Information Sciences 179 (2009) 3540–3560

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

A dominance tree and its application in evolutionary multi-objective optimization

Chuan Shi [a,*], Zhenyu Yan [b], Kevin Lü [c], Zhongzhi Shi [d], Bai Wang [a]

[a] Beijing Key Laboratory of Intelligent Telecommunications Software and Multimedia, Beijing University of Posts and Telecommunications, China
[b] Department of Systems and Information Engineering, University of Virginia, USA
[c] Brunel University, Uxbridge, UB8 3PH, UK
[d] Institute of Computing Technology, Chinese Academy of Sciences, China

ARTICLE INFO

Article history:
Received 3 May 2008
Received in revised form 16 June 2009
Accepted 22 June 2009

Keywords:
Evolutionary multi-objective optimization
Evolutionary computation
Pareto dominance
Fitness assignment
Computational complexity

ABSTRACT

Most contemporary multi-objective evolutionary algorithms (MOEAs) store and handle a population with a linear list, and this may impose high computational complexities on the comparisons of solutions and the fitness assignment processes. This paper presents a data structure for storing the whole population and their dominating information in MOEAs. This structure, called a Dominance Tree (DT), is a binary tree that can effectively and efficiently store three-valued relations (namely dominating, dominated or non-dominated) among vector values. This paper further demonstrates DT's potential applications in evolutionary multi-objective optimization with two cases. The first case utilizes the DT to improve NSGA-II as a fitness assignment strategy. The second case demonstrates a DT-based MOEA (called a DTEA), which is designed by leveraging the favorable properties of the DT. The simulation results show that the DT-improved NSGA-II is significantly faster than NSGA-II. Meanwhile, DTEA is much faster than SPEA2, NSGA-II and an improved version of NSGA-II. On the other hand, in regard to converging to the Pareto optimal front and maintaining the diversity of solutions, DT-improved NSGA-II and DTEA are found to be competitive with NSGA-II and SPEA2.

1. Introduction

Multi-objective optimization problems (MOPs) involve the simultaneous optimization of two or more objectives (often competing) and usually have no single optimal solution [8]. It is usually difficult or even impossible to assign priorities as in single-objective optimization problems (SOPs); thus, an algorithm returning a set of promising solutions is preferable to one returning only one solution that is based on certain weighting of the objectives. For this reason, in the past 20 years, there has been increased interest in applying evolutionary algorithms (EAs) to MOPs [5].

A number of multi-objective evolutionary algorithms (MOEAs) have been proposed [6,9,33]. These MOEAs use Pareto dominance to guide the search and return a set of non-dominated solutions as results. Distinct from SOP, there are two goals in multi-objective optimization: (1) converging to the Pareto optimal set and (2) maintaining the diversity of solutions in the Pareto optimal set [9]. Many strategies and methods have been introduced and utilized to achieve these two goals [28]. To address the first issue, a Pareto-based fitness assignment method is usually designed in order to guide the search toward the true Pareto front [12]. The basic idea here is to rank the solutions according to their dominating relations and many effective

* Corresponding author.
E-mail addresses: shichuan@bupt.edu.cn (C. Shi), yan_zhen_yu@hotmail.com (Z. Yan), jiang.jianglu@gmail.com (K. Lü), shizz@ics.ict.ac.cn (Z. Shi), wangbai@bupt.edu.cn (B. Wang).

0020-0255/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2009.06.035

fitness assignment strategies have been proposed [9,13,33]. In regard to the second issue, some MOEAs have provided density estimation methods in order to preserve the diversity among solutions [2,9,27,29,33].

Several MOEAs have performed well in relation to some benchmark problems and real applications. However, most MOEAs are time-consuming with the complexity $O(GMN^2)$ (where G is the number of generations, M is the number of objectives and N is the population size) [14]. In most Pareto-based MOEAs, the storage of solutions in the population is implemented by a linear list, and the fitness assignment schemes require that each solution is compared with a large number of other solutions, resulting in $O(GMN^2)$ processing time. The N^2 factor here shows that for large population sizes the processing time could become prohibitively long. We should note that in some situations it is desirable to have large population sizes in MOEAs in pursuit of the two goals mentioned above [8]. Recently, more researchers have begun to pay attention to this issue and have designed some effective approaches to improve the efficiency of MOEAs. Deb et al. [9] proposed a fast algorithm to reduce the computational complexity of the non-dominated-sort process in NSGA [25]. Jensen [14] has systematically analyzed the computational complexity of many contemporary MOEAs and has presented several efficient algorithms for the non-dominated sorting process. Moreover, to more efficiently compare and store the solutions, several data structures have also been applied. Due to the high computational complexity of updating solutions in the elite archive (for a detailed description of an elite archive, see [32]), Mostaghim et al. studied the possibility of applying a Quad-tree to store the Pareto-points [16] and Fieldsend et al. introduced a dominated/nondominated tree (DNT) so as to sort the individuals in the elite archive [11]. Although these two data structures can somewhat accelerate the non-dominated sorting process of the elite archive, especially when the elite archive is large, the individuals that they manage are limited to those in the elite archive, rather than the whole population. To resolve this issue, two other data structures were proposed. Alberto and Mateo have presented the irreducible domination graph (IDG) for representing and managing the population of a MOEA [1]. However, their experiments showed that the complexity of constructing an IDG with N nodes is still up to $O(N^2)$. Although Chen introduced another tree structure, the Pareto tree, so as to maintain the non-dominated population of the current comparison granularity, he did not prove that the structure was more efficient [4].

Moreover, the selection process for most contemporary MOEAs is usually a complicated one. In NSGA-II, a crowded-comparison operator is used to guide the selection process [9], in which the solution with the lower rank is preferred, and the solution located in a less crowded region is preferred, when both solutions are found to have the same rank. In the selection process of SPEA2 [33], the non-domination ranks are considered first, and when the archive is too small or too large, the density information is then considered. These selection strategies calculate the rank values and diversity values separately, and then decide the fitness synthetically. Moreover, during this process, many algorithms require sophisticated adjustment of a number of parameters in accordance with the given problems and the analyst's experience. For example, six strategies and four parameters are used in DMOEA [29], and six parameters should be populated in TV-MOPSO [27]. To alleviate the issue, PTSGA integrates the fitness assignment, niching strategy and a non-dominated population reducing method into the Pareto tree [4].

This paper introduces a data structure, called the Dominance Tree (DT) which is used for more efficient population storage and fitness assignment. As a result, efficient selection strategies can also be designed based on DT. The DT is a binary tree that can preserve the dominating information of vectors. Distinct from a traditional binary search tree (BST) [18] that stores the larger than or equal to (for simplicity, it is also called larger than) and smaller than relations of scalar numbers, a DT stores the three-valued relations of solution vectors: dominating, dominated and non-dominated. The structure and construction algorithms given by DT guarantee that it can preserve the necessary dominating relations, implicitly contain the density information, and significantly reduce the number of comparisons among solution vectors. In addition, we perform two case studies with nine well-known test functions to explore the potential applications of DT in evolutionary multi-objective optimization (EMO) and to examine the effectiveness and efficiency of the DT-based MOEAs.

