This paper has been published as "Y. Nan, K. Shang, H. Ishibuchi and L. He, “Reverse Strategy for Non-Dominated Archiving,” IEEE Access, vol. 8, pp. 119458-119469, Jun. 2020. DOI: 10.1109/ACCESS.2020.3005970"

Digital Object Identifier 10.1109/ACCESS.2020.DOI

Reverse Strategy for Non-dominated Archiving

YANG NAN, KE SHANG, HISAO ISHIBUCHI, (Fellow, IEEE), AND LINJUN HE
Guangdong Provincial Key Laboratory of Brain-inspired Intelligent Computation, Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China.

Corresponding author: Hisao Ishibuchi (e-mail: hisao@sustech.edu.cn).

This work was supported by National Natural Science Foundation of China (Grant No. 61876075), Guangdong Provincial Key Laboratory (Grant No. 2020B121201001), the Program for Guangdong Introducing Innovative and Enterpreneurial Teams (Grant No. 2017ZT07X386), Shenzhen Science and Technology Program (Grant No. KQTD2016112514355531), the Program for University Key Laboratory of Guangdong Province (Grant No. 2017KSYS008).

ABSTRACT In the field of evolutionary multi-objective optimization (EMO), most EMO algorithms try to find a set of non-dominated solutions to approximate the Pareto front of a multi-objective optimization problem. In these algorithms, a population is evolved from one generation to another, and the population of the last generation is presented as the final result. However, recent studies reveal that some good solutions can be discarded during the evolutionary process, whereas these solutions are non-dominated. One way to solve this issue is to store all non-dominated solutions in an unbounded external archive (UEA) during the evolutionary process and select a set of solutions from the UEA as the final result. A recently proposed ND-Tree approach is very efficient for updating the UEA whenever a new solution is generated. However, this may not be the most efficient strategy. In this paper, we propose a simple yet efficient update strategy for the ND-Tree approach. The main idea is to reverse the order of solutions with respect to their generated time when updating the UEA. The experimental results suggest that the ND-Tree approach assisted by the proposed reverse strategy is much faster than the original ND-Tree approach in obtaining the final UEA. The optimal update frequency for the proposed strategy is also investigated.

INDEX TERMS ND-Tree, Evolutionary multi-objective optimization, Unbounded external archive (UEA).

I. INTRODUCTION

Evolutionary multi-objective optimization (EMO) has been a topic of interest for the last three decades. A multi-objective optimization problem (MOP) usually has multiple objectives that conflict with each other. Due to this conflicting nature, there is no single optimal solution performing the best on all objectives. Instead, a set of optimal solutions is usually obtained to show the trade-off among the conflicting objectives.

Many multi-objective evolutionary algorithms (MOEAs) have been proposed aiming to find a set of optimal solutions in a single run. In general, these algorithms can be classified into three categories: dominance-based algorithms (e.g., NSGA-II [1], B-NSGA-III [2] and U-NSGA-III [3]), decomposition-based algorithms (e.g., MOEA/D [4] and MOEA/D-AWA [5]), and indicator-based algorithms (e.g., IBEA [6] and HypE [7]). Recently, some researchers view multi-population-based algorithms and coevolution-based algorithms as the fourth category, and many other MOEAs for specific MOPs were proposed. For instance, Chen et al. proposed a dynamic constrained MOEA (dCMOEA) to solve dynamic constrained MOPs [8], Fu et al. developed a hybrid MOEA (HMOEA) to solve hybrid flow shop scheduling problems [9], and Fu et al. proposed a multi-objective brain storm optimization (MOBSO) algorithm to solve stochastic multi-objective distributed permutation flow shop scheduling problems [10].

Most of these algorithms maintain a population (i.e., an internal archive) with a bounded size. However, as investigated in [11], most of the algorithms fail to preserve the best set of solutions in the bounded final population. Good solutions can be discarded during the evolutionary process. To address this issue, many researchers proposed to use an external archive to assist the evolutionary process. For example, Deb et al. proposed to select one of the parents from an external archive [12], whereas both parents in most of the existing algorithms are selected from the main population. In [13], an external archive serves as a guider to guide the search in the internal population by allocating the computational

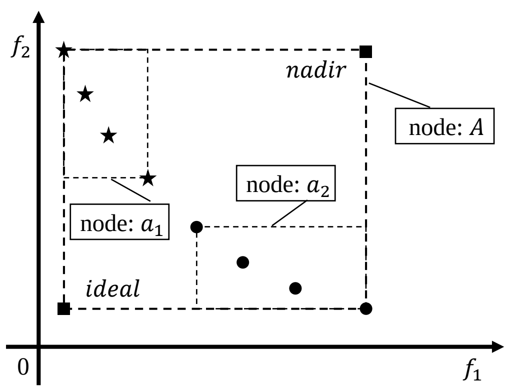

FIGURE 1. Illustration of the nodes of an ND-Tree. Node A (bounded by the ideal and nadir points) is the parent node of child node a_1 and child node a_2. The solution set of each child node has four solutions. The solution set of the parent node A is the union of the solution sets of the child nodes a_1 and a_2.

resource for each subproblem. The subproblems that can generate good solutions (i.e., solutions that can enter the external archive) are more likely to be selected at the next generation. The benefit of the bounded external archive used in [12], [13] highly relies on the quality of the solutions in the external archive. However, as reported in [14]–[16], the performance of an external archive deteriorates when it is bounded. Some algorithms proposed to use unbounded external archives (UEAs) to promote the evolutionary process [14], [17]. However, maintaining the UEA is a very time-consuming process since the size of the UEA can be very large, especially in high-dimensional objective spaces.

To maintain the UEA during the evolutionary process efficiently, several update approaches were proposed [14], [18]. The non-dominated tree (ND-Tree) approach is a recently proposed update approach [18]. It partitions the objective space into several hyper-rectangles according to the solutions in the UEA. These hyper-rectangles are used to construct an ND-Tree that can significantly reduce a vast number of unnecessary solution comparisons during the UEA updating process.

However, in some cases, we need the UEA after the execution of an EMO algorithm (not during its execution) as the final output of the algorithm [19]. As mentioned before, many researchers focus on updating the UEA during the evolutionary process, and all these approaches update the UEA right after a new solution is generated. Intuitively, this is not as effective since the data structures to store the non-dominated solutions are reconstructed frequently. In this paper, we propose a reverse strategy, which can be applied to the ND-Tree approach to make it more efficient when selecting the non-dominated solutions from all the examined solutions at the end of the evolutionary process.

In this paper, we make the following two contributions:

- We propose a reverse strategy for the ND-Tree approach, which can significantly reduce its time cost when it is used to select all non-dominated solutions at the end of the evolutionary process. The ND-Tree approach with the reverse strategy is called the ND-Tree-Reverse approach.
- To obtain the non-dominated solutions from all the examined solutions, a straightforward method is to obtain the UEA with the ND-Tree-Reverse approach at the end of the evolutionary process. Another method is to update the UEA by the ND-Tree-Reverse approach every g generations (e.g., every 10 generations). We investigate the optimal frequency of updating the UEA during the evolutionary process using the ND-Tree-Reverse approach.

The remainder of this paper is organized as follows. In Section II, basic knowledge of the ND-Tree approach is introduced. The reverse strategy is proposed in Section III. In Section IV, experimental results are shown to validate the effectiveness of our strategy. The optimal update frequency is investigated in Section V. Finally, we conclude the paper in Section VI.

II. ND-TREE APPROACH

The ND-Tree approach was initially proposed in [18]. In this section, we briefly explain this approach since it is the foundation of our proposed reverse strategy. The ND-Tree approach uses a tree structure (ND-Tree) to store all non-dominated solutions generated during the evolutionary process. When a new solution is generated, the new solution is first compared with all solutions in the ND-Tree. If the new solution is dominated, it is simply rejected. Otherwise, the new solution is inserted into the tree, and the solutions that are dominated by the new solution are removed from the ND-Tree.

The ND-Tree is constructed based on nodes. Each node indicates a hyper-rectangle in the objective space. The hyper-rectangle is specified by the ideal and nadir points of the solutions in this node. The solutions in the hyper-rectangle form the solution set ($\mathcal{L}$) of this node. Each node has no more than B (a predefined size) child nodes (branches). A node is called a leaf node if it has no child node; otherwise, it is called a branch node. If a node has no parent node, it is a root node.

