental results showed the efficiency of the metaheuristic approach.

1.2 The MFEA literature

Up to date, several close variants of the MFEA framework are also introduced in the literature. Y. Yuan (Q. Xu et al.,2021) firstly developed evolutionary multitasking in permutation-based optimization problems. They tested it on several popular combinatorial problems. The experiment results indicated the promising scalability of evolutionary multitasking to many-task environments. E. Osaba et al. (E. Osaba et al.,2020) proposed a dMFEA-II framework to exploit the complementarities among several tasks, often achieved via genetic information transfer. Their algorithm controls the knowledge transfer by adjusting the crossover probability value. The technique allows good knowledge to transfer between tasks. However, the drawback of the two papers is that there is a lack of a mechanism to exploit the good solution space explored by MFEA. Therefore, these algorithms cannot balance exploration and exploitation effectively. Recently, Ban et al. (H.B. Ban et al.,2022) have applied the MFEA with RVNS to successfully solve two problems, TSP and TRP. Its performance encourages us to use the combination to solve the TSPTW and TRPTW. This paper considers these works as a baseline for our research.

1.3 Our contributions

Currently, various algorithms have been proposed to solve them. However, they solve each problem independently. This paper proposes an algorithm based on the MFEA framework to solve two problems simultaneously. The major contributions of this work are as follows:

- From the algorithmic aspect, we develop a first metaheuristic inspired by the MFEA framework. The proposed metaheuristic utilizes the advantages of the MFEA and RVNS. The MFEA with the RVNS to have good transferrable knowledge between tasks from the MFEA and the ability to exploit good solution spaces from RVNS. Therefore, the proposed algorithm balances exploration and exploitation.

- From the computational aspect, numerical experiments show that the proposed algorithm reaches nearly optimal 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 preliminary, respectively. Section 4 describes the proposed algorithm. Computational evaluations are reported in Section 5. Section 7 is conclusions and future work.

2 THE FORMULATION AND METHODOLOGY

2.1 The formulation

We consider an example that describes the difference between two problems in a specific instance. If we use the optimal solution of n40w160.002 instance for the TSPTW [1], the objective function cost (using the

[1] https://homepages.dcc.ufmg.br/∼ rfsilva/tsptw/

function cost of the TRPTW) of this solution for the TRPTW is 7519, while the known-best cost for this instance for the TRPTW is 6351 (the known-best cost is found by our algorithm). Thus, the difference between the two objective function costs is 15.5%. It implies that a good metaheuristic algorithm for the TSPTW does not produce a good solution for the TRPTW and vice versa. The above algorithms are the best algorithms for two problems. However, they only solve each problem independently but cannot simultaneously produce good solutions for two problems.
We have an complete graph $K_n = (V,E)$, where $V = v_1, v_2, ..., v_n$ is a set of vertices showing the starting vertex and customer locations, and E the set of edges connecting the customer locations. Suppose that, in a tour $T = (v_1 = s, v_2 ..., v_n)$, each edge $(v_i, v_j) \in E$ connecting the two vertices v_i and v_j there exists a cost $c(v_i, v_j)$. This cost represents the travel time between vertex v_i and v_j. Each vertex $v_i \in V$ has a time window $[e_i, l_i]$ indicating when starting service time at vertex v_i. This implies that a vertex v_i may be reached before the start e_i, but service cannot start until e_i and no latter than l_i of its time window. Moreover, to serve each customer, the salesman spends a mount of time. Let $D(v_i), S(v_i)$ be the time at which service begins and the service time at vertex v_i. It is calculated as follows: $D(v_i) = \max\{A(v_i), e_i\}$, where $A(v_k) = D(v_{i-1}) + S(v_{i-1}) + c(v_{i-1}, v_i)$ is the arrival time at vertex v_i in the tour. A tour is feasible, if and only if $A(v_i) \leq l_i$ for all vertices. The objective functions of two problems is defined as follows:

- In the TSPTW, the salesman must return to s. Therefore, the cost of the tour T is defined as: $\sum_{i=1}^{n} c(v_i, v_{i+1})$. Note that: $v_{n+1} \equiv s$
- In the TRPTW, we also define the travel duration of vertex v_i as the difference between the beginning of service at vertex v_i and the beginning of service at s: $t_i = D(v_i) - D(s)$. The cost of the tour T is defined: $\sum_{i=2}^{n} t_i$.

Two problems consist of determining a tour, starting at the starting vertex v_1, minimizing the cost of the tour overall vertices while respecting time windows. First, note that: the man must start and end at vertex v_1.

2.2 Our methodology

For NP-hard problems, we have three approaches to solve the problem, specifically, 1) exact algorithms, 2) approximation algorithms, and 3) heuristic (or metaheuristic) algorithms:

- The exact approaches find the optimal solution. However, they are exponential time algorithms in the worst case.
- An α-approximation algorithm generates a solution that has a factor of α of the optimal solution.
- Heuristic (metaheuristic) algorithms perform well in practice and validate their performance through experiments. This approach is suitable for a problem with large sizes.

Previously, several metaheuristics have been proposed to solve the TSPTW (Y. Dumas et al.,1995; F. Focacci et al.,2002; and the TRPTW (H.B. Ban et al.,2017; H.B. Ban et al.,2021; G. Heilporna et al.,2010; J. N. Tsitsiklis et al.,1992). However, they are developed to solve each problem independently and separately. Therefore, they cannot solve both two problems well simultaneously. When we run the two best algorithms for two tasks independently, there is no transferrable knowledge between tasks, and we cannot improve solution quality.

This paper proposes a MFEA approach to solve two problems simultaneously. Our MFEA solves two tasks simultaneously: the first task is the TRPTW, and the second is the TSPTW. Experiment results indicate its efficiency: 1) for small instances, the proposed algorithm obtains the optimal solutions in both two problems; 2) for large ones, our solutions are close to the optimal ones, even much better than those of the previous MFEA approaches.

3 PRELIMINARY

The overview of multifactorial optimization is introduced in (S. Dash et al.,2012; A. Gupta et al.,2016). Assume that, k optimization problems are needed to be performed simultaneously. Without loss of generality, 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 be found k solutions $\{x_1, x_2, ..., x_{k-1}, x_k\} = \min\{f_1(x), f_2(x), ..., f_{k-1}(x), f_k(x)\}$, where x_j is a feasible solution in X_j. Each f_j is considered as an

Figure 1. The similarity and difference between EA and MFEA

additional factor that impacts the evolution of a single population of individuals.Therefore, the problem also is called $k-$ factorial problem. For the problem, a general method to compare individuals is important. Each individual $p_i(i \in \{1,2,...,|P|\})$ 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^i_j of the individual p_i is its fitness value for task T_j $(1 \le j \le k)$.
- Factorial rank r^i_j of p_i on the task T_j is the index in the set of individuals sorted in ascending order in terms of c^i_j.
- Scalar-fitness ϕ_i of p_i is given by its best factorial rank overall tasks as $\phi_i = \frac{1}{\min_{j \in 1,...,k} r^i_j}$.
- Skill-factor ρ_i of p_i is the one task, amongst all other tasks, on which the individual is most effective, i.e., $\rho_i = argmin_j\{r^i_j\}$ where $j \in \{1,2,...,k\}$.

The pseudo-code of the basic MFEA is described in Figure 1 (H.B. Ban et al.,2022): We first build the unified search space that encompasses all individual search spaces of different tasks to have a shared background on which the transfer of information can take place. We then initialize SP individuals (SP is the size of population) in the unified search space and then evaluate it by calculating the skill-factor of each individual. After the initialization, the iteration begins to produce the offsprings and assign skill-factors to them. Selective evaluation guarantees that the skill-factor of each new offspring is selected randomly among those of the parents. The offspring and the parent are merged in a new population with $2 \times SP$ individuals. The evaluation for each individual is taken only on the assigned task (in the step, the best solution for each task is updated if it is found. This best solution for each task is the output). After evaluation, the individuals of the population receive new skill-factors. The Elitist strategy keeps the SP solutions with the best skill-factors for the next generation.

Figure 1 (H.B. Ban et al.,2022) also shows the differences between the traditional EA and MFEA. The crossover and mutation operators in the MFEA are like the traditional EA. However, there are two different important aspects: 1) the parents' skill-factor and 2) random mating probability (*rmp*). Specifically, the child is created using crossover from parents with the same skill-factor. Otherwise, the child is generated by a crossover with a *rmp* value or by a mutation when parents own different skill-factors. A large *rmp* value generates more information exchanging between tasks. Also, in the traditional GA, the fitness of child is evaluated directly, while the skill-factor is assigned to it in the MFEA.

The MFEA is also unlike multiobjective Optimization . In multiobjective optimization, we have one problem with many objective functions. On the other hand, the MFEA solves many tasks at the same time. In addition, multiobjective optimization generally uses a single representative space, while the MFEA unifies multiple representative spaces for many tasks.

Running two algorithms for two tasks independently is not the idea of the MFEA approach. When two algorithms run independently, each task is represented by its own search space. There is no transferrable knowledge between tasks. Otherwise, in the MFEA, two tasks use the unified search space, and transferrable knowledge between tasks is done. It can increase convergence and improve the quality of solutions for multitasks. Lian et al. (Y. C. Lian et al.,2019) then provided a novel theoretical analysis and evidence of the efficiency of the MFEA. This study theoretically explains why some existing the MFEA perform better than traditional EAs. In addition, the MFEA also can be useful in a system with limited computation.

4 THE PROPOSED ALGORITHM

This section introduces the pseudocode of the proposed MFEA. The TSPTW task corresponds to a particular task in the MFEA, while another is the TRPTW task. The flow of the proposed algorithm is described in Figure 2. Our MFEA has core components: unified representation, assortative mating (crossover and mutation operators), selective evaluation, scalar-fitness-based selection, RVNS, and Elitism. The detail of the algorithm is shown in Algorithm 1. More specifically, the algorithm includes the following steps. In the first step, a unified search space is created for both two problems. The population with *SP* individuals is then generated in the second step. All solutions for the population must be feasible. After that, the iteration begins until the termination criterion is satisfied. Parents are selected to produce offsprings using crossover or mutation and then assign skill-factors to them. The offsprings then are added to the current population. The individuals of the population are evaluated to update their scalar-fitness and skill-factor. We select the best solutions in terms of skill-factor from the current population and convert them from the unified representation to each task's one. It then is fed into the RVNS step to find the best solution for each task. The output of the RVNS is then converted to the unified search space. Finally, it is added to the population. The Elitist strategy keeps the *SP* solutions with the best skill-factors for the next generation.

4.1 Creating Unified Search Space-USS

In the literature, various representations are proposed for two problems. Among these representations, the permutation representation shows the efficiency compared to the others. In the permutation, each individual is coded as a set of n vertices $(v_1, v_2, ..., v_k, ..., v_n)$, where k is the $k-$th index. Figure 3 demonstrates the encoding for two problems.

4.2 Initializing population

Each feasible solution is created from the RVNS to take a role as an individual in the population. Therefore, we have Sp individuals in the initial population for the genetic step.