Hence, in this paper, we provide the first detailed and comprehensive discussion of the theory and applications of the DT. The preliminary work on this idea can be found in the other work by the authors [19], where Shi et al. realized that a tree structure can store the relations of solutions of MOP. The DT concept was first proposed by the authors in [22] where the authors described the DTEA. The authors further analyzed the efficiency level of DTEA as well as its favorable characteristics in more comprehensive experiments in [24]. Since DTEA had been criticized for its restricted elite archive and absence of density estimation, Shi et al. improved the functioning of DTEA and validated its performance in [21]. Recently, Shi et al. applied the DT to improve NSGA-II as a general fitness assignment strategy [23]. This paper not only consolidates those studies, it also makes the following two other significant contributions. (1) It first introduces and proposes the formal and complete theory of DT. In this paper, DT is not only depicted as a tool that is restricted to manage the solutions of multi-objective optimization, but also as a general data structure that can effectively store the relations of vector values. This paper also presents a complete and detailed account of the DT's construction algorithms, and compares it with other similar data structures. Moreover, an analysis of the computational complexity of the DT's construction algorithms is given. (2) This paper, for the first time, also comprehensively describes the applications of DT in EMO and validates the effectiveness and efficiency of DT by comparing DT-based algorithms with several other contemporary MOEAs with nine benchmark functions. New criteria and tools are employed to offer a comparison of its performance, as well as to demonstrate its results.

The remainder of this paper is arranged as follows. Section 2 defines DT and further discusses its construction algorithms and properties. Section 3 presents the two applications of DT in EMO. Section 4 presents the experiments and discusses the results. Section 5 discusses some potential improvements of DT-based MOEAs. Finally, Section 6 concludes this paper.

2. Dominance tree

Most contemporary MOEAs store the individuals with a linear list. During the fitness assignment process, each individual is compared to the other individuals, resulting in the N^2 processing time (where N is the population size). However, many unnecessary comparisons can be observed in this process, and these may be classified into two categories: (1) the relations that can be deduced from the existing relations and (2) some relations that are less important to the accomplishment of the goals. Let us consider the dominating relations among individuals to be capable of being visualized on a graph. Fig. 2a, as an example, illustrates the relations among the five solutions that are initially shown in Fig. 1. Each node in this figure corresponds to one solution and each edge shows the relation between two adjacent nodes. Since we can intuitively assume that there may be some redundant relations among the nodes, reducing them would be an effective method to reduce the computational complexity of the fitness assignment process.

Due to the transitive property of Pareto dominance [30], some relations (i.e., edges) can be deduced from the existing relations. Taking Fig. 2a for example, since N5 dominates N3 and N3 dominates N2, we can deduce that N5 dominates N2. Thus, a comparison between N5 and N2 is unnecessary.

We have also noticed that decision makers often make their decisions according to the optimal set and usually pay little attention to other potential solutions. Although dominated solutions are useful to the process of evolution (e.g., maintaining the diversity of solutions), it may not be necessary to record their relations; thus comparisons among the dominated solutions may be avoided. In Fig. 2a, it is clear that N1 and N5 constitute the Pareto optimal set among the five nodes. Thus, the relations among N2, N3 and N4, as dominated solutions, may not be very important or relevant as to either the optimization or processes of decision making.

2.1. Structure of dominance tree

Through such analysis as has been related above, two principles for reducing unnecessary comparisons have been found: (1) to avoid redundant comparisons, (2) to preserve only the necessary and important relations among the individuals. In addition, it is also hoped that the Pareto optimal nodes in the current generation can be easily accessed.

In order to address the first principle, since most fitness assignment strategies are designed based on Pareto dominance and our method is also based on this, further study in regard to the property of Pareto dominance is necessary. Without loss of generality, we only consider the minimization problems in this paper.

Definition 1 (*Pareto dominance*). A vector $\vec{u} = (u_1, \ldots, u_n)$ is said to dominate $\vec{v} = (v_1, \ldots, v_n)$ (denoted by $\vec{u} \preceq \vec{v}$) if and only if $\vec{u}$ is partially less than $\vec{v}$, i.e.,$\forall i \in \{i, \ldots, n\}$: $u_i \leqslant v_i \wedge \exists i \in \{1, \ldots, n\} : u_i < v_i$.

The set of all Pareto optimal decision vectors is called the Pareto optimal set. The corresponding set of objective vectors is called the non-dominated set or the Pareto front. We can deduce that Pareto dominance has the following property, the proof of which can be seen in Appendix A.1.

Property 1. *Pareto dominance has properties of anti-reflexion and transition.*

In comparing MOPs with SOPs, we find a distinct difference in their respective solution comparison mechanisms. The solutions in the objective space of SOP are scalar numbers and their relations have two values: smaller than and larger than. However, the solutions from MOP are vectors and their relations have three values: $\vec{u} \preceq \vec{v}$, $\vec{v} \preceq \vec{u}$ and non-dominated. This can be defined as follows:

Definition 2 (*Better function*).

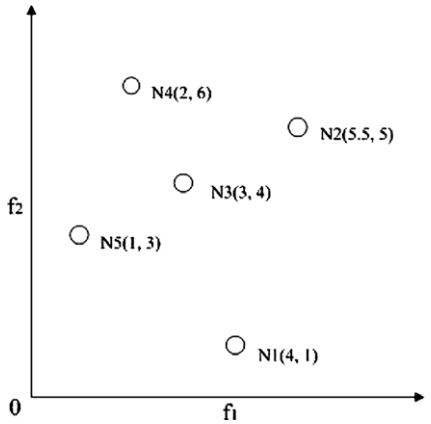

Fig. 1. Coordinates of five solutions with two objectives in the objective space.

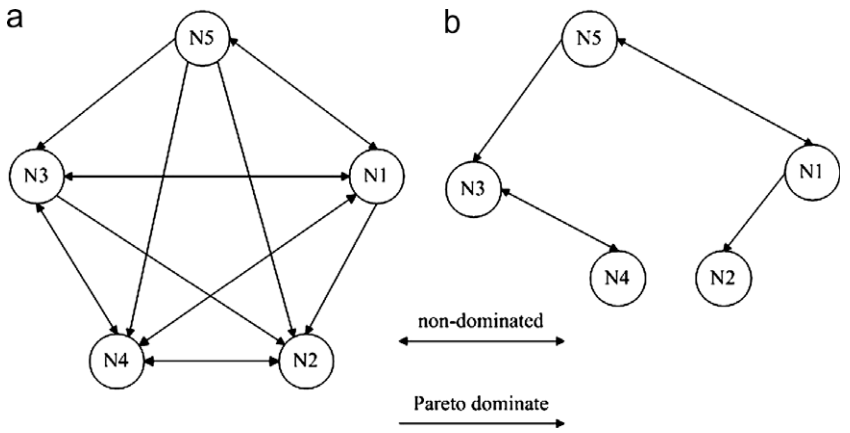

Fig. 2. (a) An illustration of the relations among the nodes with a graph. (b) An illustration of the relations among the nodes with a dominance tree.

$$Better(\vec{u},\vec{v}) = \begin{cases} 1 & \vec{u} \preceq \vec{v} \\ -1 & \vec{v} \preceq \vec{u} \\ 0 & \text{non-dominated} \end{cases}$$

The *Better* function formally defines the relation among vectors. If $Better(\vec{u},\vec{v}) = 1$, $\vec{u}$ dominates $\vec{v}$. If $Better(\vec{u},\vec{v}) = -1$, $\vec{u}$ is dominated by $\vec{v}$ (i.e., $\vec{v}$ dominates $\vec{u}$). If $Better(\vec{u},\vec{v}) = 0$, $\vec{u}$ is non-dominated by $\vec{v}$, meaning that some elements of $\vec{u}$ are not larger than that of $\vec{v}$ and others are not smaller. Note that when $\vec{u}$ and $\vec{v}$ are identical, $Better(\vec{u},\vec{v}) = 0$, since $\vec{u} \not\preceq \vec{v}$ and $\vec{v} \not\preceq \vec{u}$. The *Better* function relations of the nodes in Fig. 1 are shown in Table 1. The *Better* function has the following properties.