In Figure 1, the four stars constitute the solution set $\mathcal{L}$ of node a_1 ($a_1.\mathcal{L}$), and the four points constitute the solution set $\mathcal{L}$ of node a_2 ($a_2.\mathcal{L}$). The child nodes (i.e., nodes a_1 and a_2) of node A are the hyper-rectangles included in the hyper-rectangle of node A. The union of $a_1.\mathcal{L}$ and $a_2.\mathcal{L}$ is the solution set $\mathcal{L}$ of node A (i.e., $A.\mathcal{L}$).

A. BASIC CONCEPTS IN ND-TREE

Throughout this paper, the minimization of all objectives is assumed. That is, the ideal and nadir points of the solution set $\mathcal{L}$ are defined as follows:

$$\begin{aligned} ideal_i &= \min_{f\in\mathcal{L}} f_i, \quad i \in \{1, ..., M\}, \\ nadir_i &= \max_{f\in\mathcal{L}} f_i, \quad i \in \{1, ..., M\}, \end{aligned} \tag{1}$$

where $ideal_i$ and $nadir_i$ are the i-th elements of the ideal and nadir points, respectively, and M is the number of objectives.

Algorithm 1: ND-Tree

```
input  : $S$ (a set of $N$ examined solutions)
output: $root$ (root of ND-Tree)
begin
1   initialize $root$ with $s_1$
2   for $i \in \{2, ..., N\}$ do
3       $is\_dominated \longleftarrow$ update$(root, s_i)$
4       if $root$ is deleted then
5           initialize $root$ with $s_i$
6       else if not $is\_dominated$ then
7           insert$(root, s_i)$
```

In Figure 1, the ideal and nadir points of node A are the lower-left square and the upper-right square, respectively.

The main idea of the ND-Tree approach is to utilize the tree structure and the following three properties to avoid a massive number of unnecessary solution comparisons [18]:

- $Property$ 1: If a new solution is dominated by the nadir point of a node, it is dominated by all solutions in $\mathcal{L}$ of the node. This property is straightforward because the nadir point of the node is dominated by all solutions in $\mathcal{L}$ of the node.
- $Property$ 2: If a new solution dominates the ideal point of a node, it dominates all solutions in $\mathcal{L}$ of the node.
- $Property$ 3: If a new solution is incomparable to the ideal and nadir points of a node, the new solution is incomparable to all solutions in $\mathcal{L}$ of the node. For the detailed proof, refer to [18].

B. DETAILS OF ND-TREE

The main process of the ND-Tree approach is shown in Algorithm 1. The input is a solution set (S) storing N examined solutions (i.e., $s_1, ..., s_N$). First, a new node $root$ is initialized with the first generated solution s_1. That is, for node $root$, its solution set ($root.\mathcal{L}$), ideal point ($root.ideal$), and nadir point ($root.nadir$) are initialized with s_1. Then, node $root$ is updated with the other solutions (i.e., $s_2, ..., s_N$) one by one. For the i-th solution s_i, if it is not dominated by any solutions in the ND-Tree, s_i is inserted into the ND-Tree. It is worth noting that if $root$ is deleted by s_i, a new node $root$ is initialized with s_i.

The purposes of update (line 3 of Algorithm 1) are (1) to indicate if a new solution (p) is dominated by any solutions in a node (n) and (2) to delete the solutions that are dominated by the solution p from $n.\mathcal{L}$. In Algorithm 2, $is\_dominated$ indicates whether the solution p is dominated by any solutions in $n.\mathcal{L}$. First, if the solution p is dominated by $n.nadir$, $is\_dominated$ = True is returned ($Property$ 1). If the solution p dominates $n.ideal$, node n and its all descendant nodes are deleted, and $is\_dominated$ = False is returned ($Property$ 2). If the solution p is not dominated by $n.ideal$ and p does not dominate $n.nadir$, $is\_dominated$ = False is returned ($Property$ 3).

Algorithm 2: update(n, p)

```
input  : $n$ (a node), $p$ (a solution)
output: $is\_dominated$ ($p$ is dominated or not)
begin
1   if $n.nadir \prec p$ then
2       $is\_dominated \longleftarrow$ True
3   else if $p \prec n.ideal$ then
4       delete $n$ and its all descendant nodes
5       $is\_dominated \longleftarrow$ False
6   else if $n.ideal \prec p$ or $p \prec n.nadir$ then
7       $is\_dominated \longleftarrow$ False
8   else
9       if $n$ is a $branch\ node$ then
10          for $b \in n.branch$ do
11              $is\_dominated \longleftarrow$ update$(b, p)$
12              if $is\_dominated$ then
13                  break
14      else if $n$ is a $leaf\ node$ then
15          for $q \in n.\mathcal{L}$ do
16              if $q \prec p$ then
17                  $is\_dominated \longleftarrow$ True
18                  break
19              else if $p \prec q$ then
20                  $is\_dominated \longleftarrow$ False
21                  $n.\mathcal{L} \longleftarrow n.\mathcal{L} \setminus \{q\}$
```

If the above conditions are not met and node n is a branch node, update is recursively performed for all child nodes of node n (lines 10-13 of Algorithm 2). If the solution p is dominated by at least one child node of node n, $is\_dominated$ = True is returned. If node n has only a child node left after updating all child nodes of n with the solution p, node n is replaced by this child node.

If node n is a leaf node, all solutions in $n.\mathcal{L}$ are compared with the solution p one by one (lines 15-21 of Algorithm 2). If p is dominated by any solutions in $n.\mathcal{L}$, $is\_dominated$ = True is returned. If the solution p is not dominated by any solutions in $n.\mathcal{L}$, the solutions in $n.\mathcal{L}$ dominated by p are deleted. It is worth noting that if all solutions in $n.\mathcal{L}$ are deleted, node n is also deleted.

If the new solution p is not dominated by any solutions stored in the ND-Tree (i.e., $is\_dominated$ = False), p is inserted into the ND-Tree. In Algorithm 3, the inputs of insert are a solution p and a node n. If node n is a branch node, solution p is inserted into its nearest child node n' of node n (lines 2-9). In line 3, dist calculates the average distance between the solution p and all solutions in $n.branch[i].\mathcal{L}$. If node n is a leaf node, solution p is added to $n.\mathcal{L}$ and the ideal and nadir points of node n are updated with the solution p. If the size of $n.\mathcal{L}$ is larger than a predefined size (C), node n is

```
Algorithm 3: insert($n, p$)
input : $n$ (a node of ND-Tree), $p$ (a candidate
        solution)
begin
1   if $n$ is a $branch\ node$ then
2       $min\_index \longleftarrow 1$
3       $min\_dist \longleftarrow$ dist($p, n.branch[1].\mathcal{L}$)
4       for $i \in \{2, ..., |n.branch|\}$ do
5           $cur\_dist \longleftarrow$ dist($p, n.branch[i].\mathcal{L}$)
6           if $cur\_dist < min\_dist$ then
7               $min\_dist \longleftarrow cur\_dist$
8               $min\_index \longleftarrow i$
9       insert($n.branch[min\_index], p$)
10  else if $n$ is a $leaf\ node$ then
11      $n.\mathcal{L} \longleftarrow n.\mathcal{L} \bigcup \{p\}$
12      update_ideal_nadir($n, p$)
13      if $|n.\mathcal{L}| > C$ then
14          split($n$)
```

split by split.

In Algorithm 4, to split a node n, the solution set of n is first copied to a temporary solution set (TS). Then, the most distant solution from the other solutions is removed from TS. This removal is repeatedly performed B times, where B is a predefined number indicating the maximum number of the child nodes of a node. These B solutions removed from TS are used to initialize B new child nodes (lines 2-12). After that, each remaining solution in TS is added to the closest child node (i.e., one of the newly generated nodes). Finally, the ideal and nadir points of each child node are updated by update_ideal_nadir (lines 13-23).

Algorithm 5 shows how the ideal and nadir points are updated. Note that the two points of the ancestors of node n will also be updated if any of the two points of node n change.

The above processes show all the details of the ND-Tree approach. The main idea of this approach is to utilize the tree structure and the three properties mentioned in this section to avoid a number of unnecessary comparisons when updating the ND-Tree.