Algorithm 2 describes the constructive step. The objective function is the sum of all positive differences between the arrival time $(D(v_i))$ and its due time (l_i) on each vertex. Specifically, it is $\min \sum_{i=1}^{n} \max(0, D(v_i) - l_i)$. The algorithm runs until it finds a feasible solution. Restricted Candidate List (*RCL*) is created by ordering all non-selected vertices based on a greedy function that evaluates the benefit of including them in the solution. One vertex is then chosen from *RCL* randomly. Since all vertices are visited, we receive a solution. If this solution is a feasible one, it is an initial solution, and this step stops. Conversely, a repair procedure based on the RVNS with many neighborhoods (D. S. Johnson et al.,1995) is invoked, and the procedure iterates until a feasible solution is reached. The solution is shaken to escape from the current local optimal solution. The RVNS is then applied to create the new solution. If it is better than the best-found solution, it is set to the new current solution. The *level* is increased by one if the

Figure 2. The flow of the proposed MFEA

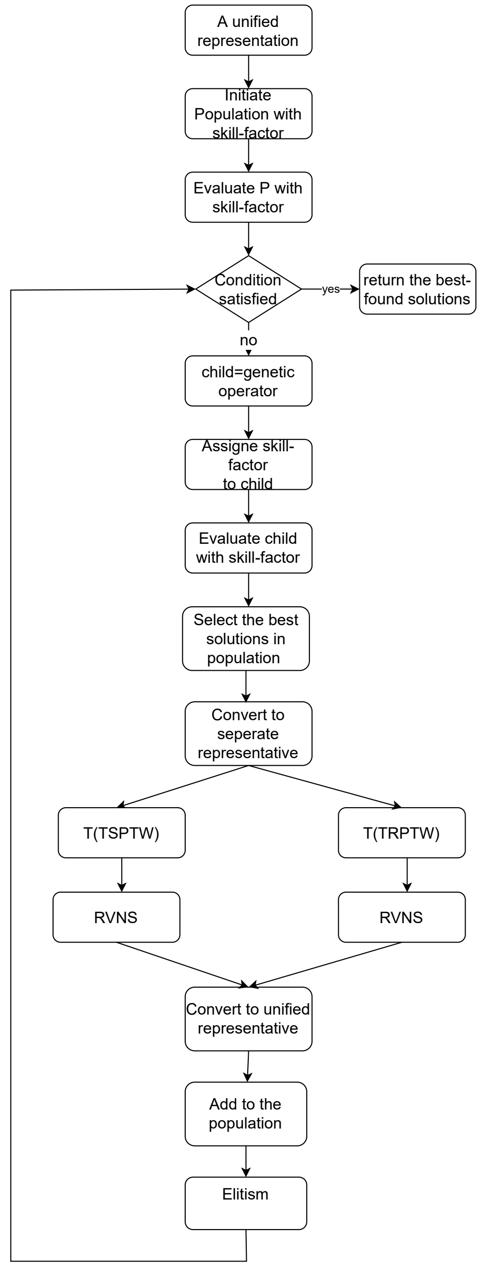
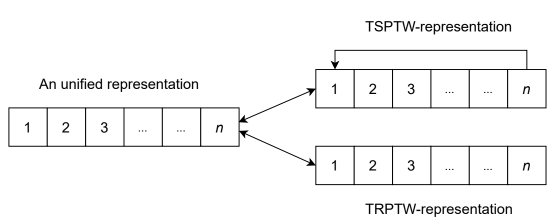


Figure 3. The interpretation of unified representation for each task



current solution is not improved, or reset to 1, otherwise. The repair procedure is described in Algorithm 3.

Algorithm 1 MFEA-RVNS

Require: K_n, C_{ij}, v_1, SP are the graph, the cost matrix, the starting vertex, and the size of population.
Ensure: The best solution T^*_{TSPTW}, T^*_{TRPTW}.

$T^*_{TSPTW}, T^*_{TRPTW} \longrightarrow Inf$; {Initiate the best solution for the TSPTW, TRPTW}
$P =$ Construction$(v_1, V, k, \alpha, level)$; {Initiate the population}
while (The termination criterion of the MFEA is not satisfied) **do**
  {MFEA step (exploration)}
  for $(j = 1; j \leqslant SP; j++)$ **do**
    (P, M) = Selection(P, NG); {select parents to mate}
    if (M and P have the same skill-factor) or (rand(1) $\leqslant rmp$) **then**
      if (M and P have the same skill-factor) **then**
        C_1, C_2 = Crossover(P, M);
        C_1, C_2's skill-factors are set to the skill-factors off P or M, respectively;
      else if (rand(1) $\leqslant rmp$) **then**
        C_1, C_2 = Crossover(P, M);
        C_1, C_2's skill-factors are set to the skill-factors off P or M randomly;
      if (C_1 or C_2 is infeasible) **then**
        if C_1 is infeasible **then**
          $C_1 =$ Repair(C_1);{convert it to feasible one}
        if C_2 is infeasible **then**
          $C_2 =$ Repair(C_2);{convert it to feasible one}
    else
      C_1 = Mutate(P);
      C_2 = Mutate(M);
      if (C_1 or C_2 is infeasible) **then**
        if C_1 is infeasible **then**
          $C_1 =$ Repair(C_1);{convert it to feasible one}
        if C_2 is infeasible **then**
          $C_2 =$ Repair(C_2);{convert it to feasible one}
      C_1's, C_2's skill-factor is set to P, M, respectively;
    $P = P \cup \{C_1, C_2\}$;
  Update scalar-fitness and skill-factor for all individuals in P;
  LT = Select the best individuals from P;
  {RVNS step (exploitation)}
  for each T in LT **do**
    (T_{TSPTW}, T_{TRPTW}) = Convert T from unified representation to one for each task;
    T'_{TSPTW} = RVNS(T_{TSPTW}); {local search}
    if ($T'_{TSPTW} < T^*_{TSPTW}$) **then**
      $T'_{TSPTW} \longrightarrow T^*_{TSPTW}$;
    T'_{TRPTW} = RVNS(T_{TRPTW}); {local search}
    if ($T'_{TRPTW} < T^*_{TRPTW}$) **then**
      $T'_{TRPTW} \longrightarrow T^*_{TRPTW}$;
    T' = convert(T'_{TSPTW}, T'_{TRPTW}) to unified representation;
    $P = P \cup \{T'\}$;
  P = Elitism-Selection(P);{keep the best SP individuals}
return T^*_{TSPTW}, T^*_{TRPTW};

In this paper, some neighborhoods widely applied in the literature (D. S. Johnson et al.,1995). We describe more details about seven neighborhoods as follows:

- **move** moves a vertex forward one position in T.
- **shift** relocates a vertex to another position in T. .
- **swap-adjacent** attempts to swap each pair of adjacent vertices in the tour.

Algorithm 2 Construction

Require: $v_1, V, k, \alpha, level$ are a starting vertex, the set of vertices in K_n, the number of vehicles and the size of RCL, the parameter to control the strength of the perturbation procedure, respectively.
Ensure: An initial solution T.

$P = \phi$; {Initially, the population is empty}
while ($|P| < SP$) **do**
  $T = \{v_1\}$; {T is a tour and it starts at v_1}
  while $|T| < n$ **do**
    Create RCL that includes α nearest vertices to v_e in V; {v_e is the last vertex in T}
    Select randomly vertex $v = \{v_i | v_i \in RCL$ and $v_i \notin T\}$;
    $T \leftarrow T \cup \{v_i\}$; {add the vertex to the tour}
  if T is infeasible solution **then**
    {Convert infeasible solution to feasible one}
    T = Repair($T, level\_max, N_i(i = 1, ..., 7)$);
  $P = P \cup \{T^{'}\}$; {add the tour to the population}
return P;

Algorithm 3 Repair$(T, level\_max, N_i(i = 1, ..., 7))$

Require: $T, level\_max, N_i(i = 1, ..., 7)$ are the infeasible solution, the parameter to control the strength of the perturbation procedure, and the number of neighbourhood respectively.
Ensure: An feasible solution T.

$level$ = 1;
while ((T is infeasible solution) and ($level \leq level\_max$)) **do**
  T'= Perturbation($T, level$);
  for $i : 1 \rightarrow 6$ **do**
    $T^{''} \leftarrow \arg\min N_i(T^{'})$; {local search}
    if $(L(T^{''} < L(T^{'}))$ **then**
      $T^{'} \leftarrow T^{''}$
      $i \leftarrow 1$
    else
      $i++$
  if $L(T^{'}) < L(T)$ **then**
    $T \leftarrow T^{'}$;
  if $L(T^{'}) == L(T)$ **then**
    $level \leftarrow 1$;
  else
    $level++$;
return T;

Algorithm 4 Selection Operator(P, NG)

Require: P, NG are the population and the size of group, respectively.
Ensure: Parents C_1, C_2.

Select randomly the NG individuals in the P;
Sort them in terms of their R values;
C_1, C_2 = Select two individuals with the best R values;
return C_1, C_2;

- **exchange** tries to swap the positions of each pair of vertices in the tour.
- **2-opt** removes each pair of edges from the tour and reconnects them.
- **Or-opt**: Three adjacent vertices are reallocated to another position of the tour.

Algorithm 5 Crossover(P, M)

Require: P, M are the parent tours, respectively.
Ensure: A new child T.

type = rand(3);
rnd = rand(2);
if (*type*==1) **then**
  {the first type crossover is selected}
  if (*rnd*==1) **then**
    C = **PMX**(P, M);{PMX is chosen}
  else if (*rnd*==2) **then**
    C = **CX**(P, M);{CX is selected}
else if (*type*==2) **then**
  {the second type is selected}
  if (*rnd*==1) **then**
    C = **EXX**(P, M);{EXX is selected}
  else if (*rnd*==2) **then**
    C = **EAX**(P, M);{EAX is selected}
else if (*type*==3) **then**
  {type 3 is selected}
  if (*rnd*==1) **then**
    C = **SC**(P, M);{SC is selected}
  else if (*rnd*==2) **then**
    C = **MC**(P, M);{MC is selected}

Algorithm 6 Mutate(C)

Require: C is the child tour, respectively.
Ensure: A new child C.

{Choose a mutation operator randomly}
rnd = rand(2);
if (*rnd*==1) **then**
  C = **Inversion**(C){Inversion mutation is selected}
else if (*rnd*==2) **then**
  C = **Insertion**(C){Insertion mutation is selected}
else
  C = **Swap**(C){Swap mutation is selected}
return C;

Algorithm 7 RVNS(T)

Require: T is a tour.
Ensure: A new solution T.

Initialize the Neighborhood List *NL*;
while $NL \neq 0$ **do**
  Choose a neighborhood N in *NL* at random
  $T^{'} \leftarrow \arg\min N(T)$; {Neighborhood search}
  if (($W(T^{'}) < W(T)$) and ($T^{'}$ is feasible)) **then**
    $T \leftarrow T^{'}$
    Update *NL*;
  else
    Remove N from the *NL*;
  if (($W(T^{'}) < W(T^{*})$) and ($T^{'}$ is feasible)) **then**
    $T^{*} \leftarrow T^{'}$;

4.3 Evaluating for individuals

The scalar-fitness function demonstrates the way of evaluating individuals. Scalar-fitness then are calculated for each individual. The larger and larger the scalar-fitness value is, the better and better the individual is.

4.4 Selection operator

In the original Tournament (H.B. Ban et al.,2022; E. G. Talbi et al.,2009), the fitness is the only criterion in choosing parents. This paper proposes a new selection operator for the MFEA algorithm that balances scalar-fitness and population diversity. The scalar-fitness is effectively transferred elite genes between tasks, while diversity is important when it can make a bottleneck against the genetic information transfer. For each solution, we count its scalar-fitness and its diversity in a set of solutions as follows:

$$R(T) = (SP - RF(T) + 1) + \alpha \times (SP - RD(T) + 1) \tag{1}$$

where SP, $\alpha \in [0,1]$, $RF(T)$, and $RD(T)$ are the population size, threshold, the rank of T in the P based on the scalar-fitness, and the rank of T in the P based on its diversity, respectively.

$$\overline{d}(T) = \frac{\sum_{i=1}^{n} d(T,T_i)}{n} \tag{2}$$

The metric distance between two solutions is the minimum number of transformations from one to another. We define the distance $d(T,T_i)$ to be n (the number of vertices) minus the number of vertices with the same position on T and T_i. Similarity, $\overline{d(T)}$ is the average distance of T in the population. The larger $\overline{d}(T)$ is, the higher its rank is. The larger R is, the better solution T is.