Property 2. *If* $Better(\vec{u},\vec{v}) = 1$, *then* $Better(\vec{v},\vec{u}) = -1$. *If* $Better(\vec{u},\vec{v}) = -1$, *then* $Better(\vec{v},\vec{u}) = 1$. *If* $Better(\vec{u},\vec{v}) = 0$, *then* $Better(\vec{v},\vec{u}) = 0$.

Property 3. *If* $Better(\vec{u},\vec{v}) = 1$, $Better(\vec{v},\vec{w}) = 1$, *then* $Better(\vec{u},\vec{w}) = 1$.

Property 4. *If* $Better(\vec{u},\vec{v}) = 1$, $Better(\vec{v},\vec{w}) = 0$, *then* $Better(\vec{u},\vec{w}) \neq -1$.

The proof for these properties can be seen in Appendix A.2–A.4. Through these properties, one can deduce the relations among some vectors based on their known relations without having to do direct comparisons. Property 2 reflects the general reflexive property of Pareto dominance, indicating the lack of a need to compare $\vec{v}$ with $\vec{u}$ if $\vec{u}$ has been compared with $\vec{v}$. Property 3 reflects the transitive property of Pareto dominance, based on which a comparison between $\vec{u}$ and $\vec{w}$ can be avoided. Property 4 is also useful in the comparison process. In fact, there are nine possible relations for $Better(\vec{u},\vec{w})$ if $Better(\vec{u},\vec{v})$ and $Better(\vec{v},\vec{w})$ are known. Properties 3 and 4 only show two of these and they will be applied in this paper. Other relations and their applications can be seen in [20].

As for the second principle, an effective method should not only be able to avoid having to store unimportant relations, but should also be able to more easily access the Pareto optimal nodes in the current generation. More specifically, the method should satisfy the following criteria: (1) the relations among the nodes in the current Pareto optimal set should be preserved and these nodes should be easily accessible; (2) each node should have at least one relation to the others so as to avoid being an isolated node; (3) the 'middle' nodes should have the capability of preserving the three-valued relations. Clearly, a binary tree offers a good solution for these requirements. Fig. 2b demonstrates such a binary tree. As compared to Fig. 2a, Fig. 2b preserves the 'important' relations and reduces some unnecessary relations. As a consequence, a binary tree can effectively preserve most of the relation information among solution vectors while also requiring fewer comparisons. As this is a special binary tree, we have named it a dominance tree. This new concept can be defined in the following manner.

Definition 3 (*Dominance tree*). A dominance tree is a binary tree defined as follows:

Table 1
The *Better* relations among nodes in Fig. 1.

	N1	N2	N3	N4	N5
N1	0	1	0	0	0
N2	−1	0	−1	0	−1
N3	0	1	0	0	−1
N4	0	0	0	0	−1
N5	0	1	1	1	0

(1) A dominance tree is either an external node or an internal node connected to a pair of dominance trees, which are called the left sub-tree and the right sub-tree of that node.
(2) Each node in the dominance tree has four fields: id, count, child and sibling, where the id field registers which vector the node represents, the count field registers the size of its left sub-tree (including itself), the child field links to its left sub-tree whose root is dominated by that node, and the sibling field links to its right sub-tree whose root is non-dominated by that node.

The definition of the DT is similar to that of the BST, and they share some similar properties. For example, in both data structures, a node's child is the root of its sub-tree; correspondingly, the node is the parent of its child. In addition, if we consider that the dominated relation in a DT corresponds to the smaller than relation in a BST and that the non-dominated relation corresponds to the larger than relation in a BST, then a DT is similar to a BST. The exception here is the fact that the relations of nodes in a BST are two-valued, whereas they are three-valued in a DT. Two differences between a DT and a BST should be noted. First, the left and right sub-tree of a node in these two tree structures have different meanings. Furthermore, they have different properties. A detailed analysis of this point can been seen in Section 2.3.

A new concept sibling chain is defined in a DT. The sibling chain of a DT refers to the chain constituted by the DT's root node and its sibling nodes (it is also called the sibling chain of the DT's root node). Following a node's sibling field, one can obtain the node's sibling chain. Taking Fig. 2b as an example, N1 and N5 form a sibling chain of the DT whose root is N5. Similarly, N3 and N4 also constitute a sibling chain of the DT whose root is N3.

2.2. Dominance tree construction algorithms

Similar to a BST, the construction algorithms of a DT are also recursive. Unlike in a BST, when inserting a new node into a DT, there are three cases. If the new node is dominated by the root, it will be inserted into the left sub-tree of the root, which is similar to the smaller than relation in a BST. Similar to the larger than relation in a BST, if the new node is non-dominated by the root, it will be inserted into the right sub-tree of the root. Based on Property 4, if the new node dominates the root, the new node cannot be dominated by the nodes in the root's sibling chain. Thus, the nodes in the root's sibling chain may be either non-dominated by or dominated by the new node. In this case, the new node should first replace the root, and then the root is inserted into the left sub-tree of the new node. After this, the algorithm continues to compare the new node with the remaining nodes in the original root's sibling chain. If nodes that are dominated by the new node exist, they should be deleted from the sibling chain and then be inserted into the left sub-tree of the new node.