III. REVERSE STRATEGY FOR ND-TREE

The ND-Tree approach updates the UEA right after a new solution is generated. However, in some cases, we can first store all the solutions generated during the evolutionary process and then update the UEA using all the stored solutions at the end of the evolutionary process. In this section, we propose a reverse strategy for the ND-Tree approach and elaborate how our proposed strategy speeds up the ND-Tree approach.

```
Algorithm 4: split($n$)
input : $n$ (a node of ND-Tree)
begin
1   $TS \longleftarrow n.\mathcal{L}$
2   for $i \in \{1, ..., B\}$ do
3       $max\_index \longleftarrow 1$
4       $max\_dist \longleftarrow$ dist($n.\mathcal{L}[1], n.\mathcal{L}$)
5       for $j \in \{2, ..., |TS|\}$ do
6           $cur\_dist \longleftarrow$ dist($n.\mathcal{L}[j], n.\mathcal{L}$)
7           if $cur\_dist > max\_dist$ then
8               $max\_dist \longleftarrow cur\_dist$
9               $max\_index \longleftarrow j$
10      initialize a node $n'$ with $TS[max\_index]$
11      $n.branch \longleftarrow n.branch \bigcup n'$
12      $TS \longleftarrow TS \setminus \{TS[max\_index]\}$
13  for $i \in \{1, ..., |TS|\}$ do
14      $min\_index \longleftarrow 1$
15      $min\_dist \longleftarrow$ dist($TS[i], n.branch[1].\mathcal{L}$)
16      for $j \in \{2, ..., |n.branch|\}$ do
17          $cur\_dist \longleftarrow$ dist($TS[i], n.branch[j].\mathcal{L}$)
18          if $cur\_dist < min\_dist$ then
19              $min\_dist \longleftarrow cur\_dist$
20              $min\_index \longleftarrow j$
21      $n' \longleftarrow n.branch[min\_index]$
22      $n'.\mathcal{L} \longleftarrow n'.\mathcal{L} \bigcup \{TS[i]\}$
23      update_ideal_nadir($n', TS[i]$)
```

A. REVERSE STRATEGY

Solutions generated at early generations are usually far from the Pareto front [20], [21]. In Figure 2, red solutions (solutions generated at the first generation) are far away from the Pareto front. With the evolutionary process, the solution set becomes increasingly closer to the Pareto front. The original ND-Tree approach updates the UEA with solutions from the first generation to the last generation (e.g., in Figure 2, the red solutions are used to update the UEA first and the black solutions are used to update the UEA last). Thus, the non-dominated solutions stored in the ND-Tree first distribute at the upper-right corner (i.e., the non-dominated solutions of the red solution set in Figure 2). During the evolutionary process, the non-dominated solution set is gradually updated and moves toward the lower-left corner (i.e., the non-dominated solutions of the green, blue, and then black solution set in Figure 2). We can clearly observe that, during this process, the solutions stored in the ND-Tree are updated too frequently. This is a significant drawback of the ND-tree approach because reconstructing the ND-Tree frequently is a very time-consuming process.

In the case of updating the UEA right after a new solution is generated, we have to follow the generation order of the solutions to update the ND-Tree. However, when we update the ND-Tree after obtaining all the examined solutions, we

Algorithm 5: update_ideal_nadir(n,p)

```
input : $n$ (a node of ND-Tree), $p$ (new candidate
        solution)
begin
1   $is\_updated \longleftarrow$ False
2   for $i \in \{1, ..., M\}$ do
3       if $p[i] < n.ideal[i]$ then
4           $is\_updated \longleftarrow$ True
5           $n.ideal[i] \longleftarrow p[i]$
6       else if $p[i] > n.nadir[i]$ then
7           $is\_updated \longleftarrow$ True
8           $n.nadir[i] \longleftarrow p[i]$
9   if $is\_updated$ and $n.parent$ exists then
10      update_ideal_nadir($n.parent, p$)
```

Algorithm 6: ND-Tree-Reverse

```
input : $S$ (a set of $N$ examined solutions)
output: $root$ (root of ND-Tree)
begin
1   initialize $root$ with $s_N$
2   for $i \in \{N-1, ..., 2\}$ do
3       $is\_dominated \longleftarrow$ update($root, s_i$)
4       if $root$ is deleted then
5           initialize $root$ with $s_i$
6       else if not $is\_dominated$ then
7           insert($root, s_i$)
```

do not have to follow the generation order of the solutions for updating the ND-Tree. Instead, we can reverse the generation order of the solutions and update the ND-Tree using the reversed sequence of the solutions. In Algorithm 6, the last generated solution (s_N) is first used to initialize the ND-Tree; then, the next solution (s_{N-1}) is used to update the ND-Tree, and so on. The potential benefit of reversing the solution order for updating the ND-Tree is the reduction of the frequency of reconstructing the ND-Tree. Since the solutions in later generations are more likely to dominate those in earlier generations, when we store the solutions of the final population in the ND-Tree first, these solutions are hard to be removed from the ND-Tree. Thus, we do not have to reconstruct the ND-Tree frequently. For the ND-Tree-Reverse approach, since the non-dominated solutions of all the examined solutions (e.g., black solutions on the Pareto front in Figure 2) are stored in the ND-Tree at the early stage of the updating, a number of solutions at early generations (e.g., green solutions in Figure 2) can be dominated by the nadir point of node $root$ of the ND-Tree. Therefore, those solutions can be simply rejected during the ND-Tree updating, which significantly decreases the computation time.

B. COMPUTATIONAL COMPLEXITY

1) ND-Tree approach

Let the number of the child nodes (B) be 2, and the probability that both children need to be further processed (i.e., neither child meets the three properties mentioned in Section II) be c_1. The computational complexity of update is $\Theta(N_1^{c_1})$, where N_1 is the number of solutions in the current ND-Tree [18]. In practice, the probability c_1 is smaller than 1, so the computational complexity of update is sublinear in time with respect to N_1.

When the number of the child nodes (B) is 2, the average-case computational complexity of insert is $\Theta(\log N_1)$. The computational complexity of update_ideal_nadir is the same as that of insert since it goes up the tree starting from a leaf node, which is equivalent to going down the tree. split has a constant time complexity since it depends on the maximum size of a leaf node (C) and the number of child nodes (B). Thus, the total computational complexity of updating an ND-Tree is $\Theta(\max\{N_1^{c_1}, \log N_1\})$.

2) ND-Tree-Reverse approach

The computational complexity of the ND-Tree-Reverse approach is the same as that of the ND-Tree approach. This is because the code of the ND-Tree-Reverse approach is the same as that of the ND-Tree approach since we only change the order of the solutions to update the ND-Tree. However, our reverse strategy can increase the probability that a newly added solution is dominated by the nadir point of node $root$. Thus, in the following toy example and Section IV, the ND-Tree-Reverse approach is much faster than the ND-Tree approach. It is worth noting that the obtained solution sets by the ND-Tree approach and the ND-Tree-Reverse approach are always the same. This is because both approaches find all non-dominated solutions of the given dataset. In this paper, we use the runtime to compare the performance of the two approaches. The performance indicators, such as HV [22] and IGD [23], are not used since the output solution sets by the two approaches are the same.

C. A TOY EXAMPLE

The proposed strategy is very simple yet effective. In the remainder of this section, we use a toy example to preliminarily verify the superiority of the ND-Tree-Reverse approach. The ND-Tree approach and the ND-Tree-Reverse approach are compared on 21 artificial bi-objective datasets.

1) Artificial bi-objective datasets

An artificial bi-objective dataset $S = \{S_g | g = 1, ..., 200\}$ where S_g is a subset containing 200 solutions (i.e., $S_g = \{s_g^1, s_g^2, ..., s_g^{200}\}$) at the g-th generation is generated as follows. We first generate a solution ($p = (p_1, p_2)$), where its first element (p_1) is sampled from a normal distribution with mean $\mu = 0$ and standard deviation $\sigma = 0.1$, and its second element (p_2) is sampled from a uniform distribution over $(-1/\sqrt{2}, 1/\sqrt{2})$. That is, $p_1 \sim N(0, 0.01)$ and

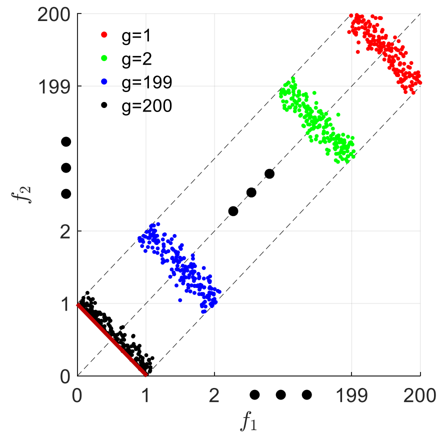

FIGURE 2. An artificial dataset that contains all solutions generated in the first 200 generations. Solutions generated at each generation form a subset in this artificial solution set (e.g., the red subset is generated at the first generation). The red solid line denotes the Pareto front of this dataset.