The selection operator selects individual parents based on their R values to mate. We choose the tournament selection operator (E. G. Talbi et al.,2009) because of its efficiency. A group of NG individuals is selected randomly from the population. Then, two individuals with the best R values in the group are chosen to become parents. The selection pressure can be increased by extending the size of the group. On average, the selected individuals from a larger group have higher R values than those of a small size. The detail in this step is described in Algorithm 4.

4.5 Crossover operator

The crossover is implemented with the predefined probability (*rmp*) or if the parents have the same skill-factor. When parents have the same skill-factor, we have inter-crossover. Otherwise, the intra-crossover is applied to parents with different skill-factor. It opens up the chance for knowledge transfer by using crossover-based exchange between tasks. In (A. Otman et al.,2015), the crossovers are divided into three main types. We found no logical investigation showing which operator brings the best performance in the literature. In a preliminary study, we realize that the algorithm's effectiveness relatively depends on selected crossover operators. Since trying all operators leads to computationally expensive efforts, our numerical analysis is conducted on randomly selected operators for each type. The following operators are selected from the study to balance solution quality and running time.

- The first type is related to the position of certain genes in parents (PMX, CX).
- The second selects genes alternately from both parents, without genes' repetition (EXX, EAX).
- The third is an order-based crossover (SC, MC).

Initially, we select a crossover randomly. If any improvement of the best solution is found, the current crossover operator is continued to use. Otherwise, if the improvement of the best solution is not found after the number of generations (*NO*), another crossover operator is replaced randomly. Using multiple crossovers helps the population be more diverse than one crossover. Therefore, these operators prevent the algorithm from premature convergence. If the offsprings are infeasible, the fix procedure is invoked to convert them to feasible ones. The offsprings' skill-factors are set to the one of the father or mother randomly. The detail in this step is described in Algorithm 5.

4.6 Mutation operator

A mutation is used to keep the diversity of the population. Some mutations are used in the proposed algorithm:

- The Inversion Mutation picks two vertices at random and then inverts the list of vertices between them. It preserves most adjacency information and only breaks two links, leading to the disruption of order information.
- The Insertion Mutation removes the vertex from the current index and then inserts it in a random index on the solution. The operator preserves most of the order and the adjacency information.
- Swap Mutation selects two vertices at random and swaps their positions.

It preserves most of the adjacency information, but links broken disrupt order more. We randomly select one of three operators when this mutation is performed. After the mutation operator, two offsprings are created from the parents. If the offsprings are infeasible, the repair procedure converts them to feasible ones. Their skill-factors are set to those of parents, respectively. The detail in the mutation is described in Algorithm 6.

4.7 RVNS

The combination between the MFEA and the RVNS allows good transferrable knowledge between tasks from the MFEA and the ability to exploit good solution spaces from RVNS. We select some best solutions in the current population to feed into the RVNS. In the RVNS step, we convert a solution from unified representation to separated representation for each task. The RVNS then applies to each task separately. Finally, the output of the RVNS is represented in the unified space. The improved solution will be added to the population.

For this step, we use popular neighborhoods such as move, shift, swap-adjacent, exchange, 2-opt, and or-opt in (D. S. Johnson et al.,1995; C. R. Reeves et al.,1999). In addition, the pseudocode of the RVNS algorithm is given in Algorithm 7.

4.8 Elitism operator

Elitism is a process that ensures the survival of the fittest so they do not die through the evolutionary processes. Researchers show the number (E. G. Talbi et al.,2009) (usually below 15%) of the best solutions that automatically go to the next generation. The proposed algorithm selects Sp individuals for the next generation, in which about 15% of them are the best solutions in the previous generation, and the remaining individuals are chosen randomly from P.

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

5 COMPUTATIONAL EVALUATIONS

The experiments are conducted on a personal computer equipped with a Xeon E-2234 CPU and 16 GB bytes of RAM. The program was coded in C language. Generation number (Ng), population size (SP), group size (NG) in selection, and crossover rates (rmp) influences to the algorithm's results. Many efforts in the literature studied the algorithm sensitivity to parameter changes. We found that no work shows which values are the best for all cases. However, the following suggestions help us in choosing parameter values:

- A large generation number does not improve performance. Besides, it consumes much time to run. A small value makes the algorithm fail to reach the best solution (M. Angelova et al.,2011).
- A higher crossover value obtains new individuals more quickly while a low crossover rate may cause stagnation (M. Angelova et al.,2011).
- A large population size can increase the population diversity. However, it can be unhelpful in the algorithm and increase the running time of it (T. Chena et al.,2017).
- Increased selection pressure can be provided by simply increasing the group size. When the selection pressure is too low, the convergence rate is slower, while if it is too high, the chance of the algorithm prematurely converges (Y. Lavinas et al.,1993).

Table 1. The variable parameters

Parameter	Value Range
SP	$50 \le \beta_r \le 200$, incremented by 50
NG	$5 \le \alpha \le 15$, incremented by 5
rmp	$0.5 \le \beta_\eta \le 1$, incremented by 0.1
α	$5 \le \tau_0 \le 20$, incremented by 5
$level$	$5 \le p \le 15$, incremented by 5
Ng	$50 \le Ng \le 150$, incremeted by 50

- The α and level values help to create the diversity of the initial population. A larger value leads to the same as the random method, while a small value decreases the diversity.

Based on the suggestions, we determine a suitable range for each parameter in Table 1. In the next step, we choose the best value from the range as follows: finding the best configuration by conducting all instances would have been too expensive in computation, and we test numerical analysis on some instances. The configuration selected in many combinations is tested, and the one that has obtained the best solution is chosen. In Table 1, we determine a range for each parameter that generates different combinations, and we run the proposed algorithm on some selected instances of the combinations. We find the following settings so that our algorithm obtains the best solutions: $SP = 100, NG = 5, rmp = 0.7, \alpha = 10, level = 5$, and $Ng = 100$. This parameter setting has thus been used in the following experiments.

We found no algorithm based on the literature's MFEA framework for the TRPTW and TSPTW. Therefore, the proposed algorithm's results directly compare with the known best solutions of the TSPTW and TRPTW on the same benchmark. Moreover, to compare with the previous MFEA framework (E. Osaba et al.,2020; Y. Yuan et al.,2016), our MEFA is tested on the benchmark for the TSP and TRP. They are specific variants of TSPTW and TRPTW without time window constraints. Therefore, the instances are used in the paper as follows:

- Dumas et al. propose the first set citebib09 and contains 135 instances grouped in 27 test cases. Each group has five Euclidean instances, coordinates between 0 and 50, with the same number of customers and the same maximum range of time windows. For example, the instances n20w60.001, n20w60.002, n20w60.003, n20w60.004, and n20w60.005 have 20 vertices and the time window for each vertex is uniformly random, between 0 and 60.
- Gendreau et al. proposes the second set of instances citebib12 and contains 140 instances grouped in 28 test cases.
- Ohlmann et al. propose the third set of instances citebib30 and contains 25 instances grouped in 5 test cases.
- The fourth sets in the majority are the instances proposed by Dumas et al. (Y. Dumas et al.,1995) with wider time windows.
- The TSPLIB [2] includes fourteen instances from 50 to 100 instances.

The efficiency of the metaheuristic algorithm can be evaluated by comparing the best solution found by the proposed algorithm (notation: *Best.Sol*) to 1) the optimal solution (notation: *OPT*); and 2) the known best solution (notation: *KBS*) of the previous metaheuristics (note that: In the TSPTW, *KBS* is the optimal solution) as follows:

$$gap[\%] = \frac{Best.Sol - KBS(OPT)}{KBS(OPT)} \times 100\% \tag{3}$$

The smaller and smaller the value of *gap* is, the better and better our solution is. All instances and found solutions are available in the link [3].

[2] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/
[3] https://sites.google.com/soict.hust.edu.vn/mfea-tsptw-trptw/home

Table 2. Comparison the best-found values between MFEA-NR and MFEA for TSPTW and TRPTW instances proposed by Dumas et al. (Y. Dumas et al.,1995), and Silva et al. (R. F. Silva et al.,2010)

instances	MFEA-NR		MFEA		diff[%]	
	TSPTW	TRPTW	TSPTW	TRPTW	TSPTW	TRPTW
n20w20.002	286	2560	286	2560	0.00	0.00
n20w40.002	333	2679	333	2679	0.00	0.00
n20w60.002	244	2176	244	2176	0.00	0.00
n20w80.003	338	2669	338	2669	0.00	0.00
n20w100.002	222	2082	222	2082	0.00	0.00
n40w40.002	483	7202	461	7104	-4.55	-1.36
n40w60.002	487	7303	470	7247	-3.49	-0.77
n40w80.002	468	7209	431	7123	-7.91	-1.19
n40w100.002	378	6789	364	6693	-3.70	-1.41
n60w20.002	626	14003	605	13996	-3.35	-0.05
n60w120.002	472	12622	427	12525	-9.53	-0.77
n60w140.002	475	11914	464	11810	-2.32	-0.87
n60w160.002	443	12920	423	12719	-4.51	-1.56
n80w120.002	587	18449	577	18383	-1.70	-0.36
n80w140.002	495	18243	472	18208	-4.65	-0.19
n80w160.002	588	17334	553	17200	-5.95	-0.77
aver					-3.23	-0.58

Table 3. Comparison the best-found values between MFEA-NLS and MFEA for TSPTW and TRPTW instances proposed by Dumas et al. (Y. Dumas et al.,1995), and Silva et al. (R. F. Silva et al.,2010)

instances	MFEA-NLS		MFEA		diff[%]	
	TSPTW	TRPTW	TSPTW	TRPTW	TSPTW	TRPTW
n20w20.002	286	2560	286	2560	0.00	0.00
n20w40.002	333	2679	333	2679	0.00	0.00
n20w60.002	244	2176	244	2176	0.00	0.00
n20w80.003	338	2669	338	2669	0.00	0.00
n20w100.002	222	2082	222	2082	0.00	0.00
n40w40.002	522	7530	461	7104	-11.69	-5.66
n40w60.002	503	7517	470	7247	-6.56	-3.59
n40w80.002	475	7763	431	7123	-9.26	-8.24
n40w100.002	400	7502	364	6693	-9.00	-10.78
n60w20.002	626	14097	605	13996	-3.35	-0.72
n60w120.002	480	13680	427	12525	-11.04	-8.44
n60w140.002	502	12951	464	11810	-7.57	-8.81
n60w160.002	461	13953	423	12719	-8.24	-8.84
n80w120.002	620	19860	577	18383	-6.94	-7.44
n80w140.002	520	19742	472	18208	-9.23	-7.77
n80w160.002	614	19516	553	17200	-9.93	-11.87
aver					-5.80	-5.14

Table 4. Comparison between our results with the best-found values for TSPTW and TRPTW instances proposed by Dumas et al. (Y. Dumas et al.,1995), and Silva et al. (R. F. Silva et al.,2010)