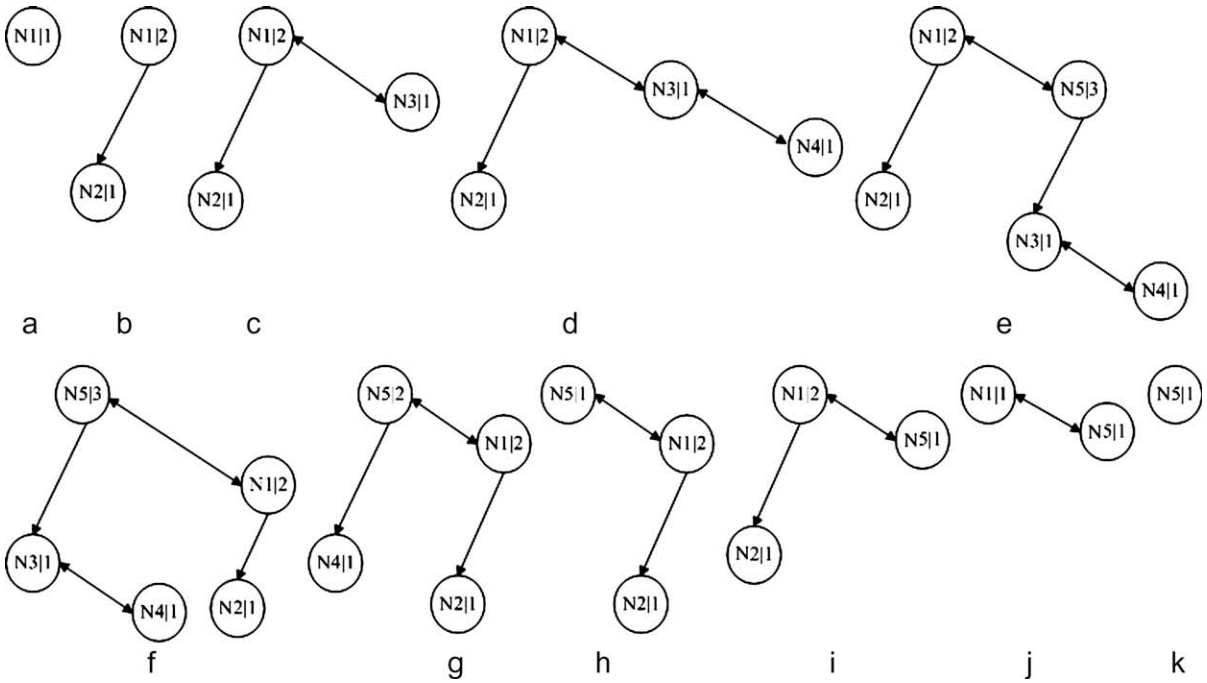

Fig. 3. An illustration of the constructing and deleting process of a dominance tree. The left number in the node is its id, and the right number is its count. (a–f) Illustrates the actions of inserting five nodes from N1 to N5. (f–k) Illustrates the actions of deleting the leftmost node. (f) and (i) Illustrates the actions of balancing the tree.

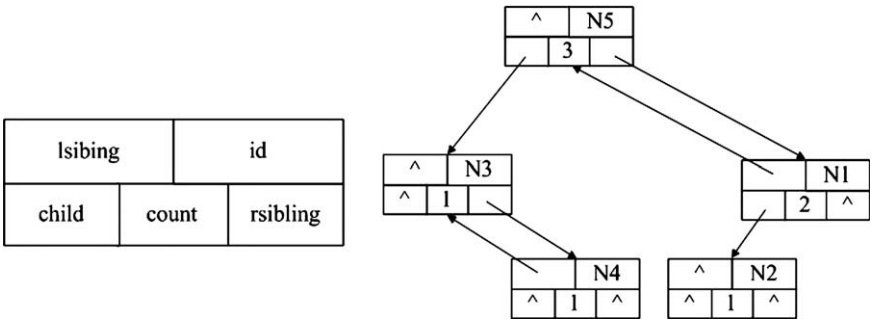

Fig. 4. An illustration of the relations among nodes in Fig. 1 with the data structure of a dominance tree.

Fig. 3a–f demonstrates the process of creating the DT that is initially depicted in Fig. 2b. The input order of these nodes is N1, N2, N3, N4 and N5, and the results after inserting each node are shown from Fig. 3a–e, respectively. Finally, the complete DT with five nodes inserted within it is shown in Fig. 3e. It is clear that the DT is unbalanced here, as the count of N5 (equal to 3) is larger than that of N1 (equal to 2) in Fig. 3e. To balance this tree, N5 with its left sub-tree moves along the sibling chain in the left direction, as shown in Fig. 3f; please note that, as there are only two nodes in the sibling chain of the DT, N1 and N5 only need to switch their positions. After the balancing operation, we can see that Fig. 3f is the same as Fig. 2b. It should be noted that, as a result of the balancing operation, the nodes in the same sibling chain are sorted in their count descending order. The balancing operation is desirable due to the following reasons. First, the operation can make a DT as balancing as possible without changing the relations of the nodes in the tree. Second, in a balanced DT, adding or deleting a node is less time-consuming (see Section 2.4). In addition, the count field of a node in a balanced DT implicitly contains the node's density information (see Section 3.2).

Based on the discussion above, we present the dominance tree construction algorithms in Algorithms 1–5. The data structure and algorithms are implemented with C programming language. Fig. 4 illustrates how to store the relations of the five nodes (N1–N5) with the data structure (note that these nodes are originally depicted in Fig. 2b). Please note that the sibling field of a node (say Node A) includes the lsibling and rsibling fields. The lsibling field of Node A links to its parent node that is non-dominated by it and its rsibling field links to its right sub-tree whose root is non-dominated by it. With these two fields, the sibling chain actually becomes a double link list, and thus traversing of the sibling chain is straightforward.

```
typedef struct DTNode{
  int id, count;
  struct DTNode *child;
  struct DTNode *lsibling, *rsibling;
} DTNode, *DTTree;
```

ConstructTree in Algorithm 1 is the main loop. *AddinTree* in Algorithms 2 and *AddinSibling* in Algorithm 3 insert a new node into a DT and guarantee that the DT is still valid after the operation. There are three choices in *AddinSibling*. When the inserted node (pNewnode) is dominated by an existing node (pChild), pNewnode is inserted into pChild's left sub-tree, as shown in Fig. 3b. When pNewnode is non-dominated by pChild, it is inserted into the right sub-tree (examples can be found in Fig. 3c and d). When pNewnode dominates pChild, the operation *DominatingAction* in Algorithm 4 is invoked. The main steps of *DominatingAction* include: (1) replacing pChild with pNewnode; (2) inserting pChild into pNewnode's left sub-tree; and (3) continuing to compare pNewnode with its right sibling nodes. In step 3, for any node in pNewnode's right sibling chain, if it is dominated by pNewnode, the node should first be deleted from the sibling chain and then be inserted into pNewnode's left sub-tree. Fig. 3e illustrates this situation. *BalanceTree* in Algorithm 5 implements the tree balancing operation and can guarantee that the nodes in the same sibling chain are sorted in the descending order of their counts. For example, if we compare the tree before the balancing operation in Fig. 3e and the tree after the balancing operation in Fig. 3f, we can see that N5 and N1 exchange their positions after balancing and that the counts of the nodes (N5 and N1) in the sibling chain in Fig. 3f are in the descending order. The same thing happens to the balancing operation depicted in Fig. 3h and i.

Algorithm 1. Creating a dominance tree. pTree is the root of the whole tree (for simplicity, pTree is only used as the root and it does not represent a vector).