$p_2 \sim U(-1/\sqrt{2}), 1/\sqrt{2})$. Then, we apply the following transformation to rotate p anticlockwise to obtain p^*:

$$p^* = p \cdot \begin{bmatrix} \cos\theta & -\cos\theta \\ \sin\theta & \cos\theta \end{bmatrix}, \tag{2}$$

where θ is a rotation degree, and it is set to $\pi/4$ here. We iterate the above procedure to generate 200 rotated solutions. The 200 rotated solutions form the initial subset S_o. The subset S_g is obtained by applying the following transformation:

$$S_g = S_o + (200 - g + 0.5), g = 1, 2, ..., 200. \tag{3}$$

So far, a subset S_g has been obtained. All subsets (i.e., $S_1, S_2, ..., S_{200}$) can be easily obtained by repeating the above procedure with different values of g. The artificial bi-objective dataset S is composed of the 200 subsets.

Figure 2 shows part of the solution sets in S. For example, red points are solutions generated at the first generation (i.e., S_1). It is worth noting that when we generate S_{200} (i.e., the black solutions in Figure 2), S_o in (3) are generated only from the positive part of the normal distribution (i.e., half-normal distribution). This is to guarantee that the Pareto front of this dataset S is a line with two ends (1,0) and (0, 1) (i.e., the red line in Figure 2).

2) Experiments

We generate 21 artificial bi-objective datasets by 21 independent runs of the abovementioned dataset generation procedure. The ND-Tree approach and the ND-Tree-Reverse approach are compared on each artificial dataset. The two parameters C and B in the ND-Tree approach are set as 20 and 6, respectively, as suggested in [18]. Figure 3 shows the comparison of runtime (in milliseconds) of the ND-Tree approach and the ND-Tree-Reverse approach on the 21 artificial datasets. We can observe that the ND-Tree-Reverse approach is approximately seven times faster than the original ND-Tree approach.

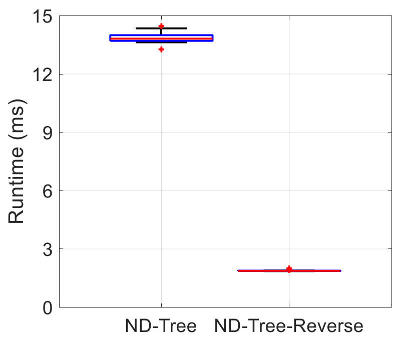

FIGURE 3. Comparison of runtime (in milliseconds) of the ND-Tree approach and the ND-Tree-Reverse approach on the 21 artificial bi-objective datasets.

When all the examined solutions are stored, the ND-Tree-Reverse approach is much faster than the original ND-Tree approach. This is because the ND-Tree-Reverse approach can avoid many unnecessary reconstruction operations. In most cases, solutions generated in earlier generations are likely to be dominated by the nadir point of node $root$. The ND-Tree-Reverse approach simply rejects those solutions without changing the ND-Tree structure. This toy example shows that the time cost of the ND-Tree-Reverse approach is much less than that of the original ND-Tree approach. In the next section, we will use more realistic and complex datasets to examine the effectiveness of the proposed reverse strategy.

IV. EXPERIMENTS

A. SETTINGS FOR GENERATING DATASETS

To obtain the datasets for computational experiments in this paper, we run NSGA-II on four MOPs: multi-objective knapsack problem (MOKP) [22], multi-point distance minimization problem (MPDMP) [24], PS1 and PS2 [25]. For each problem, we obtain 21 datasets by 21 independent runs. In each run, the generated solutions at each generation are stored as a subset (e.g., S_1). All the subsets generated during the whole evolutionary process form a single dataset. Different numbers of objectives are considered (i.e., $M \in \{2, 3, 5, 10\}$).

It is worth noting that the same datasets are used by different approaches (i.e., the ND-Tree approach and the ND-Tree-Reverse approach). Thus, the outputs of the approaches are the same since the non-dominated solution sets of the same datasets are identical. In this section, we only use the runtime to evaluate their performance.

1) Multi-objective knapsack problem (MOKP)

The multi-objective knapsack problem (MOKP [22]) has been used in many studies for evaluating EMO algorithms. MOKP has two parameters: the number of items and the number of knapsacks. The former is the dimension of the decision space, and the latter is the number of objectives. In this paper, the number of items is set to 250. The number of knapsacks is 2, 3, 5, and 10.

2) Multi-point distance minimization problem (MPDMP)

The multi-point distance minimization problem (MPDMP [24]) has been used for visually examining the behavior of EMO algorithms. In MPDMP [24], the dimension of the decision space is set as two. There is a regular M-sided polygon in the center of the decision space. In the decision space, the distances between a solution and the vertexes of the polygon are the objectives to be minimized. In the original MPDMP, the randomly generated initial solutions are often very close to or on the Pareto front since they distribute uniformly in the two-dimensional decision space. This characteristic is very rare in real-world problems [20], [21]. To address this issue, we use a mapping approach to convert the two-dimensional decision space to a high-dimensional decision space [26]. In this way, the randomly generated initial solutions are not very close to or on the Pareto front. In this paper, the dimension of the decision space of MPDMP is set as 10.

3) PS1 and PS2

The other two problems are PS1 and PS2 [25]. Similar to MPDMP, the PS problems are distance-based problems, and the objectives of a solution are the distances between the solution and the vertexes of the polygon in the decision space. However, the polygon of the PS problems is an $(M-1)$-dimensional polygon rather than a two-dimensional M-sided polygon. The difference between PS1 and PS2 is that PS1 is unconstrained and PS2 is constrained. In addition, for PS2, the dimension of the Pareto set can be modified. In this paper, the dimensions of the decision spaces of PS1 and PS2 are both set to 30.

4) Non-dominated sorting genetic algorithm II (NSGA-II)

The non-dominated sorting genetic algorithm II (NSGA-II [1]) is a popular MOEA based on the Pareto domination relation in the evolutionary optimization community. NSGA-II uses non-dominated sorting and density estimation to guarantee the convergence and the diversity of solutions, respectively. The non-dominated sorting is used as the main fitness evaluation criterion to push the population of solutions toward the Pareto front. The density estimation (i.e., crowding distance in [1]) is used as the secondary fitness evaluation criterion to increase the diversity of solutions in the population over the entire Pareto front.

To generate datasets by NSGA-II, the population size is set to 200 and the maximum number of the function evaluations is set to 40000. That is, each dataset has 200 subsets of solutions (i.e., $S_1, S_2, ..., S_{200}$), where S_i denotes the solutions generated in the i-th generation. The SBX crossover and the polynomial mutation are used to generate the datasets for MPDMP, PS1 and PS2, where each solution is represented by a real number vector (i.e., read number coding). Following the practice in [4], [5], the crossover probability and mutation probability are set to 1 and $1/D$, respectively, where D is the dimension of the decision space (i.e., the number of decision variables). The distribution indices of both operators are set to 20. For MOKP where each solution is represented by a binary string (i.e., binary coding), the one-point crossover with the probability 1 and the bit-flip mutation with the mutation probability $1/D$ are used to generate new solutions. The same greedy repair method as in [21], [22] is used to handle infeasible solutions. It is worth noting that all the datasets are obtained from PlatEMO [27].

B. SETTINGS FOR ND-TREE

There are two parameters in the ND-Tree: the maximum size of a leaf node (C) and the number of child nodes (B). In the experiments, different values of B are considered (i.e., $B \in \{6, 10, 14, 18\}$), and different values of C are considered (i.e., $C \in \{10, 15, 20, 25\}$).

We use a slightly different implementation of the ND-Tree approach in [18]. The only difference lies in the method of handling 2-objective cases. In the implementation of the ND-Tree approach provided by [18], when a new 2-objective solution is inserted into a leaf node, the new solution is compared with the solutions in the leaf node. However, this process is redundant since it has already been performed when the ND-Tree is updated with the new solution, as described in Section II. In this study, to perform a fair comparison, this redundant process is removed. For three or more objectives, our implementation is exactly the same as in [18].