Instances	TSPTW	TRPTW	MFEA							
			TSPTW				TRPTW			
	OPT	*KBS*	*Best.Sol*	*Aver.Sol*	*gap*	*Time*	*Best.Sol*	*Aver.Sol*	*gap*	*Time*
n20w20.001	378	2528	378	378	0.0	3	2528	2528	0.0	3
n20w20.002	286	2560	286	286	0.0	2	2560	2560	0.0	2
n20w20.003	394	2671	394	394	0.0	2	2671	2671	0.0	2
n20w20.004	396	2975	396	396	0.0	6	2975	2975	0.0	6
n20w40.001	254	2270	254	254	0.0	2	2270	2270	0.0	2
n20w40.002	333	2679	333	333	0.0	5	2679	2679	0.0	5
n20w40.003	317	2774	317	317	0.0	3	2774	2774	0.0	2
n20w40.004	388	2568	388	388	0.0	2	2568	2568	0.0	3
n20w60.001	335	2421	335	335	0.0	3	2421	2421	0.0	2
n20w60.002	244	2176	244	244	0.0	2	2176	2176	0.0	2
n20w60.003	352	2694	352	352	0.0	2	2694	2694	0.0	2
n20w60.004	280	2020	280	280	0.0	2	2020	2020	0.0	2
n20w80.001	329	2990	329	329	0.0	2	2990	2990	0.0	3
n20w80.002	338	2669	340	340	0.6	1	2669	2669	0.0	2
n20w80.003	320	2643	320	320	0.0	2	2643	2643	0.0	2
n20w80.004	304	2627	306	306	0.7	3	2552	2552	-2.9	2
n20w100.001	237	2294	237	237	0.0	3	2269	2269	-1.1	3
n20w100.002	222	2082	222	222	0.0	2	2082	2082	0.0	2
n20w100.003	310	2416	310	310	0.0	2	2416	2416	0.0	3
n20w100.004	349	2914	349	349	0.0	1	2862	2862	-1.8	2
n40w20.001	500	7875	500	500	0.0	9	7875	7875	0.0	8
n40w20.002	552	7527	552	552	0.0	7	7527	7527	0.0	8
n40w20.003	478	7535	478	478	0.0	8	7535	7535	0.0	9
n40w20.004	404	7031	404	404	0.0	8	7031	7031	0.0	9
n40w40.001	465	7663	465	465	0.0	7	7663	7663	0.0	9
n40w40.002	461	7104	461	461	0.0	8	7104	7104	0.0	8
n40w40.003	474	7483	474	474	0.0	8	7483	7483	0.0	8
n40w40.004	452	6917	452	452	0.0	8	6917	6917	0.0	9
n40w60.001	494	7066	494	494	0.0	9	7066	7066	0.0	7
n40w60.002	470	7247	470	470	0.0	8	7247	7247	0.0	8
n40w60.003	408	6758	410	410	0.0	9	6758	6758	0.0	8
n40w60.004	382	5548	382	382	0.0	9	5548	5548	0.0	9
n40w80.001	395	8229	395	395	0.0	8	8152	8152	0.0	9
n40w80.002	431	7176	431	431	0.0	8	7123	7123	0.0	9
n40w80.003	412	7075	418	418	0.0	8	7075	7075	0.0	9
n40w80.004	417	7166	417	417	0.6	9	7166	7166	0.0	10
n40w100.001	429	6858	432	432	0.0	8	6800	6800	0.0	9
n40w100.002	358	6778	364	364	0.7	11	6693	6693	-2.9	10
n40w100.003	364	6178	364	364	0.0	9	6926	6926	-1.1	11
n40w100.004	357	7019	361	361	0.0	9	7019	7019	0.0	8
n60w20.002	605	13996	605	605	0.0	18	13996	13996	0.0	19
n60w20.003	533	13782	533	533	0.0	17	12965	12965	-1.8	18
n60w20.004	616	12965	616	616	0.0	17	15102	15102	0.0	18
n60w40.003	603	15034	612	612	0.0	19	15034	15034	0.0	19

Table 5. Comparison between our results with the best-found values for TSPTW and TRPTW instances proposed by Dumas et al. (Y. Dumas et al.,1995), and Silva et al. (R. F. Silva et al.,2010) (continue)

Instances	TSPTW	TRPTW	MFEA							
			TSPTW				TRPTW			
	OPT/ KBS	*KBS*	*Best.Sol*	*Aver.Sol*	*gap*	*Time*	*Best.Sol*	*Aver.Sol*	*gap*	*Time*
n60w160.004	401	11645	401	401	0.0	19	11778	11778	1.1	19
n60w180.002	399	12015	399	399	0.0	17	12224	12224	1.7	21
n60w180.003	445	12214	445	445	0.0	18	12679	12679	3.8	21
n60w180.004	456	11101	456	456	0.0	19	11245	11245	1.3	18
n60w200.002	414	11748	414	414	0.0	20	11866	11866	1.0	19
n60w200.003	455	10697	460	460	1.1	19	10697	10697	0.0	18
n60w200.004	431	11441	431	431	0.0	16	11740	11740	2.6	17
n80w120.002	577	18181	577	577	0.0	17	18383	18383	1.1	21
n80w120.003	540	17878	548	548	1.5	19	17937	17937	0.3	21
n80w120.004	501	17318	501	501	0.0	18	17578	17578	1.5	18
n80w140.002	472	17815	472	472	0.0	19	18208	18208	2.2	21
n80w140.003	580	17315	580	580	0.0	20	17358	17358	0.2	21
n80w140.004	424	18936	424	424	0.0	19	19374	19374	2.3	18
n80w160.002	553	17091	553	553	0.0	27	17200	17200	0.6	18
n80w160.003	521	16606	521	521	0.0	21	16521	16521	-0.5	20
n80w160.004	509	17804	509	509	0.0	20	17927	17927	0.7	18
n80w180.002	479	17339	479	479	0.0	21	17904	17904	3.3	19
n80w180.003	530	17271	530	530	0.0	20	17160	17160	-0.6	20
n80w180.004	479	16729	479	479	0.0	22	16849	16849	0.7	21
n100w120.002	540	29882	556	556	3.0	38	29818	29818	0.0	45
n100w120.003	617	25275	646	646	4.7	37	24473	24473	0.0	42
n100w120.004	663	30102	663	663	0.0	39	31554	31554	0.0	41
n100w140.002	622	30192	632	632	1.6	38	30087	30087	0.0	45
n100w140.003	481	28309	481	481	0.0	39	28791	28791	0.0	47
n100w140.004	533	27448	533	533	0.0	40	27990	27990	0.0	45
n150w120.003	747	42340	769	769	2.9	75	42339	42339	0.0	72
n150w140.001	762	42405	785	785	3.0	70	42388	42388	-0.1	74
n150w160.001	706	45366	732	732	3.6	72	45160	45160	-0.4	78
n150w160.002	711	44123	735	735	3.3	74	44123	44123	0.0	76
n200w200.001	9424	1094630	9424	9424	0.0	101	1093537	1093537	-0.1	89
n200w200.002	9838	1099839	9885	9885	0.5	110	1099839	1099839	0.0	86
n200w200.003	9043	1067171	9135	9135	1.0	99	1067161	1067161	0.0	93
n200w300.001	7656	1052884	7791	7791	1.7	100	1047893	1047893	-0.5	106
n200w300.002	7578	1047893	7721	7721	1.8	105	1047893	1047893	0.0	110
n200w300.003	8600	1069169	8739	8739	1.6	120	1069169	1069169	0.0	93
n200w300.004	8268	1090972	8415	8415	1.7	112	1090972	1090972	0.0	96
n200w300.005	8030	1022000	8190	8190	1.9	114	1016765	1016765	-5.1	98
n200w400.001	7420	1064456	7661	7661	3.2	109	1064456	1064456	0.0	100
aver					0.49	26.24			-0.11	25.6

Table 6. Comparison between our results with the best-found values for TSPTW and TRPTW instances proposed by Gendreau et al. (M. Gendreau et al.,1998), and Ohlmann et al. (J. W. Ohlmann et al.,2007)

Instances	TSPTW	TRPTW	MFEA							
			TSPTW				TRPTW			
	KBS	*KBS*	*Best.Sol*	*Aver.Sol*	*gap*	*T*	*Best.Sol*	*Aver.Sol*	*gap*	*Time*
n20w120.002	218	2193	218	218	0.0	2	2193	2193	0.0	3
n20w120.003	303	2337	303	303	0.0	4	2337	2337	0.0	2
n20w120.004	300	2686	300	300	0.0	2	2686	2686	0.0	2
n20w140.002	272	2330	272	272	0.0	2	2330	2330	0.0	3
n20w140.003	236	2194	236	236	0.0	2	2196	2196	0.1	2
n20w140.004	255	2279	264	264	3.5	4	2278	2278	0.0	5
n20w160.002	201	1830	201	201	0.0	2	1830	1830	0.0	2
n20w160.003	201	2286	201	201	0.0	3	2286	2286	0.0	2
n20w160.004	203	1616	203	203	0.0	2	1616	1616	0.0	2
n20w180.002	265	2315	265	265	0.0	4	2315	2315	0.0	2
n20w180.003	271	2414	271	271	0.0	2	2414	2414	0.0	2
n20w180.004	201	2624	201	201	0.0	3	1924	1924	-26.7	2
n20w200.002	203	1799	203	203	0.0	2	1799	1799	0.0	2
n20w200.003	249	2144	260	260	4.4	2	2089	2089	-2.6	1
n20w200.004	293	2624	293	293	0.0	1	2613	2613	-0.4	2
n40w120.002	445	6265	446	446	0.2	3	6265	6265	0.0	8
n40w120.003	357	6411	360	360	0.8	2	6411	6411	0.0	7
n40w120.004	303	5855	303	303	0.0	3	5855	5855	0.0	6
n40w140.002	383	5746	383	383	0.0	2	5746	5746	0.0	8
n40w140.003	398	6572	398	398	0.0	3	6572	6572	0.0	7
n40w140.004	342	5719	350	350	2.3	8	5680	5680	-0.7	8
n40w160.002	337	6368	338	338	0.3	9	6351	6351	-0.3	8
n40w160.003	346	5850	346	346	0.0	9	5850	5850	0.0	9
n40w160.004	288	4468	289	289	0.3	8	4440	4440	-0.6	9
n40w180.002	347	6104	349	349	0.6	8	6104	6104	0.0	9
n40w180.003	279	6040	282	282	1.1	7	6031	6031	-0.1	8
n40w180.004	354	6103	361	361	2.0	8	6283	6283	2.9	8
n40w200.002	303	6674	303	303	0.0	8	5830	5830	-12.6	9
n40w200.003	339	5542	343	343	1.2	7	5230	5230	-5.6	8
n40w200.004	301	6103	301	301	0.0	9	5977	5977	-2.1	10
n60w120.002	427	12517	427	427	0.0	19	12525	12525	0.1	19
n60w120.003	407	11690	419	419	2.9	19	11680	11680	-0.1	19
n60w120.004	490	11132	492	492	0.4	13	11135	11135	0.0	19
n60w140.002	462	11782	464	464	0.4	18	11810	11810	0.2	19
n60w140.003	427	13128	448	448	4.9	13	13031	13031	-0.7	16
n60w140.004	488	13189	488	488	0.0	15	12663	12663	-4.0	15
n60w160.002	423	12471	423	423	0.0	19	12719	12719	2.0	17
n60w160.003	434	10682	447	447	3.0	14	10674	10674	-0.1	15