Input: vectors set P.
Output: the dominance tree.

```
DTTree ConstructTree(VectorSet P) {
  foreach (pNewnode ∈ P)
```

```
    AddinTree (pTree, pNewnode);
  return pTree;
}
```

Algorithm 2. Insert a new node (pNewnode) into pRoot's left sub-tree when pNewnode is dominated by pRoot.

```
AddinTree(DTNode *pRoot, DTNode *pNewnode) {
  pRoot->count = pRoot->count + pNewnode->count;
  if(pRoot->child == NULL)
    pRoot->child = pNewnode;
  else AddinSibling(pRoot, pRoot->child, pNewnode);
}
```

Algorithm 3. Following Algorithm 2, when pNewnode is inserted into pParent's left sub-tree, pNewnode is compared with pChild, a node in the sibling chain of pParent's left sub-tree.

```
AddinSibling (DTNode *pParent, DTNode *pChild, DTNode *pNewnode){
  switch(Better(pNewnode, pChild))
  case 0: //non-dominated
    if(pChild->rsibling == NULL){
    pChild->rsibling = pNewnode;
    pNewnode->lsibling = pChild;
    }
    else AddinSibling(pParent, pChild->rsibling, pNewnode);
  case 1: //dominating
    DominatingAction(pParent, pChild, pNewnode);
    BalanceTree(pParent, pNewnode, L);
  case-1: //dominated
    AddinTree (pChild, pNewnode);
    BalanceTree (pParent, pChild, L);
}
```

Algorithm 4. The action to be taken when pNewnode dominates pChild, a node in the sibling chain of pParent's left sub-tree.

```
DominatingAction (DTNode *pParent, DTNode *pChild, DTNode *pNewnode) {
  /*pNewnode take the place of pChild*/
  pNewnode->rsibling = pChild->rsibling;
  pNewnode->lsibling = pChild->lsibling;
  if(pChild->lsibling != NULL)
    pChild->lsibling->rsibling = pNewnode;
  elsepParent->child = pNewnode;
  if (pChild->rsibling != NULL)
    pChild->rsibling->lsibling = pNewnode;
  pChild->lsibling = NULL;
  pChild->rsibling = NULL;
  /*insert pChild into pNewnode's left sub-tree*/
  AddinTree(pNewnode, pChild);
  /*pNewnode continues to compare with its right sibling nodes*/
  DTNode *p = pNewnode->rsibling;
  while (p){
    DTNode *q = p;
    p = p->rsibling;
    if(Better(q, pNewnode) == -1) {
    /*delete the dominated node q from the sibling chain*/
    if(q->lsibling != NULL)
      q->lsibling->rsibling = q->rsibling;
    if(q->rsibling != NULL)
```

```
      q->rsibling->lsibling = q->lsibling;
    q->lsibling = NULL;
    q->rsibling = NULL;
    /*add q into pNewnode's left sub-tree*/
    AddinTree(pNewnode, q);
    }
  }
}
```

Algorithm 5. Tree balancing operation. Sort the sibling chain of the pParent's left sub-tree in their count descending order. If the count of pMovenode (a node in the sibling chain) increases, it moves along the sibling chain to the left together with its left sub-tree, or else it moves to the right. *L* and *R* are integers that represent the direction of such a movement.

```
BalanceTree (DTNode *pParent, DTNode *pMovenode, int direction) {
  if(direction == L) {
    DTNode *p = pMovenode->lsibling;
    while(p && pMovenode->count > p->count){
      /*exchange the position of p and pMovenode*/
      pMovenode->lsibling = p->lsibling;
      p->rsibling = pMovenode->rsibling;
      pMovenode->rsibling = p;
      p->lsibling = pMovenode;
      if(pMovenode->lsibling == NULL)
      pParent->child = pMovenode;
    else pMovenode->lsibling->rsibling = pMovenode;
    if(p->rsibling != NULL)
      p->rsibling->lsibling = p;
    p = pMovenode->lsibling;
    }
  }
  if(direction == R) {
    DTNode *p = pMovenode->rsibling;
    while(p && pMovenode->count < p->count) {
      /*exchange the position of p and pMovenode*/
      pMovenode->rsibling = p->rsibling;
      p->lsibling = pMovenode->lsibling;
      pMovenode->lsibling = p;
      p->rsibling = pMovenode;
      if(p->lsibling == NULL)
        pParent->child = p;
      else p->lsibling->rsibling = p;
      if(pMovenode->rsibling != NULL)
        pMovenode->rsibling->lsibling = pMovenode;
      p = pMovenode->rsibling;
    }
  }
}
```

Having considered the construction algorithms, let us examine the deleting algorithm of the DT. Although more general algorithms for deleting an arbitrary node can be designed, we will only show a special case that deletes the leftmost node here, as it has some preferable features in EMO. On the one hand, the algorithm can be used to design an effective eliminating strategy for EMO (a detailed analysis of this can be seen in Section 3.2). On the other hand, the algorithm has less computational complexity as compared to other eliminating strategies (see Property 6 and Table 2). In most situations, starting from the root node, the leftmost node can be identified by continuing to trace the left sub-tree (i.e., the child field). Taking Fig. 3f as an example, the leftmost node of the DT is N3. In a special situation where all nodes are non-dominated, the DT becomes a list and thus the root node of the DT becomes the leftmost node (e.g., N1 in Fig. 3j). Algorithm 6 gives the deleting algorithm. Fig. 3f–k illustrates the process with an example which is a continuation of that used for illustrating the construction algorithms. In Fig. 3h, the DT becomes unbalanced again as a consequence of the deleting operation (the count of N5 (equal to 1) is smaller than that of N1 (equal to 2)). In order to balance the tree, N5, together with its left sub-tree (its left

Table 2
The average computational complexity of an individual evaluated in some popular MOEAs. M is the number of objectives, N is the population size.

Algorithm	Evaluating fitness	Niching strategy
SPEA2	$O(MN)$	$O(MN)$
NSGA-II	$O(MN)$	$O(M\ln N)$
NSGA-QS	$O(\log^{M-1}N)$	$O(M\ln N)$
NSGA-DT	$O(M\ln^2 N + MN/2^M)$	$O(M\ln N)$
DTEA	$O(M\ln^2 N + MN/2^M)$	$O(\ln N/M)$

sub-tree is NULL in Fig. 3h), moves along the sibling chain to the right as shown in Fig. 3i. Note that, while benefiting from the balancing operation, the deleting algorithm still preserves the count descending order of nodes within the same sibling chain.

Algorithm 6. Delete the leftmost node of a dominance tree.

```
Input: a dominance tree
Output: the leftmost node
DTNode *DeleteMostLeftNode(DTTree pTree) {
  DTNode *p = pTree->child;
  if(p == NULL)
    return NULL;
  else{
    pTree->count—;
    if(p->child == NULL) {
    pTree->child = p->rsibling;
    if(p->rsibling != NULL)
      p->rsibling->lsibling = NULL;
    p->rsibling = NULL;
    return p;
    }
    else return DeleteNode (pTree);
  }
}
DTNode *DeleteNode(DTNode *pRoot){
  DTNode *p = pRoot->child;
  DTNode *q = p->child;
  p->count—;
  if(q->child == NULL) {
    p->child = q->rsibling;
    if(q->rsibling != NULL)
    q->rsibling->lsibling = NULL;
    q->rsibling = NULL;
  }
  else q = DeleteNode (p);
  BalanceTree(pRoot, p, R);
  return q;
}
```

2.3. Properties of dominance tree

Lemma 1. *The sibling chain of a dominance tree contains and only contains all of the Pareto optimal nodes in the tree.*

Rationale. If there is only one node, it is clear that the lemma is true. According to the dominance tree construction algorithms, if the lemma is true for N nodes, there are three possibilities for a new node (pNode) to be inserted. (1) If pNode is dominated by the root, it is inserted into the left sub-tree of the root. (2) If pNode is non-dominated by the root, it should continue to be compared with the other nodes in the sibling chain of the DT. If it is dominated by one of them, it should be inserted into the left sub-tree of the first one. If it dominates a node in the sibling chain, the dominated node will be deleted from the sibling chain. If it is non-dominated by the remaining nodes in the sibling chain, it becomes a member of the sibling

chain. (3) If pNode dominates the root, the root is deleted from the sibling chain. In regard to the other nodes in the sibling chain, if they are dominated by pNode, these nodes are also deleted from the sibling chain. Therefore, the construction process guarantees that the sibling chain only contains all of the Pareto optimal nodes in the tree.