All experiments are performed on a machine with 32 Intel(R) Xeon(R) CPU E5-2667 v4 @ 3.20 GHz running Ubuntu 16.4.

C. COMPARISON OF THE RESULTS

In this subsection, we compare the runtime between the ND-Tree approach and the ND-Tree-Reverse approach on the datasets generated by NSGA-II on the above four problems. Table 1 shows the runtime (in microseconds) of the ND-Tree approach and the ND-Tree-Reverse approach with $C \in \{10, 15, 20, 25\}$ and $B = 6$. In Table 1, the ND-Tree approach and the ND-Tree-Reverse approach are referred to as NT and NTR, respectively. The ND-Tree approach with C is denoted as NT(C). Similarly, the ND-Tree-Reverse approach with C is denoted as NTR(C). Table 2 shows the runtime (in microseconds) of the ND-Tree approach and the ND-Tree-Reverse approach with $B \in \{6, 10, 14, 18\}$ and $C = 20$. Similar to Table 1, the ND-Tree approach and ND-Tree-Reverse approach with B are denoted as NT(B) and NTR(B), respectively. From Table 1 and Table 2, we can conclude that the ND-Tree-Reverse approach is much faster than the ND-Tree approach in all cases. For example, when $C = 10$, $B = 6$, and $M = 2$, the ND-Tree-Reverse approach is more than two times faster than the ND-Tree approach.

1) Relative runtime

To show the results in Table 1 more clearly, in this subsection, we define the relative runtime as the runtime of the ND-Tree-Reverse approach divided by the runtime of the ND-Tree approach. For example, if the relative runtime is 0.5, the runtime of the ND-Tree-Reverse approach is half that of the

TABLE 1. Runtime (in microseconds) of the ND-Tree approach (NT) and the ND-Tree-Reverse approach (NTR) on the datasets of MOKP, MPDMP, PS1, and PS2. Four different values of C are considered (i.e., 10, 15, 20, and 25) and $B = 6$. NT(C) and NTR(C) refer to the ND-Tree approach with C and the ND-Tree-Reverse approach with C, respectively. The best results are shaded. The three symbols $+$, $-$, and $\approx$ denote that the NTR is significantly faster than, slower than, and similar to NT, respectively.

Problem	M	D	NT(10)	NTR(10)	NT(15)	NTR(15)	NT(20)	NTR(20)	NT(25)	NTR(25)
MOKP	2	250	1.9002e+4	8.9996e+3 $+$	1.2470e+4	8.3010e+3 $+$	1.0551e+4	8.6120e+3 $+$	1.0511e+4	8.7804e+3 $+$
	3	250	6.3344e+4	2.2174e+4 $+$	4.8746e+4	2.0716e+4 $+$	3.9857e+4	2.0759e+4 $+$	3.7490e+4	2.1060e+4 $+$
	5	250	2.1935e+5	1.0258e+5 $+$	1.8680e+5	9.6708e+4 $+$	1.8203e+5	9.5769e+4 $+$	1.6616e+5	9.2818e+4 $+$
	10	250	1.1664e+6	6.3084e+5 $+$	1.0423e+6	5.9112e+5 $+$	1.0035e+6	5.9726e+5 $+$	9.5705e+5	5.8763e+5 $+$
MPDMP	2	10	1.1866e+4	5.0122e+3 $+$	9.7887e+3	4.9204e+3 $+$	9.3640e+3	4.9295e+3 $+$	8.9476e+3	4.9280e+3 $+$
	3	10	2.7196e+4	1.0527e+4 $+$	2.2789e+4	1.0375e+4 $+$	2.0848e+4	1.0245e+4 $+$	1.9722e+4	1.0218e+4 $+$
	5	10	3.6136e+4	1.6113e+4 $+$	3.1437e+4	1.5938e+4 $+$	2.9230e+4	1.6220e+4 $+$	2.7932e+4	1.5917e+4 $+$
	10	10	5.7476e+4	2.3157e+4 $+$	4.5101e+4	2.1651e+4 $+$	4.1549e+4	2.1933e+4 $+$	3.9573e+4	2.2121e+4 $+$
PS1	2	30	5.6908e+3	2.9329e+3 $+$	4.8182e+3	2.8170e+3 $+$	4.7301e+3	2.8335e+3 $+$	4.5155e+3	2.8385e+3 $+$
	3	30	1.5731e+4	6.2913e+3 $+$	1.1961e+4	5.5731e+3 $+$	1.0560e+4	5.5311e+3 $+$	1.0179e+4	5.5885e+3 $+$
	5	30	4.2317e+4	1.5371e+4 $+$	3.3415e+4	1.4602e+4 $+$	2.9820e+4	1.4525e+4 $+$	2.8065e+4	1.4596e+4 $+$
	10	30	1.2534e+5	4.1492e+4 $+$	1.0034e+5	3.8273e+4 $+$	9.0863e+4	3.8325e+4 $+$	8.7784e+4	3.9374e+4 $+$
PS2	2	30	4.5778e+3	2.5836e+3 $+$	3.8136e+3	2.5366e+3 $+$	3.7453e+3	2.5250e+3 $+$	3.6521e+3	2.5803e+3 $+$
	3	30	1.1072e+4	4.6837e+3 $+$	8.8050e+3	4.4263e+3 $+$	8.1184e+3	4.4557e+3 $+$	7.7475e+3	4.3978e+3 $+$
	5	30	3.3116e+4	1.1436e+4 $+$	2.7193e+4	1.1194e+4 $+$	2.4332e+4	1.1117e+4 $+$	2.3153e+4	1.1225e+4 $+$
	10	30	1.1155e+5	3.5527e+4 $+$	9.4342e+4	3.3771e+4 $+$	8.5717e+4	3.3326e+4 $+$	8.3521e+4	3.3295e+4 $+$
$+/-/\approx$				16/0/0		16/0/0		16/0/0		16/0/0

TABLE 2. Runtime (in microseconds) of the ND-Tree approach (NT) and the ND-Tree-Reverse approach (NTR) on the datasets of MOKP, MPDMP, PS1, and PS2. Four different values of B are considered (i.e., 6, 10, 14, and 18) and $C = 20$.

Problem	M	D	NT(6)	NTR(6)	NT(10)	NTR(10)	NT(14)	NTR(14)	NT(18)	NTR(18)
MOKP	2	250	1.0551e+4	8.6120e+3 $+$	1.0451e+4	8.5581e+3 $+$	1.0746e+4	8.6389e+3 $+$	1.0730e+4	8.8947e+3 $+$
	3	250	3.9857e+4	2.0759e+4 $+$	4.0181e+4	2.1058e+4 $+$	4.0116e+4	2.0004e+4 $+$	4.2334e+4	2.0281e+4 $+$
	5	250	1.8203e+5	9.5769e+4 $+$	1.7426e+5	9.4028e+4 $+$	1.8114e+5	9.3628e+4 $+$	1.8180e+5	9.8489e+4 $+$
	10	250	1.0035e+6	5.9726e+5 $+$	1.0279e+6	6.1915e+5 $+$	1.0083e+6	6.5270e+5 $+$	1.1116e+6	6.8001e+5 $+$
MPDMP	2	10	9.3640e+3	4.9295e+3 $+$	9.2861e+3	5.0585e+3 $+$	9.3030e+3	4.9399e+3 $+$	9.6074e+3	5.2319e+3 $+$
	3	10	2.0848e+4	1.0245e+4 $+$	2.1004e+4	1.0480e+4 $+$	2.1645e+4	1.1065e+4 $+$	2.2390e+4	1.1653e+4 $+$
	5	10	2.9230e+4	1.6220e+4 $+$	2.9484e+4	1.6086e+4 $+$	3.0077e+4	1.7129e+4 $+$	3.1029e+4	1.8137e+4 $+$
	10	10	4.1549e+4	2.1933e+4 $+$	4.2233e+4	2.2498e+4 $+$	4.2884e+4	2.3794e+4 $+$	4.3927e+4	2.5313e+4 $+$
PS1	2	30	4.7301e+3	2.8335e+3 $+$	4.6253e+3	2.8074e+3 $+$	4.6541e+3	2.8643e+3 $+$	4.7088e+3	2.9074e+3 $+$
	3	30	1.0560e+4	5.5311e+3 $+$	1.0683e+4	5.4045e+3 $+$	1.0831e+4	5.5941e+3 $+$	1.1036e+4	5.5802e+3 $+$
	5	30	2.9820e+4	1.4525e+4 $+$	2.9333e+4	1.4151e+4 $+$	3.0047e+4	1.4699e+4 $+$	3.0483e+4	1.5245e+4 $+$
	10	30	9.0863e+4	3.8325e+4 $+$	9.1078e+4	3.7051e+4 $+$	8.9858e+4	3.7083e+4 $+$	9.2019e+4	3.8141e+4 $+$
PS2	2	30	3.7453e+3	2.5250e+3 $+$	3.6771e+3	2.5345e+3 $+$	3.7225e+3	2.5490e+3 $+$	3.7502e+3	2.5633e+3 $+$
	3	30	8.1184e+3	4.4557e+3 $+$	8.0886e+3	4.3797e+3 $+$	8.3292e+3	4.3667e+3 $+$	8.4683e+3	4.4433e+3 $+$
	5	30	2.4332e+4	1.1117e+4 $+$	2.4422e+4	1.0717e+4 $+$	2.4968e+4	1.1302e+4 $+$	2.5162e+4	1.1286e+4 $+$
	10	30	8.5717e+4	3.3326e+4 $+$	8.4188e+4	3.1243e+4 $+$	8.3321e+4	3.1453e+4 $+$	8.4333e+4	3.1611e+4 $+$
$+/-/\approx$				16/0/0		16/0/0		16/0/0		16/0/0