Table 7. Comparison between our results with the best-found values for TSPTW and TRPTW instances proposed by Gendreau et al. (M. Gendreau et al.,1998), and Ohlmann et al. (J. W. Ohlmann et al.,2007) (continue)

Instances	TSPTW	TRPTW	MFEA							
			TSPTW				TRPTW			
	OPT	*KBS*	*Best.Sol*	*Aver.Sol*	*gap*	*Time*	*Best.Sol*	*Aver.Sol*	*gap*	*Time*
n60w160.004	401	11645	401	401	0.0	19	11778	11778	1.1	19
n60w180.002	399	12015	399	399	0.0	17	12224	12224	1.7	21
n60w180.003	445	12214	445	445	0.0	18	12214	12679	0.0	21
n60w180.004	456	11101	456	456	0.0	19	11245	11245	1.3	18
n60w200.002	414	11748	414	414	0.0	20	11866	11866	1.0	19
n60w200.003	455	10697	460	460	1.1	19	10697	10697	0.0	18
n60w200.004	431	11441	431	431	0.0	16	11441	11441	0.0	17
n80w120.002	577	18181	577	577	0.0	17	18383	18383	1.1	21
n80w120.003	540	17878	548	548	1.5	19	17937	17937	0.3	21
n80w120.004	501	17318	501	501	0.0	18	17578	17578	1.5	18
n80w140.002	472	17815	472	472	0.0	19	17815	17815	0.0	21
n80w140.003	580	17315	580	580	0.0	20	17358	17358	0.2	21
n80w140.004	424	18936	424	424	0.0	19	18936	18936	0.0	18
n80w160.002	553	17091	553	553	0.0	27	17200	17200	0.6	18
n80w160.003	521	16606	521	521	0.0	21	16521	16521	-0.5	20
n80w160.004	509	17804	509	509	0.0	20	17927	17927	0.7	18
n80w180.002	479	17339	479	479	0.0	21	17339	17339	0.0	19
n80w180.003	530	17271	530	530	0.0	20	17160	17160	-0.6	20
n80w180.004	479	16729	479	479	0.0	22	16849	16849	0.7	21
n100w120.002	540	29882	556	556	3.0	38	29818	29818	0.0	45
n100w120.003	617	25275	646	646	4.7	37	24473	24473	0.0	42
n100w120.004	663	30102	663	663	0.0	39	31554	31554	0.0	41
n100w140.002	622	30192	632	632	1.6	38	30087	30087	0.0	45
n100w140.003	481	28309	481	481	0.0	39	28791	28791	0.0	47
n100w140.004	533	27448	533	533	0.0	40	27990	27990	0.0	45
aver					0.64	13.6			-0.44	14.7

Table 8. Comparison between our results with the best-found values for TSPTW and TRPTW instances proposed by Gendreau et al. (M. Gendreau et al.,1998), and Ohlmann et al. (J. W. Ohlmann et al.,2007)

Instances	TSPTW	TRPTW		MFEA					
				TSP			TRP		
	OPT	*BKS* Ban et al.	*BKS* Heilporna et al.	*Best.Sol*	*Aver.Sol*	*Time*	*Best.Sol*	*Aver.Sol*	*Time*
n20w120.001	274	2175	2535	274	274	2	2175	2175	2
n20w140.001	176	1846	1908	176	176	2	1826	1826	2
n20w160.001	241	2146	2150	241	241	2	2148	2148	2
n20w180.001	253	2477	2037	253	253	2	2477	2477	2
n20w200.001	233	1975	2294	233	233	2	1975	1975	2
n40w120.001	434	6800	7496	434	434	9	6800	6800	9
n40w140.001	328	6290	7203	328	328	10	6290	6290	10
n40w160.001	349	6143	6657	349	349	11	6143	6143	12
n40w180.001	345	6952	6578	345	345	12	6897	6897	11
n40w200.001	345	6169	6408	345	345	10	6113	6113	13
n60w120.001	392	11120	9303	392	392	25	11288	11288	28
n60w140.001	426	10814	9131	426	426	26	10981	10981	27
n60w160.001	589	11574	11422	589	589	27	11546	11546	28
n60w180.001	436	11363	9689	436	436	24	11646	11646	25
n60w200.001	423	10128	10315	423	423	25	9939	9939	27
n80w120.001	509	11122	11156	512	509	41	16693	16693	45
n80w140.001	530	14131	14131	530	530	42	14131	14131	47
n80w180.001	605	11222	11222	605	605	41	11222	11222	42

Table 9. The average results for TSPTW, TRPTW instances

Instances	TSPTW		TRPTW	
	$\overline{gap}$	*Time*	$\overline{gap}$	*Time*
TSPTW	0.49	26.24	-0.11	26.5
TSPTW	0.64	13.6	-0.44	14.7
aver	0.56	19.9	-0.28	20.6

Table 10. The difference between the optimal TSPTW using TRPTW objective function and vice versa

Instances	TRPTW			TSPTW		
	TRPTW(OPT_TSPTW)	KBS	diff[%]	TSPTW(BKS_TRPTW)	KBS	diff[%]
n20w120.002	2592	2193	15.4	256	218	14.8
n20w140.002	2519	2330	7.5	311	272	12.5
n20w160.002	2043	1830	10.4	249	201	19.3
n40w120.002	6718	6265	6.7	552	446	19.2
n40w140.002	5865	5746	2.0	449	383	14.7
n40w160.002	7519	6351	15.5	456	338	25.9
n60w120.002	13896	12517	9.9	581	444	23.6
n60w140.002	12898	11795	8.6	616	464	24.7
n60w160.002	14091	12489	11.4	616	428	30.5
aver			9.7			20.6

Table 11. Comparison between our results with the others for TSP and TRP (Q. Xu et al.,2021)

Instances	OPT		YA (Q. Xu et al.,2021)		OA (J. N. Tsitsiklis et al.,1992)		MFEA							
	TSP	TRP	TSP	TRP	TSP	TRP	TSP				TRP			
			best.sol	*best.sol*	*best.sol*	*best.sol*	*best.sol*	*aver.sol*	*gap*	*Time*	*best.sol*	*aver.sol*	*gap*	*Time*
eil51	426*	10178*	446	10834	450	10834	426	426	0.00	23	10178	10178	0.00	22
berlin52	7542*	143721*	7922	152886	8276	152886	7542	7542	0.00	22	143721	143721	0.00	21
st70	675*	20557*	713	22283	772	22799	680	680	0.01	41	22283	22283	8.40	39
eil76	538*	17976*	560	18777	589	18008	559	559	0.04	43	18008	18008	0.18	40
pr76	108159*	3455242*	113017	3493048	117287	3493048	108159	108159	0.00	47	3455242	3455242	0.00	45
pr107	44303*	2026626*	45737	2135492	46338	2135492	45187	45187	0.02	71	2052224	2052224	1.26	72
rat99	1211*	58288*	1316	60134	1369	60134	1280	1280	0.06	66	58971	58971	1.17	65
kroA100	21282*	983128*	22233	1043868	22233	1043868	21878	21878	0.03	63	1009986	1009986	2.73	63
kroB100	22141*	986008*	23144	1118869	24337	1118869	23039	23039	0.04	64	1003107	1003107	1.73	63
kroC100	20749*	961324*	22395	1026908	23251	1026908	21541	21541	0.04	68	1007154	1007154	4.77	66
kroD100	21294*	976965*	22467	1069309	23833	1069309	22430	22430	0.05	70	1019821	1019821	4.39	72
kroE100	22068*	971266*	22960	1056228	23622	1056228	22964	22964	0.04	60	1034760	1034760	6.54	64
rd100	7910*	340047*	8381	380310	8778	365805	8333	8333	0.05	63	354762	354762	4.33	64
eil101	629*	27519*	681	28398	695	28398	662	662	0.05	62	27741	27741	0.81	61
aver									0.03				2.59	

'*' is the optimal values

Table 12. Comparison between our results with the others
(H.B. Ban et al.,2022; E. Osaba et al.,2020; Y. Yuan et al.,2016) for TSPTW and TRPTW

instances	YA (Q. Xu et al.,2021)		OA (J. N. Tsitsiklis et al.,1992)		BA (Ban et al., 2022)		MFEA	
	TSPTW	TRPTW	TSPTW	TRPTW	TSPTW	TRPTW	TSPTW	TRPTW
n40w40.002	-	-	-	-	-	-	461	7104
n40w60.002	-	-	-	-	-	-	470	7247
n40w80.002	-	-	-	-	-	-	431	7123
n40w100.002	-	-	-	-	-	-	364	6693
n60w20.002	-	-	-	-	-	-	605	13996
n60w120.002	-	-	-	-	-	-	427	12525
n60w140.002	-	-	-	-	-	-	464	11810
n60w160.002	-	-	-	-	-	-	423	12719
n80w120.002	-	-	-	-	-	-	577	18383
n80w140.002	-	-	-	-	-	-	472	18208
n80w160.002	-	-	-	-	-	-	553	17200
n100w120.002	-	-	-	-	-	-	556	29818
n100w140.002	-	-	-	-	-	-	632	30087
n100w120.003	-	-	-	-	-	-	646	24473
n100w140.003	-	-	-	-	-	-	481	28791
n100w140.004	-	-	-	-	-	-	533	27990

Table 13. Comparison computational effort between single-tasking and multitasking

Type	TSPTW	TRPTW
	gap	*gap*
single-tasking (100 generations)	0.59	0.08
multi-tasking (100 generations)	0.56	-0.28

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. Note that: Y. Yuan et al. supported the source code of their algorithm in (Y. Yuan et al.,2016) while the dMFEA-II algorithm (E. Osaba et al.,2020) was implemented again by us. Tables 2 and 3 evaluate the efficiency of the proposed selection in MFEA. Tables from 4 to 8 compare the proposed MFEA with the known best or optimal solutions for the TSPTW, and TRPTW instances (H. Abeledo et al.,2013; H.B. Ban et al.,2017; H.B. Ban et al.,2021; Y. Dumas et al.,1995; G. Heilporna et al.,2010; R. F. Silva et al.,2010; J. W. Ohlmann et al.,2007; M. W. Salvesbergh et al.,1985; C. R. Reeves et al.,1999). In the Tables, the *KBS*, *OPT*, *Aver.Sol*, and *Best.Sol* columns are the best known, optimal, average, and best solution, respectively, while the *gap* column presents the difference between the best solution and the optimal one. Table 9 compares the MFEA with RVNS, OA (E. Osaba et al.,2020), and YA (Y. Yuan et al.,2016). In the TSP, the optimal solutions of the TSPLIB-instances are obtained by running Concord tool [4]. In the TRP, the optimal or best solutions are obtained from (H. Abeledo et al.,2013).

5.1 Evaluating the efficiency of selection

In this experiment, a new selection operator for the MFEA algorithm effectively balances knowledge transfer and diversity. Due to being too expensive in computation, we choose some instances to evaluate the efficiency of this operator. In Table 2, the column MFEA-NR is the results of the MFEA with the selection-based scalar-fitness only, while the column MFEA is the results of the MFEA with both scalar-fitness and diversity. The $diff[\%]$ column is the difference between the MFEA and MFEA-NR.

In Table 2, the MFEA outperforms the MFEA-NR in all cases. The selection operation that considers both scalar-fitness and diversity to pick parents is more effective than the one with scalar-fitness only. The proposed MFEA algorithm adopts a fitness-based criterion for effectively transferring elite genes between tasks. Furthermore, population diversity is important since it becomes a bottleneck against genetic knowledge transfer.