From Lemma 1, it is a straightforward process to derive a useful corollary.

Corollary 1. *The nodes in the sibling chain of a dominance tree are non-dominated.*

According to Lemma 1, the nodes in the sibling chain of a DT are the Pareto optimal nodes in the tree, so these nodes are non-dominated.

Lemma 1 and Corollary 1 show that the dominance tree construction algorithms can be used to find the Pareto optimal set of the population and that all of the Pareto optimal nodes are stored in the sibling chain of the tree. Since DT is a recursive definition, each node can be seen as a DT whose sibling chain conserves all of the Pareto optimal nodes in that DT.

Lemma 2. *The root of a dominance tree dominates all of the nodes in its left sub-tree.*

Rationale. If there is only one node, it is clear that the lemma is true. If this is true for N nodes, there are three possibilities for a new node (pNode). (1) If pNode is non-dominated by the root, it is inserted into its right sub-tree. (2) If pNode is dominated by the root, it is inserted to the left sub-tree of the root, so the lemma is still true. (3) If pNode dominates the root, the root and the other nodes in the root's sibling chain that are dominated by pNode should be inserted into the left sub-tree of pNode, so pNode dominates all of the nodes in its left sub-tree according to Property 3. The lemma is always true under the three conditions. Moreover, the *BalanceTree* operation does not change the nodes in the left sub-tree of a DT.

Lemmas 1and 2 are the major relations recorded in a DT. Since the DT reduces the comparisons among nodes, some relations among nodes are not directly shown in the tree. In a BST, the root is smaller than all of the nodes in its right sub-tree. However, due to reduced comparisons and the *BalanceTree* operation, a DT does not have this similar property as in a BST. In other words, in a DT the non-dominated relations between a root and the nodes in the root's right sub-tree may be violated. For example in Fig. 2b, N5 is not non-dominated by N2 which is in the right sub-tree of N5. Lemmas 1and 2 also show that the construction algorithms satisfy the two principles proposed in Section 2.1.

Lemma 3. *The root node of a dominance tree has the largest count value among the nodes in its sibling chain.*

Rationale. In the construction algorithms, the *BalanceTree* operation in Algorithm 5 sorts the nodes in the sibling chain in a count descending order once a node is inserted or deleted. So the root node of a DT always has the largest count value after *BalanceTree* operation.

Lemmas 3 guarantees that the DT is as balancing as possible. As a side benefit, the DT can get better average of the running time when a node is inserted or deleted.

2.4. Analysis of computational complexity

According to the *Better* function, as the comparison result of two vectors is three-valued, a vector can divide the objective space into three parts. This can be defined as follows.

Definition 5 (*Pareto domination space*). The objective space Ω is divided by a vector $\vec{u}_0$ into three parts: the Pareto dominating space Ω_{di}, the Pareto dominated space Ω_{dd} and the non-dominated space Ω_{dn}. $\Omega_{di} \bigcup \Omega_{dd} \bigcup \Omega_{dn} = \Omega$. And $\Omega_{di} = \{\vec{u} | \vec{u} \in \Omega \wedge Better(\vec{u}, \vec{u}_0) = 1\}$, $\Omega_{dd} = \{\vec{u} | \vec{u} \in \Omega \wedge Better(\vec{u}, \vec{u}_0) = -1\}$, $\Omega_{dn} = \{\vec{u} | \vec{u} \in \Omega \wedge Better(\vec{u}, \vec{u}_0) = 0\}$.

The objective space is divided into three parts by a vector. The Pareto dominated space is similar to the notion 'forbidden region' in [29]. As shown in Fig. 5, the two-dimension objective space is divided into three subspaces by node $\vec{u}_0$. We can

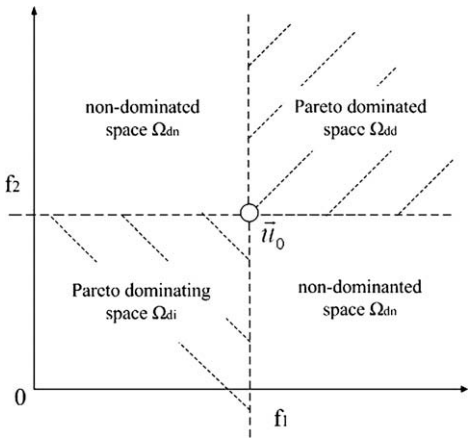

Fig. 5. An illustration of the Pareto dominating space in the two-dimension objective space.

observe that the probability that a random node (uniformly distributed in the objective space) falls in Ω_{dn} is larger than those in Ω_{di} or Ω_{dd}. In general, with the increase in the dimension of the objective space, the probability that a uniform random node appears in Ω_{di} or Ω_{dd} become smaller (a detailed analysis of this can be seen in the proof of Property 5 in the Appendix A.5). In a MOEA, a solution in Ω_{dd} should have a greater likelihood of being eliminated than the nodes in the other spaces as a dominated solution (dominated by $\vec{u}_0$). It is anticipated that less and less solutions exist in Ω_{dd} and more solutions exist in Ω_{di} or Ω_{dn} as the evolutionary process goes on. The reason for this is that the solutions in Ω_{di} are more prone to converge to the true Pareto front. Moreover, the solutions in Ω_{dn} are useful to maintain the diversity of the population and at the same time more preferable to the solutions in Ω_{dd} in terms of convergence to the Pareto front.

In order to analyze the computational complexity of constructing a DT, we need to calculate the expected number of comparisons when a node is inserted into a DT. The base operation is the comparison of two scalar numbers. If the newly inserted node always dominates the nodes that it is compared to, then this is the best case and the number of comparisons is M. If the newly inserted node is always non-dominated by the nodes that it is compared to, this is the worst case and the number of comparisons is MN (where M is the dimension of the vector, namely the number of objectives; N is the size of the DT). The average computational complexity is as follows.

Property 5. *The average computational complexity of inserting a new node into a dominance tree is* $O(Mln^2N + MN/2^M)$.

The proof for Property 5 can be found in Appendix A.5. From the computational complexity equation, we can see that more comparisons are required with the increase of N and M. Based on Property 5, we can infer that the average computational complexity of constructing a DT with the size of N is $O(MNln^2N + MN^2/2^M)$.

As for the deleting algorithm, the minimum number of searches for the leftmost node is 1 in the case when all nodes are non-dominated (i.e., the DT degrades into a list with the child fields of all nodes being NULL), and the maximum number of searches is N in the case when no nodes are non-dominated (i.e., the DT degrades into a list with the sibling fields of all nodes being NULL). The average computational complexity is as follows.

Property 6. *The average computational complexity of deleting the leftmost node is* $O(\ln N/M)$.

The proof for Property 6 can be found in Appendix A.6. We can see that the expected number of searches increases as N increases. However, as M increases, more nodes become non-dominated and the size of the DT's left sub-tree becomes smaller. As a consequence, it is more convenient to find the leftmost node. Therefore, the number of searches decreases here.