ND-Tree approach. Thus, the smaller the relative runtime is, the better the ND-Tree-Reverse approach performs relative to the ND-Tree approach.

2) Effects of different values of C

Figure 4 (a) shows the relative runtime of the ND-Tree-Reverse approach on MOKP with $C \in \{10, 15, 20, 25\}$ and $B = 6$. When $M = 2$ and $C = 10$, the relative runtime for MOKP is approximately 0.5. The relative runtime gradually increases with the increase in C. For example, when $M = 2$, the runtime of the ND-Tree-Reverse approach is approximately 0.5 times smaller than that of the ND-Tree approach with $C = 10$. However, with $C = 25$, the runtime of the ND-Tree-Reverse approach is approximately 0.8 times smaller than that of the ND-Tree approach. Although the relative runtime with $M > 2$ increases with the increase in C, their growth is smaller than that with $M = 2$. In general, the relative runtime for MOKP is approximately 0.5. Figure 4 (b)-(d) show the relative runtimes on MPDMP, PS1, and PS2, respectively. Similarly, the relative runtime for the three MOPs increases with the increase in C, and their relative runtime is approximately 0.5.

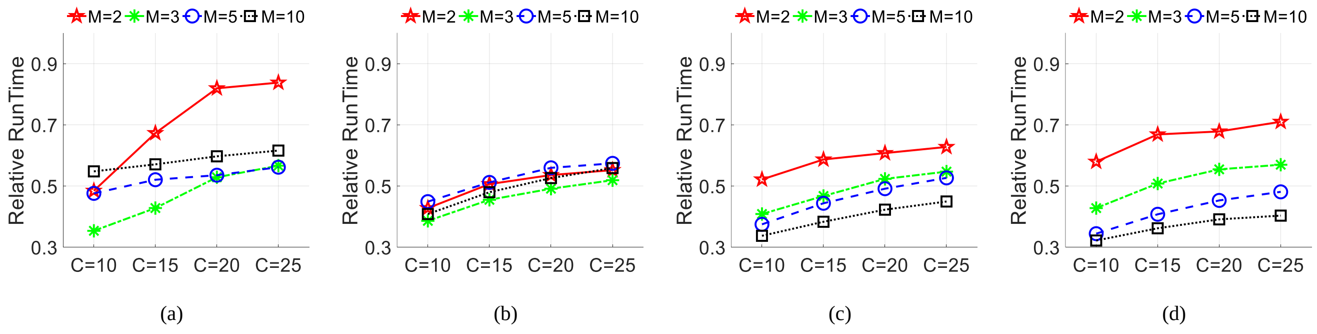

FIGURE 4. Relative runtimes over different values of C on the datasets of (a) MOKP, (b) MPDMP, (c) PS1, and (d) PS2 with $M \in \{2, 3, 5, 10\}$.

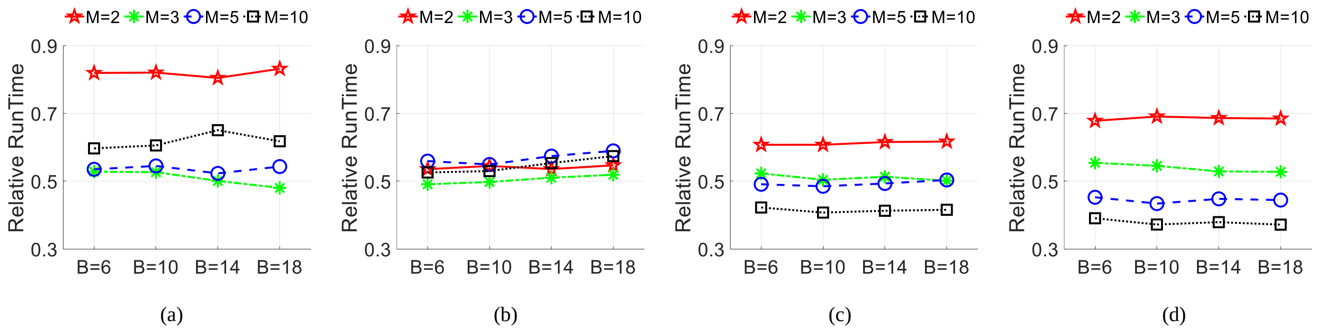

FIGURE 5. Relative runtimes over different values of B on the datasets of (a) MOKP, (b) MPDMP, (c) PS1, and (d) PS2 with $M \in \{2, 3, 5, 10\}$.

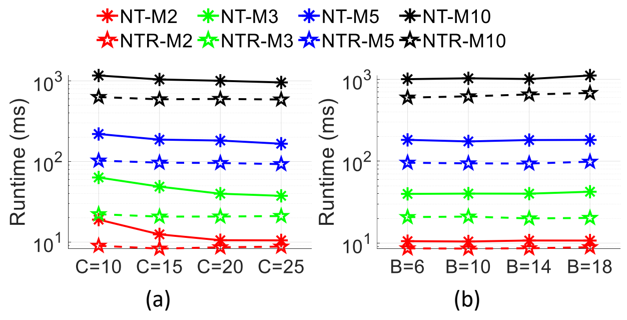

FIGURE 6. Runtime (in log scale) of the ND-Tree approach (NT) and the ND-Tree-Reverse approach (NTR) on the datasets of MOKP for (a) different values of C and (b) different values of B. NT-Mm and NTR-Mm denote the ND-Tree approach and the ND-Tree-Reverse approach when the number of objectives M is m, respectively.

3) Effects of different values of B

Figure 5 shows the relative runtime of the ND-Tree-Reverse approach on MOKP, MPDMP, PS1 and PS2 when $C = 20$ and $B \in \{6, 10, 14, 18\}$. In all cases, the relative runtime has almost no change with the increase in the parameter B. This observation shows that the performance of the proposed strategy is not sensitive to the parameter B.

4) Discussion

As discussed in Section III, split in the ND-Tree approach has a constant time complexity since it depends on C and B. When C and B are set to small values, split is executed frequently, but the time cost of each execution is very low. When C and B are very large, split is executed less frequently, but the time cost of each execution is large. Thus, the time cost of split compensates the frequency of split to keep the runtime of the two approaches less sensitive to C and B. This is also consistent with the results from [18].

Figure 6 (a) shows the runtime (in log scale) of the ND-Tree approach and the ND-Tree-Reverse approach for different values of C on MOKP with $B = 6$. The ND-Tree approach with $M = m$ is denoted as NT-Mm in Figure 5. Similarly, the ND-Tree-Reverse approach with $M = m$ is denoted as NTR-Mm. As shown in Figure 6 (a), the runtime of the ND-Tree approach and the ND-Tree-Reverse approach for MOKP has no significant change with the increase in C. However, compared to the ND-Tree-Reverse approach, when C increases, the runtime of the ND-Tree approach decreases slightly. Thus, the relative runtime of the ND-Tree-Reverse approach increases with the increase in C. In other words, the ND-Tree-Reverse approach is less sensitive to the value of C compared to the ND-Tree approach. The two approaches show results similar to those of the other three problems.