5.2 Evaluating efficiency of local search

In this experiment, the efficiency of local search is considered. We run two MFEA on the same selected instances. In Table 3, the MFEA-NLS column is the MFEA without local search, while the MFEA column is the MFEA with local search. The $diff[\%]$ column is the difference between the MFEA and MFEA-NLS.

In Table 3, the MFEA obtains much better solutions than the MFEA-NLS in all cases. It indicates that the ability to exploit good solution spaces from RVNS is effective. That means the combination between the MFEA and RNVS improves the solution quality.

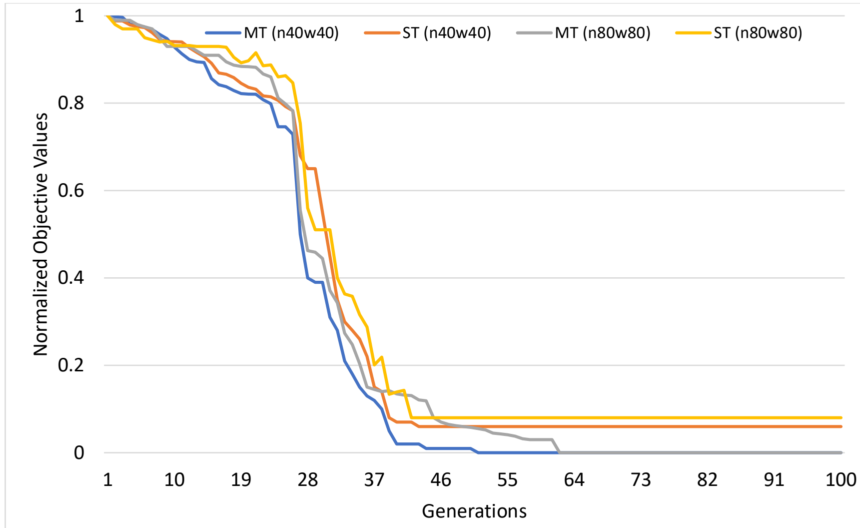
5.3 Comparisons with the TSPTW and TRPTW algorithms

The values of Table 9 are the average values of Tables from 4 to 7. The average difference with the optimal solution for the TSPTW is 0.56%, even for the instances with up to 200 vertices. It shows that our solutions are near-optimal for the TSPTW. In addition, the proposed algorithm reaches the optimal solutions for the instances with up to 80 vertices for the TSPTW. In Table 8, for the TRPTW, our MFEA is better than the previous algorithms such as Ban et al. (H.B. Ban et al.,2017; H.B. Ban et al.,2021), and G. Heilporna (G. Heilporna et al.,2010) in the literature when the average *gap* is -0.28% (note that: Ban et al.'s and G. Heilporna's et al.'s algorithms is developed to solve the TRPTW only). The obtained results are impressive since it can be observed that the proposed algorithm finds not only near-optimal solutions but also the new best-known ones for two problems simultaneously. It also indicates the efficiency of positive transferrable knowledge control techniques between optimization tasks in improving the solution quality.

It is impossible to expect that the MFEA always outperform in comparison with the state-of-the-art metaheuristic algorithms for the TSPTW and TRPTW in all cases because their algorithms are designed to solve each problem independently. Table 10 shows that the efficient algorithms for the TSPTW may not be effective for the TRPTW and vice versa. On average, the optimal solution for the TSPTW with the TRPTW objective cost is about 9.7% worse than the optimal one for the TRPTW. Similarly, the known best solution for the TRPTW using the TSPTW objective function is 20.6% worse than the optimal solution for the TSPTW. We conclude that if the proposed MFEA simultaneously reaches the good

[4] https://www.math.uwaterloo.ca/tsp/concorde.html

Figure 4. Comparing convergence trends of f_1 in multi-tasking and single-tasking for the TSPTW in n40w40 and n80w80 instances



solutions that are close to the optimal solutions for both problems and even better than the state-of-the-art algorithms in many cases, we can say that the proposed MFEA with RVNS for multitasking is beneficial.

5.4 Comparison with the previous MFEA algorithms

In the experiment, we adopt the proposed algorithm to solve the TSP and TRP problems. Otherwise, we also use three algorithms (H.B. Ban et al.,2022; E. Osaba et al.,2020; Y. Yuan et al.,2016) to solve the TSPTW and TRPTW.

Table 11 compares our results to those of three algorithms (H.B. Ban et al.,2022; E. Osaba et al.,2020; Y. Yuan et al., for some instances in both the TSP and TRP problems. The results show that the proposed algorithm obtains better solutions than the others in all cases. The difference between our average result and the optimal value is below 2.59%. Obviously, our solution is very near optimal one. In addition, our algorithm reaches the optimal solution for the instance with 76 vertices. Obviously, the proposed algorithm applies well in the case of the TSP and TRP.

Table 12 compares our results to those of three algorithms (H.B. Ban et al.,2022; E. Osaba et al.,2020; Y. Yuan et al., in both of the TSPTW and TRPTW problems. The results show that three algorithms cannot find feasible solutions in most cases while the proposed algorithm reaches feasible ones in all cases. It is understandable because these algorithms drives the search to solution spaces that maybe not contain feasible solutions. Otherwise, the proposed algorithm guides the search process to feasible solution spaces. It is an important contribution because finding a feasible solution for the TSPTW and TRPTW is even NP-hard (H.B. Ban et al.,2021).

5.5 Convergence trend

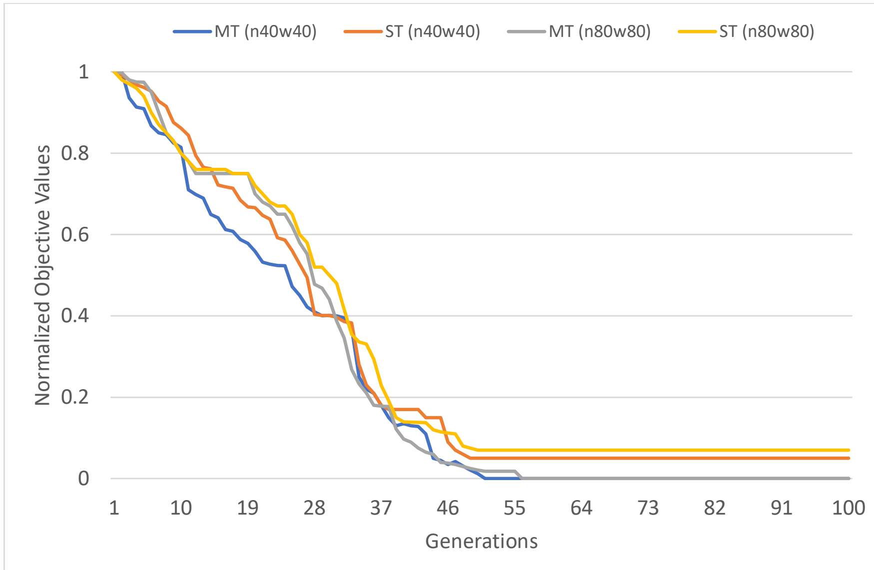
The normalized objective cost can be used to analyze the convergence trends of our MFEA algorithm:

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

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

The convergence trend of the two strategies is described in Figures 4 and 5 for n40w40 and n80w80 instances. The x-axis describes the number of generations, while the y-axis illustrates the normalized

Figure 5. Comparing convergence trends of f_2 in multi-tasking and single-tasking for the TRPTW in n40w40 and n80w80 instances



objective value. The less and less the normalized objective value is, the better and better the algorithm is. In Figures 4 and 5, Single-tasking (ST) converges better than multitasking (MT) in the whole evolutionary process while avoiding premature convergence to sub-optimal solutions by exchanging knowledge among tasks. That means, in general, multitasking converges to a better objective value.

When the multitasking is run with the same number of generations as single-tasking, on average, it only consumes $\frac{1}{K}$ computational effort for each task (K is the number of tasks). Therefore, we consider the worst-case situation when the number of generations for multitasking is K times the one for single-tasking. If multitasking obtains better solutions than single-tasking in this case, we can say that multitasking saves computational efforts. The experimental results are described in Table 13. In Table 13, the first row shows the average gap of single-task for the TSPTW and TRPTW, while the second shows the average gap of multitasking with 100 generations. The result shows that the multitasking consumes only $\frac{1}{2}$ computational efforts to obtain better solutions than the single-tasking.

In short, the efficiency of the multitasking is better in comparison with the single-tasking because the process of transferring knowledge during multitasking. It is the impressive advantage of the evolutionary multitasking paradigm.

5.6 Discussions

The MFEA framework (S. Dash et al.,2012; A. Gupta et al.,2016; E. Osaba et al.,2020; Y. Yuan et al.,2016; Q. Xu et al has been proposed to incorporate Evolutionary into multitasking to handle multiple problems at the same time. Instead of solving a pool of similar optimization problems independently, it performs multiple tasks for systems. Therefore, it can be useful in a system with limited computation. In addition, the advantage of the approach compared with single-task EA is that the phenomenon of genetic information transfer in multitasking can exploit transferrable knowledge between optimization tasks. Therefore, it can find better solutions for multitasks. That is an important characteristic of the MEFA.

In current, some MFEA algorithms (A. Langevin et al.,1993; E. G. Talbi et al.,2009) proposed to solve the close variant of TSP and TRP simultaneously. However, the drawback of two algorithms (A. Langevin et al.,1993; E. G. Talbi et al.,2009) is that there is a lack of a mechanism to exploit the good solution space explored by MFEA. Therefore, these algorithms cannot balance exploration and exploitation effectively. Recently, Ban et al. (H.B. Ban et al.,2022) have applied the MFEA with RVNS to solve the TSP and TRP successfully. Its performance encourages us to use the combination to solve the different variants such as the TSPTW and TRPTW.

The TSPTW and TRPTW are combinatorial optimization problems that have many practical situations. Currently, there exist many algorithms that are proposed to solve them. We can find the TSPTW's and TRPTW's algorithms in the literature and run them parallel. However, these algorithms are designed to solve each problem independently and separately. They do not use a unified search space. Therefore, they cannot exploit positive transferrable knowledge between optimization tasks. In addition, running two algorithms in parallel can require a strong enough computation system while our MFEA runs sequentially. The MFEA is suitable in a system with limited computation. This paper introduces the first algorithm that combines the MFEA framework and RVNS for solving two tasks simultaneously. The combination is to have positive transferrable knowledge between tasks from the MFEA and the ability to exploit good solution spaces from RVNS. Due to the important characteristics, finding better solutions will be increased. In comparison with the previous schemes (H.B. Ban et al.,2022; A. Langevin et al.,1993; E. G. Talbi et al.,2009), our scheme includes new contributions as follows:

- We propose a new selection operator that balances skill-factor and population diversity. The skill-factor effectively transfers elite genes between tasks, while diversity in the population is important when it meets a bottleneck against the information transfer.
- Multiple crossover schemes are applied in the proposed MFEA. They help the algorithm have good enough diversity. In addition, two types of crossover (intra- and inter-) are used. It opens up the chance for knowledge transfer through crossover-based exchange between tasks.
- The combination between the MFEA and the RVNS is to have good transferrable knowledge between tasks from the MFEA and the ability to exploit good solution spaces from the RVNS. However, focusing only on reducing cost function maybe lead the search to infeasible solution spaces like the algorithm (H.B. Ban et al.,2022). Therefore, the repair method is incorporated into the proposed algorithm to balance finding feasible solution spaces and reducing cost function.