3. Applying dominance tree to evolutionary multi-objective optimization

As a data structure that can efficiently store the three-valued relations of vectors, DT can be an ideal structure for EMO. This paper demonstrates two cases. In the first case, we replace the non-dominated-sort process of NSGA-II with the fitness assignment strategy based on DT. In order to further explore the potential of DT, in the second case, we discuss a MOEA design based on DT.

3.1. DT-improved NSGA-II

The process of creating a DT can actually be regarded as a fitness assignment process, because a DT can preserve the dominating relations of solutions in the whole population naturally and dynamically. In NSGA-II, a non-dominated-sort approach is applied for creating the Pareto ranks that divide solutions into different fronts with different ranks. There exist solutions in the lower front dominating those in the higher front, but the inverse is not the case [9]. In SPEA2, each individual in both the main population and the elite archive is assigned a strength value. On the basis of the strength value, the final rank value is determined by the summation of the strength values of the individuals that dominate the current individual [33]. Fig. 6

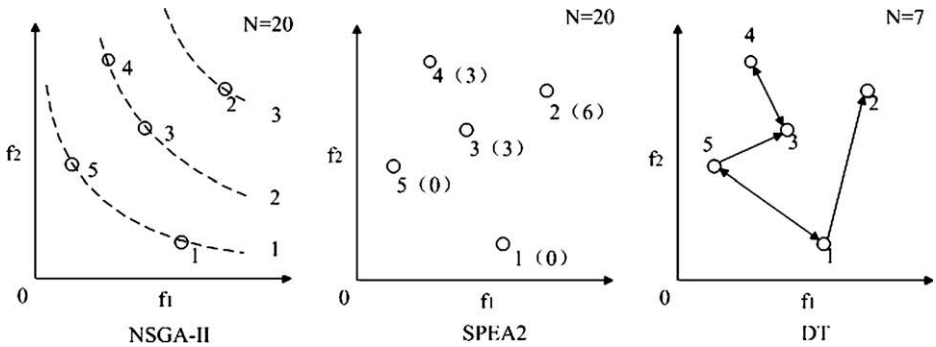

Fig. 6. An illustration of the different fitness assignments in the objective space. The number in top right corner of each figure is the number of comparisons in each fitness assignment.

illustrates these three different fitness assignment strategies in the objective space. The number of comparisons among solutions in each fitness assignment process is also shown in the figure. We can see that the number of comparisons in the DT is much smaller than that in NSGA-II and SPEA2.

As a well-known MOEA, NSGA-II performs well in many test functions and in real applications [9]. The processing time of the non-dominated-sort procedure in NSGA-II is $O(MN^2)$, because the procedure involves comparing the objective values of each solution with those of all other solutions in the population [9]. Jensen has presented a fast way to do the non-dominated-sort process, which presents a sweep line algorithm for the two-objective case and a 'divide-and-conquer' algorithm for three or more objectives [14]. Jensen's approach realizes the same function of the non-dominated-sort process with the different algorithms that have less time complexity.

To validate the applicability of DT in MOEA, we replace the non-dominated-sort process in NSGA-II with the fitness assignment strategy based on DT. In order to have a fair comparison, we keep all the other settings of NSGA-II unchanged and meanwhile tune the DT to accommodate those settings in NSGA-II. In particular, the nodes in the DT are also assigned rank values as their 'fronts'.

Definition 6. (Rank Assignment in DT). The ranks of nodes in a dominance tree can be assigned by the following steps:

(1) The root's rank is 1.
(2) If a node's rank is k, its child node's rank is $k+1$; and its sibling node's rank is k.

A node's rank value will be used as its front. The rank assignment can be realized either with the depth-first search algorithm or breadth-first search algorithm. The computational complexity of the two algorithms are both $O(N)$. The assignment approach is similar to Fonseca and Fleming's approach in which the rank of a solution is assigned the number of solutions associated with the current population dominating it [12]. However, influenced by the tree structure, the rank of a node in a DT may be smaller than the actual number of solutions dominating it. Fig. 7 demonstrates the ranks of the five nodes (originally given in Fig. 6) that are assigned based on the DT-based assignment strategy. In the figure, we can see that N2 is dominated by three nodes: N1, N3 and N5, whereas its rank is 2 in the tree. In some situations, we may find that the front of a node assigned by DT may differ from the front of the same node that is assigned by the non-dominated-sort process in NSGA-II. For example, in the right figure of Fig. 6, the front of N2 assigned by DT is 2, whereas for the same node, the front assigned in NSGA-II is 3, as shown in the left figure of Fig. 6.

3.2. Dominance tree based evolutionary algorithm

To fully utilize the properties of DT, the second case presents a DT based MOEA, namely Dominance Tree based Evolutionary Algorithm (DTEA). This algorithm is described in Algorithm 7. It first randomly generates an initial population and then goes on to create a DT with the individuals in the population via the construction algorithms (see Algorithms 1–5). In each iteration, a newly generated individual will be inserted into the DT and the leftmost node will be deleted from the DT. This iteration is repeated until the stopping criterion is satisfied. As a flexible algorithmic framework described in Algorithm 7, DTEA can employ almost all of the existing crossover and mutation operators for the generation of new individuals; it can also generate or delete one or more individuals in one iteration. For example, Shi et al. have applied the multi-parent crossover operator to generate one individual in one iteration [19] and the algorithms in [21] control the individual generation and deletion to maintain all of the Pareto optimal individuals. In this paper, we employ SBX and polynomial mutation [7] to generate and delete two individuals in one iteration.

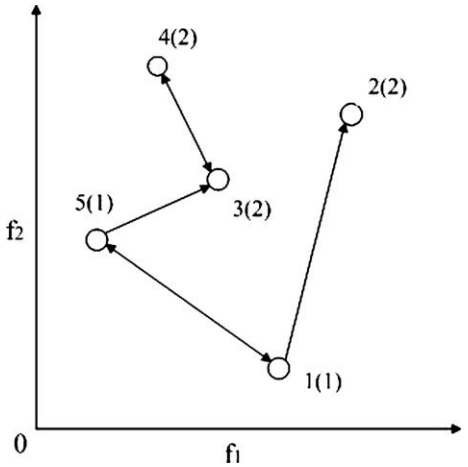

Fig. 7. The rank of the nodes in Fig. 1 using the rank assignment in dominance tree. The number in the bracket is its rank value.

Algorithm 7. The main loop of DTEA. P stands for the population, T stands for the dominance tree, c stands for the new generated child, w stands for the deleted node and t stands for the number of generations.

```
DTEA {
  generate the initial population P_0 at random;
  T_0 = ConstructTree (P_0);
  set t = 0;
  while (stopping criterion is not satisfied){
    generate child c_t from P_t;
    AddinTree (T_t, c_t);
    delete the leftmost node w_t = DeleteMostLeftNode (T_t);
    P_{t+1} = P_t − {w_t} + {c_t};
    set t = t + 1;
  }
  return P_t;
}
```

From Algorithm 7 above, we can observe that the eliminating strategy in DTEA deletes the leftmost node of a DT in each iteration. Its rationality comes from Lemmas 3. According to this lemma, unless all nodes are non-dominated, the leftmost node in the DT is always dominated by a relatively large number of nodes; thus, deleting the leftmost node is consistent with the essence of EA: survival of the fittest. In addition, by always deleting the leftmost node, DTEA can maintain the diversity of the population implicitly. A node with a big count means that it dominates many nodes in the objective space, namely that its Pareto dominated space is more 'crowded', so the nodes in its Pareto dominated space should be more likely to be eliminated. As a consequence, less and less solutions exist in the dominated space and more solutions exist in the dominating or non-dominated space as the evolutionary process goes on. Please note that the nodes in the dominating space are more prone to converge to the true Pareto front and that the solutions in the non-dominated space are useful to maintain the diversity of the population. Thus, this eliminating strategy enables DTEA to preserve better individuals towards the two goals that were discussed at the beginning of this paper. In summary, although DTEA does not use any explicit diversity maintenance strategy as in most existing MOEAs, the DT implicitly contains the density information in the count field.