Figure 6 (b) shows the runtime (in log scale) of the ND-

TABLE 3. Runtime (in microseconds) of the seven update approaches on the datasets generated by NSGA-II on MOKP, MPDMP, PS1, and PS2 with $M \in \{2, 3, 5, 10\}$ over 21 independent runs, where $C = 20$ and $B = 6$. NT denotes the ND-Tree approach, NTR denotes the ND-Tree-Reverse approach, and the NTRg denotes the ND-Tree-Reverse approach for updating the non-dominated tree every g generations. The best results are shaded. The three symbols $+$, $-$, and $\approx$ denote that NTRg is significantly faster than, slower than, and similar to NT, respectively.

Problem	M	D	NT	NTR5	NTR10	NTR20	NTR50	NTR100	NTR
MOKP	2	250	1.0551e+4	1.0582e+4 $\approx$	1.0318e+4 $\approx$	9.6887e+3 $+$	8.9110e+3 $+$	8.7445e+3 $+$	8.6120e+3 $+$
	3	250	3.9857e+4	3.8606e+4 $\approx$	3.6100e+4 $+$	3.2123e+4 $+$	2.5611e+4 $+$	2.1483e+4 $+$	2.0759e+4 $+$
	5	250	1.8203e+5	1.6640e+5 $+$	1.6205e+5 $+$	1.4911e+5 $+$	1.2703e+5 $+$	1.1036e+5 $+$	9.5769e+4 $+$
	10	250	1.0035e+6	9.5866e+5 $\approx$	9.5937e+5 $\approx$	9.0812e+5 $+$	7.7766e+5 $+$	6.9964e+5 $+$	5.9726e+5 $+$
MPDMP	2	10	9.3640e+3	8.1043e+3 $+$	7.1250e+3 $+$	6.3194e+3 $+$	5.4740e+3 $+$	4.9742e+3 $+$	4.9295e+3 $+$
	3	10	2.0848e+4	1.8699e+4 $+$	1.6857e+4 $+$	1.4911e+4 $+$	1.2615e+4 $+$	1.0899e+4 $+$	1.0245e+4 $+$
	5	10	2.9230e+4	2.6690e+4 $\approx$	2.4229e+4 $+$	2.1513e+4 $+$	1.8165e+4 $+$	1.6757e+4 $+$	1.6220e+4 $+$
	10	10	4.1549e+4	3.8244e+4 $+$	3.4563e+4 $+$	3.0230e+4 $+$	2.5253e+4 $+$	2.3215e+4 $+$	2.1933e+4 $+$
PS1	2	30	4.7301e+3	4.1393e+3 $+$	3.7302e+3 $+$	3.3757e+3 $+$	3.0137e+3 $+$	2.8532e+3 $+$	2.8335e+3 $+$
	3	30	1.0560e+4	9.1438e+3 $+$	8.0517e+3 $+$	7.1481e+3 $+$	6.2994e+3 $+$	5.5673e+3 $+$	5.5311e+3 $+$
	5	30	2.9820e+4	2.6301e+4 $+$	2.3378e+4 $+$	1.9398e+4 $+$	1.5816e+4 $+$	1.4591e+4 $+$	1.4525e+4 $+$
	10	30	9.0863e+4	8.0584e+4 $+$	7.0924e+4 $+$	6.0635e+4 $+$	4.5906e+4 $+$	3.8632e+4 $+$	3.8325e+4 $+$
PS2	2	30	3.7453e+3	3.2827e+3 $+$	3.0005e+3 $+$	2.7685e+3 $+$	2.6420e+3 $+$	2.5536e+3 $+$	2.5250e+3 $+$
	3	30	8.1184e+3	6.9984e+3 $+$	6.2407e+3 $+$	5.3938e+3 $+$	4.9126e+3 $+$	4.5034e+3 $+$	4.4557e+3 $+$
	5	30	2.4332e+4	2.1380e+4 $+$	1.8464e+4 $+$	1.5693e+4 $+$	1.2663e+4 $+$	1.1213e+4 $+$	1.1117e+4 $+$
	10	30	8.5717e+4	7.7782e+4 $+$	6.7664e+4 $+$	5.3119e+4 $+$	4.2655e+4 $+$	3.3682e+4 $+$	3.3326e+4 $+$
$+/-/\approx$				12/0/4	14/0/2	16/0/0	16/0/0	16/0/0	16/0/0

Tree approach and the ND-Tree-Reverse approach for different values of B on MOKP with $C = 20$. The runtime of the ND-Tree approach and the ND-Tree-Reverse approach for MOKP almost remains constant with the increase in B. This is why the relative runtime is not sensitive to the parameter B.

V. INVESTIGATION OF OPTIMAL FREQUENCY

To obtain the non-dominated solutions from all the examined solutions, there are two strategies for updating the ND-Tree. That is, the ND-Tree approach updates the ND-Tree with the normal order (i.e., $\{S_1, S_2, ..., S_{200}\}$), and the ND-Tree-Reverse approach updates the ND-Tree with the reverse order (i.e., $\{S_{200}, S_{199}, ..., S_1\}$), where S_g is a subset of solutions generated at the g-th generation. In this section, we investigate the effect of the update frequency on the performance of the ND-Tree-Reverse approach. That is, the ND-Tree-Reverse approach is applied every g generations, where $g \in \{5, 10, 20, 50, 100\}$. For instance, if the ND-Tree is updated every 100 generations, the solutions used to update the ND-Tree are in the order of $\{\{S_{100}, S_{99}, ..., S_1\}, \{S_{200}, S_{199}, ..., S_{101}\}\}$. Table 3 shows the runtime (in microseconds) of the ND-Tree approach, the ND-Tree-Reverse approach, and the ND-Tree-Reverse approach for updating the ND-Tree every g generations, where $g \in \{5, 10, 20, 50, 100\}$. Because the results are similar for different values of C and B, we only show the results with $C = 20$ and $B = 6$ in Table 3. The original ND-Tree approach is denoted as NT, the ND-Tree-Reverse approach for updating the ND-Tree every g generations is denoted as NTRg, and the ND-Tree-Reverse approach for updating the ND-Tree once at the end of the evolutionary process is denoted as NTR. Clearly, for all instances, the ND-Tree-Reverse approach is the optimal approach. That is, updating the ND-Tree once with the reverse strategy at the end of the evolutionary process is suggested.

To show the results more clearly, Figure 7 shows the relative runtime of the six update approaches (i.e., NTR5, NTR10, NTR20, NTR50, NTR100, and NTR) for the four MOPs (i.e., MOKP, MPDMP, PS1, and PS2). Similar to the definition of the relative runtime in Section IV, the relative runtime of an approach is the runtime of the approach divided by the runtime of the ND-Tree approach, which can be viewed as NTR1 since it updates the ND-Tree every generation. As shown in Figure 7, with the decrease in the update frequency (i.e., the increase in g), the relative runtime of the update approach also decreases. For instance, in Figure 7 (a), when $M = 10$, the relative runtime of MOKP decreases from 1 to 0.6 with the decrease in the update frequency. This is because when we update the ND-Tree with a low frequency, the solutions inserted into the ND-Tree earlier are less likely to be replaced by later solutions.

Although the experimental results suggest utilizing the ND-Tree-Reverse approach to obtain the non-dominated solutions once at the end of the evolutionary process, the ND-Tree-Reverse approach with a certain frequency is an excellent approach when we need to use the UEA during the evolutionary process. In this situation, the ND-Tree-Reverse approach with a certain frequency is a suggested approach instead of the original ND-Tree approach.