6 CONCLUSIONS AND FUTURE WORK

In the paper, we propose an effective algorithm based on the MFEA framework for simultaneously solving the TSPTW and TRPTW, which combines the MFEA and Randomized RVNS. Extensive experiments on benchmark dataset indicate that the proposed algorithm solves both problems well at the same time. In addition, it obtains better solutions than the previous MFEA algorithms in many cases. More interestingly, it finds the new best-known solutions compared to the state-of-the-art metaheuristics only for the TRPTW in many cases. Finally, it indicates the efficiency of transferrable knowledge between optimization tasks in the MFEA framework.

In future work, we will study how to apply multiple population ideas for multitasking. Many researchers is interested in the approach (Y. Chen et al.,2017; G. Li et al.,2019; T. Wei et al.,2009; T. Wei et al.,2009). The approach's advantages include: 1) each population evolves with different genetic operators, and each individual can be represented differently; 2) individuals migrate between populations. The approach maintains the diversity and improves convergence trends.

REFERENCES

[H. Abeledo et al.,2013] H. Abeledo, R. Fukasawa, A. Pessoa, and E. Uchoa, The time dependent traveling salesman problem: polyhedra and algorithm, J. Mathematical Programming Computation, 5, 2013, pp. 27-55.

[M. Angelova et al.,2011] M. Angelova, T. Pencheva, Tuning genetic algorithm parameters to improve convergence time. Int. J. Chem. Eng, 2011, pp. 1-7.

[H.B. Ban et al.,2017] H.B. Ban, N.D. Nghia, Metaheuristic for the Traveling Repairman Problem with Time Windows, Proc. RIVF, 2017, pp.1-6.

[H.B. Ban et al.,2021] H.B. Ban, A metaheuristic for the delivery man problem with time windows, J. Joco, 41 (4), 2021, pp. 794-816.

[H.B. Ban et al.,2022] H.B. Ban, H.D. Pham. Multifactorial Evolutionary Algorithm for Simultaneous Solution of TSP and TRP, J CAI, 40 (6), 2022, pp. 1370–1397.

[T. Chena et al.,2017] T. Chena, K. Tang, G. Chen, X. Yao, A large population size can be unhelpful in evolutionary algorithms, J. Theoretical Computer Science, 436, 2012, pp. 54-70.

[Y. Chen et al.,2017] Y. Chen, J. Zhong, L. Feng, and J. Zhang, An adaptive archive-based evolutionary framework for many-task optimization, J. IEEE Trans. onEmerg, 4 (3), pp. 369-384, 2020.

[S. Dash et al.,2012] S. Dash, O. Günlük, A. Lodi, and A. Tramontani. A Time Bucket Formulation for the Traveling Salesman Problem with Time Windows, INFORMS Journal on Computing, 24, pp 132-147, 2012.

[Y. Dumas et al.,1995] Y. Dumas, J. Desrosiers, E. Gélinas, An optimal algorithm for the Traveling Salesman Problem with Time Windows, J. Operations Research, 43, 1995, pp. 367-371.

[T. A. Feo et al.,1995] T. A. Feo, and M.G.C. Resende, Greedy Randomized Adaptive Search Procedures, J. Global Opt., 1995, pp. 109-133.

[F. Focacci et al.,2002] F. Focacci, A. Lodi, M. Milano, A hybrid exact algorithm for the TSPTW, INFORMS Journal on Computing, 14 (4), 2002, pp. 403-417.

[M. Gendreau et al.,1998] M. Gendreau, A. Hertz, G. Laporte, M. Stan, A generalized insertion heuristic for the traveling salesman problem with time windows, J. Operations Research, 43, 1998, pp. 330–335.

[A. Gupta et al.,2016] A. Gupta, Y.S. Ong, L. Feng, Multifactorial evolution: toward evolutionary multitasking, J. IEEE Trans Evol Comput, 20(3), pp. 343–357, 2016.

[G. Heilporna et al.,2010] G. Heilporna, Jean-François Cordeaua, and Gilbert Laporte, The Delivery Man Problem with time windows, 7, 2010, pp. 269-282.

[T. Ibaraki et al.,2008] T. Ibaraki, S. Imahori, K. Nonobe, K. Sobue, T. Uno, and M. Yagiura, An iterated local search algorithm for the vehicle routing problem with convex time penalty functions, J. Discrete Applied Mathematics, 11 (156), 2008, pp. 2050–2069.

[A. Langevin et al.,1993] A. Langevin, M. Desrochers, J. Desrosiers, Sylvie Gélinas, and F. Soumis. A two-commodity flow formulation for the traveling salesman and makespan problems with time windows. Networks, 23(7), 1993, pp. 631-640.

[Y. Lavinas et al.,1993] Y. Lavinas, C. Aranha, T. Sakurai, and Marcelo Ladeira, Experimental Analysis of the Tournament Size on Genetic Algorithms, Proc. SMC, 2019, pp. 3647–3653.

[Y. C. Lian et al.,2019] Y. C. Lian, Z. X. Huang, Y. R. Zhou, Z. F. Chen, Improve theoretical upper bound of Jumpk function by evolutionary multitasking, Proc. HPCCT, pp. 22–24, 2019, pp. 44–50.

[G. Li et al.,2019] G. Li, Q. Zhang, and W. Gao, Multipopulation evolution frame workfor multifactorial optimization, Proc. GECCO, 2018, pp. 215–216.

[O. Martin et al.,1991] O. Martin, S. W. Otto, E. W. Felten, Large-step Markov Chains for the Traveling Salesman Problem, J. Complex Systems, 5 (3), 1991, pp. 299–326.

[E. Osaba et al.,2020] E. Osaba, A.D. Martinez, A. Galvez, A. Iglesias, J. Del Ser, dMFEA-II: An Adaptive Multifactorial Evolutionary Algorithm for Permutation-based Discrete Optimization Problems, Proc. GECCO, pp. 1690–1696, 2020.

[A. Otman et al.,2015] A. Otman, and A. Jaafar, A Comparative Study of Adaptive Crossove for Genetic Algorithms to Resolve the Traveling Salesman Problem, J. Computer Applications, 31 (11), pp. 49-57, 2011.

[N. Mladenovic et al.,1997] N. Mladenovic, P. Hansen, Variable Neighborhood Search, J. Operations Research, 24 (11), 1997, pp.1097-1100.

[R. F. Silva et al.,2010] R. F. Silva, S. Urrutia, A General VNS heuristic for the traveling salesman problem with time windows, J. Discrete Optimization, 7 (4), 2010, pp. 203-211.

[D. S. Johnson et al.,1995] D. S. Johnson, and L. A. McGeoch, The traveling salesman problem: A case study in local optimization in Local Search in Combinatorial Optimization, E. Aarts and J. K. Lenstra, eds., 1995, pp. 215-310.

[J. N. Tsitsiklis et al.,1992] J. N. Tsitsiklis, Special cases of Traveling Salesman and Repairman Problems with time windows, J. Networks, 22, 1992, pp. 263–283.

[E. G. Talbi et al.,2009] E. G. Talbi, Metaheuristics: from design to implementation, NewJersey, Wiley, 2009.

[T. Wei et al.,2009] T. Wei and J. Zhong, “A preliminary study of knowledge transfer inmulti-classification using gene expression programming,” J. Frontiers in Neuroscience, Vol. 13, p. 1-14, 2020.

[T. Wei et al.,2009] T. Wei, S. Wang, J. Zhong, D. Liu, A Review on Evolutionary Multi-Task Optimization: Trends and Challenges, J. IEEE Transactions on Evolutionary Computation, pp 1-20, DOI: 10.1109/TEVC.2021.3139437.

[J. W. Ohlmann et al.,2007] J. W. Ohlmann and B. W. Thomas, A Compressed-annealing Heuristic for the

Traveling Salesman Problem with time windows, J. INFORMS, 19 (1), pp. 80-90, 2007.

[M. W. Salvesbergh et al.,1985] M. W. Salvesbergh, Local search in routing problems with time windows, J. Annals of Operations Research, 4, 1985, pp. 285-305

[C. R. Reeves et al.,1999] C. R. Reeves, Landscapes, operators and heuristic search, Annals of Operations Research 86(0), 1999, pp. 473-490.

[Y. Yuan et al.,2016] Y. Yuan, Y.S. Ong, A. Gupta, P.S. Tan, H. Xu, Evolutionary multitasking in permutation-based combinatorial optimization problems: realization with tsp, qap, lop, and jsp, Proc. TENCON, 2016, pp. 3157-3164.

[Q. Xu et al.,2021] Q. Xu, N. Wang, L. Wang, W. Li, and Q. Sun, “Multi-Task Optimization and Multi-Task Evolutionary Computation in the Past Five Years: A Brief Review, J. Mathematics, 9 (864), 2021, pp. 1-44.

Figure 1

Figure 5

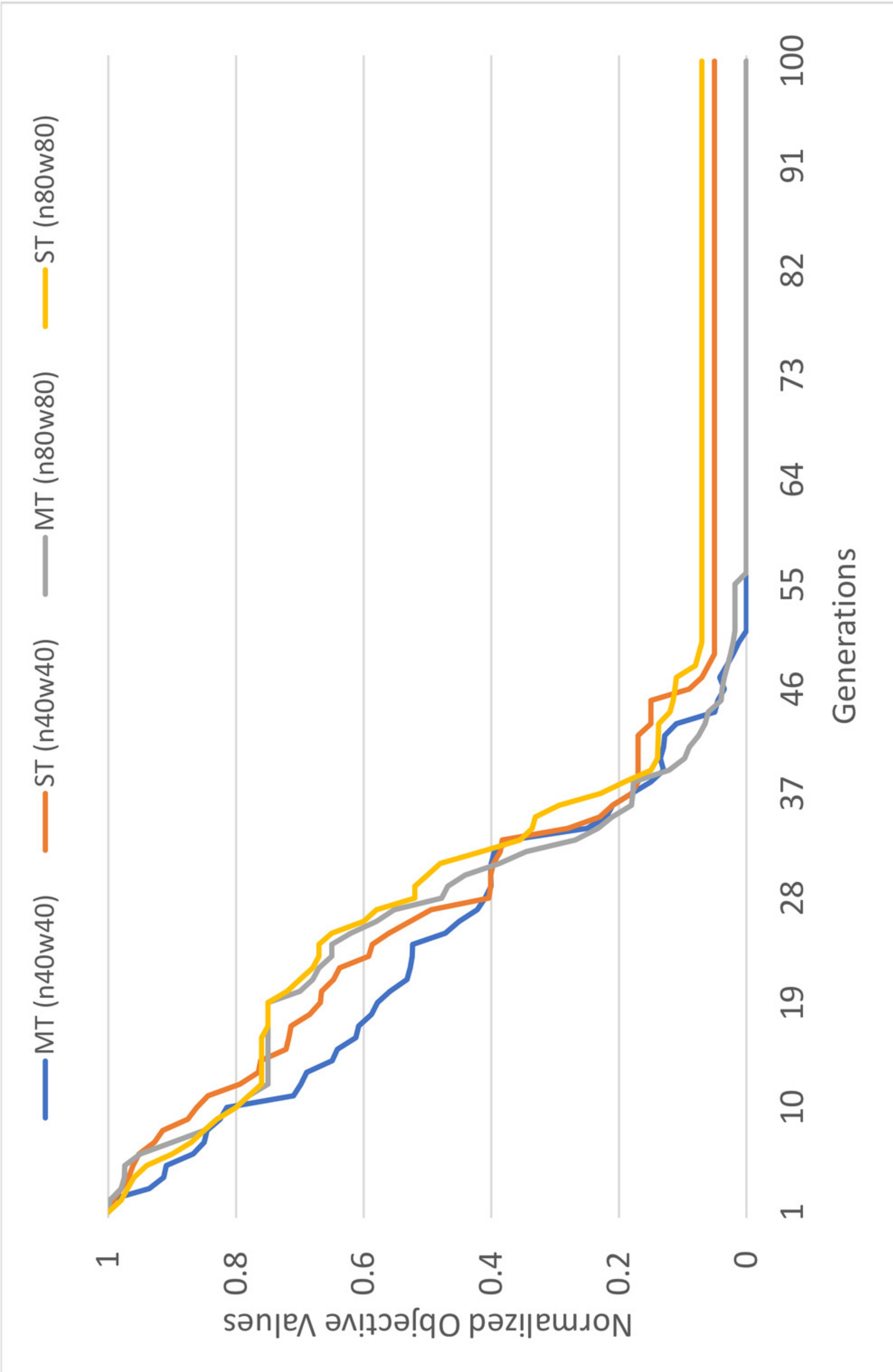

MT (n40w40)
ST (n40w40)
MT (n80w80)
ST (n80w80)
Normalized Objective Values
1
0.8
0.6
0.4
0.2
0
Generations
1
10
19
28
37
46
55
64
73
82
91
100

Figure 2

Figure 3

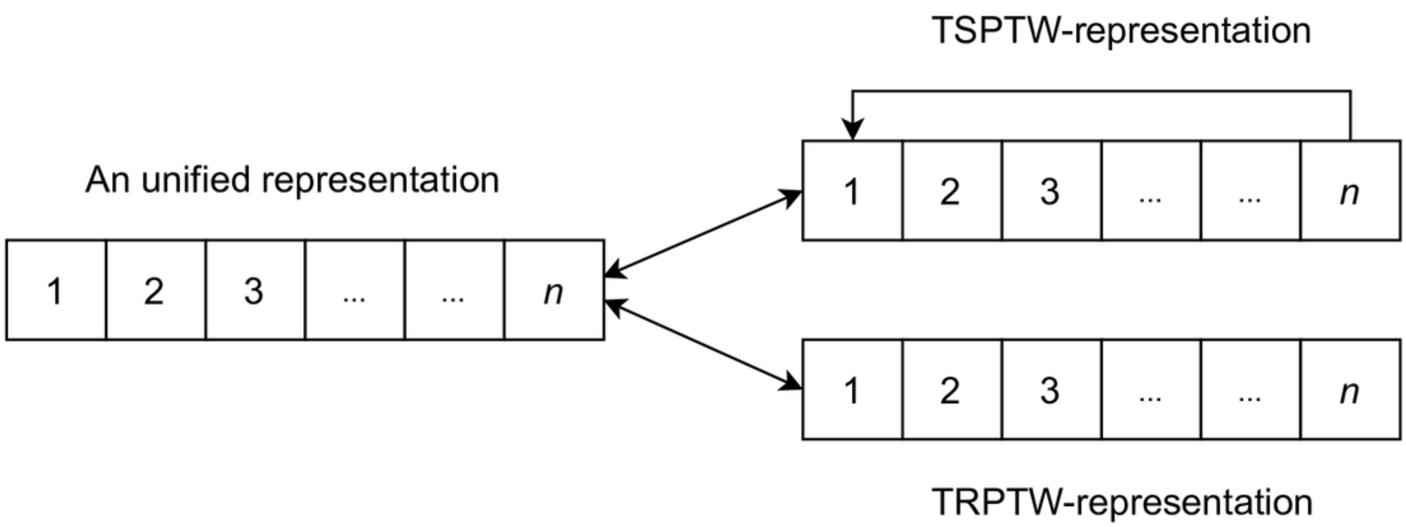


Figure 3

Figure 4

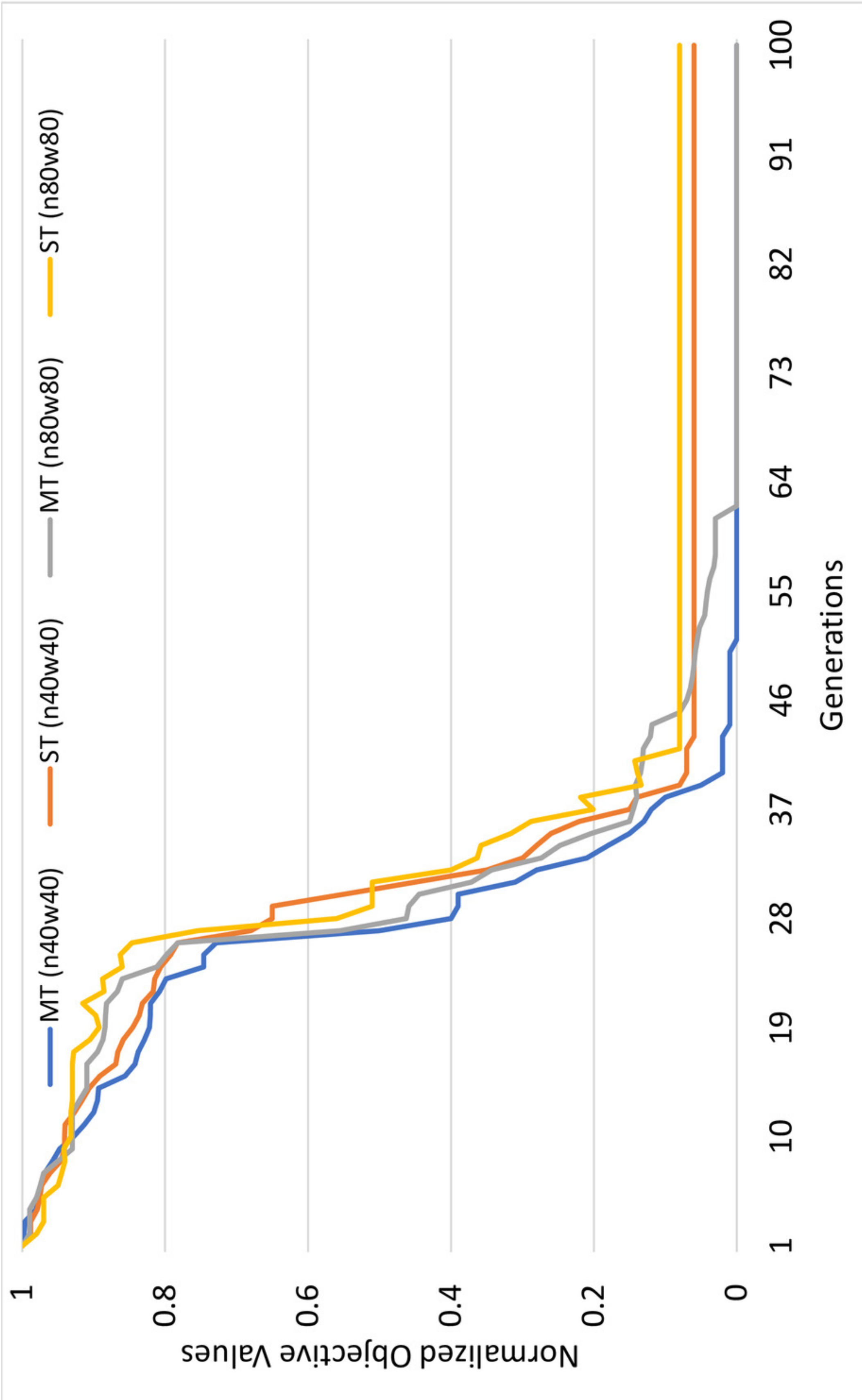

MT (n40w40)
ST (n40w40)
MT (n80w80)
ST (n80w80)
Normalized Objective Values
1
0.8
0.6
0.4
0.2
0
Generations
1
10
19
28
37
46
55
64
73
82
91
100

Figure 4

Figure 2

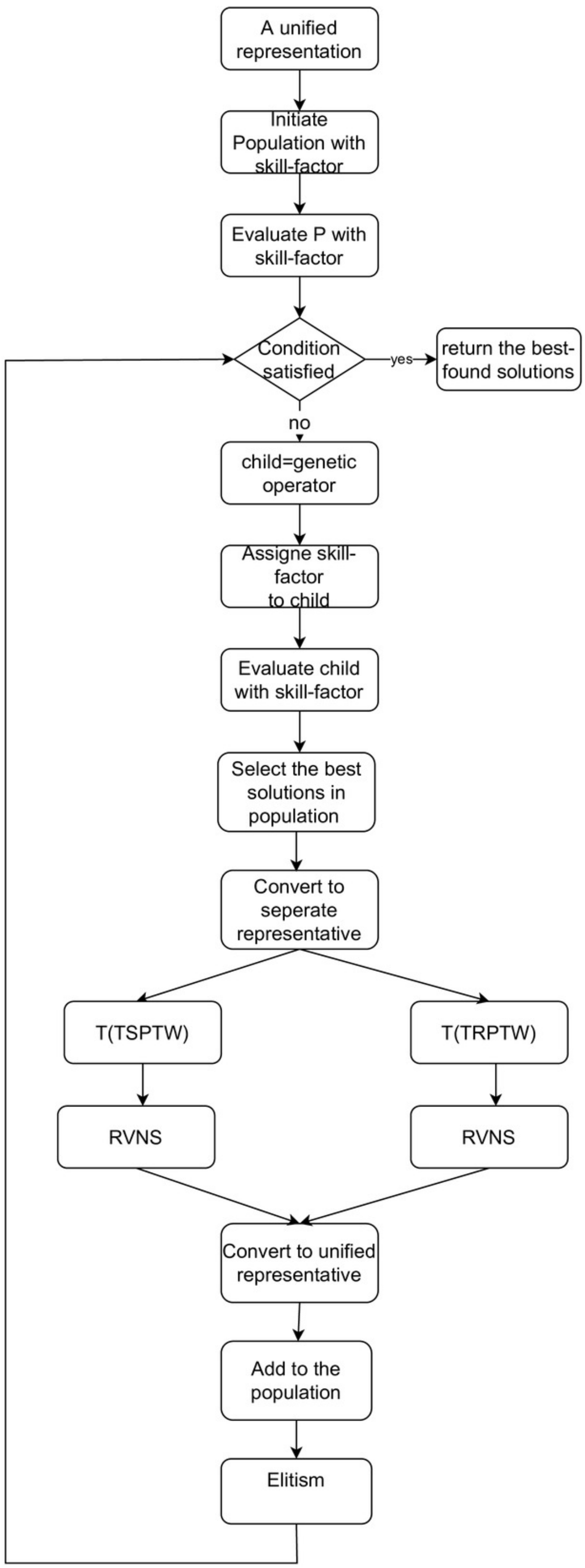

A unified representation
Initiate Population with skill-factor
Evaluate P with skill-factor
Condition satisfied
yes
return the best-found solutions
no
child=genetic operator
Assigne skill-factor to child
Evaluate child with skill-factor
Select the best solutions in population
Convert to seperate representative
T(TSPTW)
T(TRPTW)
RVNS
RVNS
Convert to unified representative
Add to the population
Elitism

Figure 5

Figure 1


A representation
Initiate Population with fitness
Evaluate P with fitness
Condition satisfied
yes
return the best solutions
no
child=genetic operator
Evaluate child with fitness
Add child to P
Elitist (P)
A unified representation
Initiate Population with skill-factor
Evaluate P with skill-factor
Condition satisfied
yes
return the best solutions
no
child=genetic operator
Assigne skill-factor to child
Evaluate child with skill-factor
Add child to P
Elitist (P)