VI. CONCLUSION

In this paper, we proposed a simple yet effective strategy to enhance the ND-Tree approach to obtain the non-dominated solutions at the end of the evolutionary process. Experimental results showed that the ND-Tree approach with the reverse

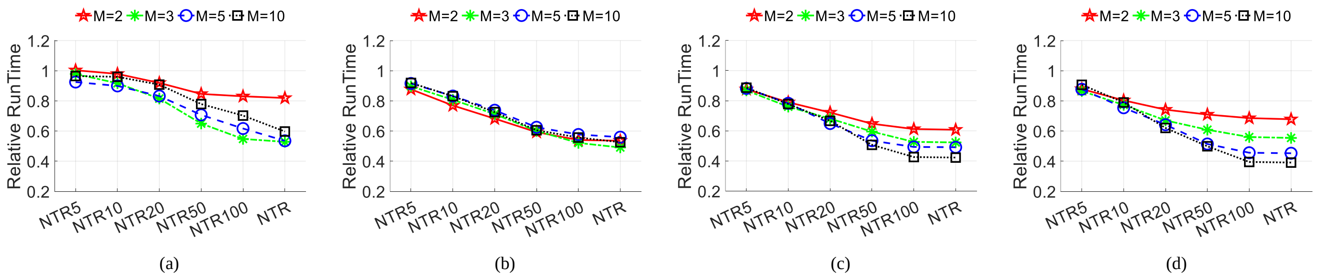

FIGURE 7. Relative runtimes over different values of g on the datasets of (a) MOKP, (b) MPDMP, (c) PS1, and (d)PS2 with $M \in \{2, 3, 5, 10\}$.

strategy is much faster than the ND-Tree approach on MOKP, MPDMP, PS1, and PS2 with up to 10 objectives. We also investigated the effect of the frequency of updating the UEA on the performance of the reverse strategy. The experimental results suggested that the most efficient way is to update the UEA only once at the end of the evolutionary process. Moreover, with the decrease in the update frequency, the runtime of the ND-Tree-Reverse approach also decreases.

In the future, we may need to examine the performance of various approaches (including ND-Tree and ND-Tree-Reverse) on very large datasets. In our computational experiments in this paper, the termination condition was specified as 40,000 (200 generations of the population with 200 solutions). This means that the total number of examined solutions is 40,000. The advantages of the proposed ND-Tree-Reverse approach can be more clearly shown by difficult test problems. This is because these problems require a greater computational load (e.g., 10,000 generations of the population with 1000 solutions) for initial solutions that are far away from the Pareto front. It is also an interesting research topic to examine their performance on datasets where almost all solutions are non-dominated with each other.

REFERENCES

[1] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[2] H. Seada, M. Abouhawwash, and K. Deb, “Multi-phase balance of diversity and convergence in multiobjective optimization,” *IEEE Trans. Evol. Comput.*, vol. 23, no. 3, pp. 503–513, Jun. 2019.

[3] H. Seada and K. Deb, “U-NSGA-III: A unified evolutionary optimization procedure for single, multiple, and many objectives: Proof-of-principle results,” in *Proc. Evol. Multi-Criter. Optim.*, Guimarães, Portugal, Mar. 2015, pp. 34–49.

[4] Q. Zhang and H. Li, “MOEA/D: A multiobjective evolutionary algorithm based on decomposition,” *IEEE Trans. Evol. Comput.*, vol. 11, no. 6, pp. 712–731, Dec. 2007.

[5] Y. Qi, X. Ma, F. Liu, L. Jiao, J. Sun, and J. Wu, “MOEA/D with adaptive weight adjustment,” *Evol. Comput.*, vol. 22, no. 2, pp. 231–264, Jun. 2014.

[6] E. Zitzler and S. Künzli, “Indicator-based selection in multiobjective search,” in *Proc. Int. Conf. Parallel Prob. Solv. Nat.*, Birmingham, United Kingdom, Sep. 2004, pp. 832–842.

[7] J. Bader and E. Zitzler, “HypE: An algorithm for fast hypervolume-based many-objective optimization,” *Evol. Comput.*, vol. 19, no. 1, pp. 45–76, Mar. 2011.

[8] Q. Chen, J. Ding, S. Yang, and T. Chai, “A novel evolutionary algorithm for dynamic constrained multiobjective optimization problems,” *IEEE Trans. Evol. Comput.*, (Early Access).

[9] Y. Fu, M. Zhou, X. Guo, and L. Qi, “Scheduling dual-objective stochastic hybrid flow shop with deteriorating jobs via bi-population evolutionary algorithm,” *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, (Early Access).

[10] Y. Fu, G. Tian, A. M. Fathollahi-Fard, A. Ahmadi, and C. Zhang, “Stochastic multi-objective modelling and optimization of an energy-conscious distributed permutation flow shop scheduling problem with the total tardiness constraint,” *J. Cleaner Prod.*, vol. 226, pp. 515–525, Jul. 2019.

[11] M. Li and X. Yao, “An empirical investigation of the optimality and monotonicity properties of multiobjective archiving methods,” in *Proc. Evol. Multi-Criter. Optim.*, East Lansing, MI, USA, Mar. 2019, pp. 15–26.

[12] K. Deb, M. Mohan, and S. Mishra, “Evaluating the ε-domination based multi-objective evolutionary algorithm for a quick computation of Pareto-optimal solutions,” *Evol. Comput.*, vol. 13, no. 4, pp. 501–525, Dec. 2005.

[13] X. Cai, Y. Li, Z. Fan, and Q. Zhang, “An external archive guided multiobjective evolutionary algorithm based on decomposition for combinatorial optimization,” *IEEE Trans. Evol. Comput.*, vol. 19, no. 4, pp. 508–523, Aug. 2015.

[14] J. E. Fieldsend, R. M. Everson, and S. Singh, “Using unconstrained elite archives for multiobjective optimization,” *IEEE Trans. Evol. Comput.*, vol. 7, no. 3, pp. 305–323, Jun. 2003.

[15] H. G. de Medeiros, E. F. G. Goldbarg, and M. C. Goldbarg, “Investigation of archiving techniques for evolutionary multi-objective optimizers,” *Revista de Informática Teórica e Aplicada.*, vol. 25, no. 4, pp. 11–27, Nov. 2018.

[16] R. Tanabe and H. Ishibuchi, “Non-elitist evolutionary multi-objective optimizers revisited,” in *Proc. Conf. Genet. Evol. Comput.*, Prague Czech Republic, Jul. 2019, pp. 612–619.

[17] G. T. Parks and I. Miller, “Selective breeding in a multiobjective genetic algorithm,” in *Proc. Int. Conf. Parallel Prob. Solv. Nat.*, Amsterdam, The Netherlands, Sep. 1998, pp. 250–259.

[18] A. Jaszkiewicz and T. Lust, “ND-Tree-based update: A fast algorithm for the dynamic nondominance problem,” *IEEE Trans. Evol. Comput.*, vol. 22, no. 5, pp. 778–791, Oct. 2018.

[19] R. Tanabe, H. Ishibuchi, and A. Oyama, “Benchmarking multi-and many-objective evolutionary algorithms under two optimization scenarios,” *IEEE Access.*, vol. 5, pp. 19 597–19 619, Sep. 2017.

[20] H. Ishibuchi, R. Imada, Y. Setoguchi, and Y. Nojima, “Performance comparison of NSGA-II and NSGA-III on various many-objective test problems,” in *Proc. IEEE Congr. Evol. Comput.*, Vancouver, BC, Canada, Jul. 2016, pp. 3045–3052.

[21] H. Ishibuchi, N. Akedo, and Y. Nojima, “Behavior of multiobjective evolutionary algorithms on many-objective knapsack problems,” *IEEE Trans. Evol. Comput.*, vol. 19, no. 2, pp. 264–283, Apr. 2015.

[22] E. Zitzler and L. Thiele, “Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach,” *IEEE Trans. Evol. Comput.*, vol. 3, no. 4, pp. 257–271, Nov. 1999.

[23] P. A. Bosman and D. Thierens, “The balance between proximity and diversity in multiobjective evolutionary algorithms,” *IEEE Trans. Evol. Comput.*, vol. 7, no. 2, pp. 174–188, Apr. 2003.

[24] M. Köppen and K. Yoshida, “Substitute distance assignments in NSGA-II for handling many-objective optimization problems,” in *Proc. Evol. Multi-Criter. Optim.*, Matsushima, Japan, Mar. 2007, pp. 727–741.

[25] O. Schutze, A. Lara, and C. A. C. Coello, “On the influence of the number of objectives on the hardness of a multiobjective optimization problem,” *IEEE Trans. Evol. Comput.*, vol. 15, no. 4, pp. 444–455, Aug. 2011.

[26] H. Ishibuchi, Y. Peng, and K. Shang, "A scalable multimodal multiobjective test problem," in *Proc. IEEE Congr. Evol. Comput.*, Wellington, New Zealand, Jun. 2019, pp. 310–317.

[27] Y. Tian, R. Cheng, X. Zhang, and Y. Jin, "PlatEMO: A MATLAB platform for evolutionary multi-objective optimization," *IEEE Computational Intelligence Magazine*, vol. 12, no. 4, pp. 73–87, Nov. 2017.