Benefiting from the favorable properties of the DT, DTEA has three unique features. First, by taking advantage of the fact that the DT not only preserves the dominating relations of solutions, but also implicitly contains the density information, DTEA realizes the two goals of MOEA (i.e., convergence and diversity) with one integrated DT-based strategy. Second, the DT-based eliminating strategy in DTEA (i.e., eliminating only the leftmost node in one generation) realizes elitism without any extra effort. Third, because of the above two beneficial features, as compared to traditional MOEAs, DTEA is simpler and more flexible to implement and has fewer parameters.

3.3. Comparison with other data structures in MOEAs

Several special data structures have been applied to MOEAs for performance enhancement. It is necessary to compare the DT with these data structures. As a data structure for preserving dominating information among individuals, DT is different from such data structures in a number of aspects. Mostaghim et al. used Quad-tree to store the Pareto-points of the archives [16]. Unlike the DT, which is a binary tree, Quad-tree is a m-ary tree. In addition, as the nodes in a Quad-tree are all non-dominated by each other, it cannot preserve the dominating or dominated information among individuals. Fieldsend et al. proposed another DNT structure to avoid the linear comparison in the elite set for every newly inserted individual [11]. It is coincidental that DNT and DT are both binary trees that aim to utilize the properties of BST to reduce unnecessary comparisons. One major difference here is that DNT transforms the original individuals into composite points such that the composite points are in partial order, and then the properties of BST can be used. However, the DT and its construction algorithms promise that the DT's structure itself can contain the three-valued relations among solutions directly without the need for composition or transformation. More importantly, the purpose of and applications for the three structures are different. DNT and Quad-tree store and sort the individuals only in the elite archive. Whereas, DT can preserve the dominating information of the whole population and can be directly applied for fitness assignment. In addition, a DT can also be used for diversity maintenance.

Alberto and Mateo proposed an IDG structure to maintain the relations among the individuals in the whole population [1]. Similar to DT, IDG also strives to reduce redundant comparisons by utilizing the transitive property of Pareto dominance. However, they do differ in several aspects. IDG is a directed graph containing two-valued relations (i.e., dominating and dominated relations). Whereas, a DT is a binary tree containing the three-valued relations. Furthermore, a DT has a lower computational complexity and is more efficient than IDG when inserting or deleting a node.

Among these data structures, the most similar work to that of DT is probably that of the Pareto tree, proposed by Chen [4]. Similar to the DT, the Pareto tree can integrate the convergence and diversity strategies into the data structure and maintain diversity without any special strategies. However, distinct from the DT, which preserves the dominating information of the

whole population, the Pareto tree only contains the non-dominated population at the current leaf level. Moreover, so far no literature indicates that the algorithms with the Pareto tree are more efficient than other MOEAs.

Jensen has systematically analyzed the computational complexity of some contemporary MOEAs and has proposed several efficient algorithms to reduce their computational complexity [14]. Let us compare the average computational complexities of the DT-based algorithms with other MOEAs. We select two famous MOEAs, namely SPEA2 and NSGA-II, as benchmarks for this comparison. We call the DT-improved NSGA-II NSGA-DT and Jensen's improved version of NSGA-II NSGA-QS. In order to ensure that there is a fair comparison here, we employ an 'individual based evaluation' method. In other words, the average computational time for evaluating one new solution is compared in these algorithms. Thus, as there are N individuals evaluated in a generation, the average computational complexity of MOEAs discussed in [14] should be divided by N. The comparative results are shown in Table 2. (The results of NSGA-II and SPEA2 are taken from Tables 1 and 2 in [14], respectively.) Since a new density estimation metric is used in NSGA-II, the computational complexity of NSGA-II is less than that of SPEA2 in regard to the niching strategy. Jensen's algorithm (i.e., NSGA-QS) speeds up the fitness evaluation of the original NSGA-II. Among the five algorithms, DTEA has the least computational complexity for the implementation of the niching strategy. In addition, it also significantly speeds up the fitness evaluation process. We will compare their running times further in the following section using numerical experiments.

4. Experiments

The experiments focus on the comparison of effectiveness and efficiency of MOEAs. Therefore, the experiments include two parts. In part one (i.e., comparison of performance), the candidate algorithms are compared in terms of both the quality of results and the running time. In part two (i.e., comparison of running time), we further compare their running times by varying the population size and the number of objectives. The algorithms to be compared in the experiments are NSGA-II, NSGA-QS, SPEA2, NSGA-DT and DTEA. The experiments are carried out on a 3GHz and 512MRAM Pentium IV computer running Windows 2000.

4.1. Comparison of performance

Five two-objective functions and four three-objective functions will be used in the performance experiments. The algorithms will be compared using three criteria: convergence to the true Pareto front, maintenance of diversity and running time. Each of the algorithms runs for 50 times to obtain reliable results. In this subsection, the algorithms to be compared are SPEA2, NSGA-II, NSGA-DT and DTEA. Since NSGA-QS only speeds up NSGA-II and does not improve the quality of solutions, it will not be included in this subsection.

4.1.1. Test functions and performance assessment

We first describe the test functions to be used to compare the different MOEAs. The first five test functions: ZDT1, ZDT2, ZDT3, ZDT4 and ZDT6, proposed by Zitzler et al. [32], each have two objectives. These test functions have appeared in many studies [9,19,29]. Test functions DTLZ1–DTLZ4 proposed in [10] have also been widely used by many researchers as functions with three-objectives [14,22,29].

For a fair comparison of these four algorithms, they all use real-code, simulated binary crossover (SBX) and polynomial mutation [7]. The same parameters are used for the four algorithms. Please note that no effort is made to find the best parameter settings for the DT-based algorithms. The population size is 200 and the archive size is 200. The distribution indexes for crossover and mutation operators are η_c and η_m, respectively, and they are both 20. The crossover probability is 1 and the mutation probability is $1/n$ for all test functions, where n is the number of variables. The number of individuals to be generated and evaluated are settled as follows: 25,000 for ZDT1–ZDT6, 40,000 for DTLZ1–DTLZ4. The generations of SPEA2, NSGA-II and NSGA-DT are equal to the number of the total evaluated individuals divided by the population size. Because the SBX operator used in DTEA only generates two new individuals and DTEA deletes two individuals in one generation, the generations of DTEA are equal to the number of the evaluated individuals divided by two. In DTEA, the parent individuals are randomly selected from the current population.

Measure criterion is an important issue in multi-objective optimization [34] and this study employs two popular measure criteria. The first metric Υ measures the extent of convergence to a known set of Pareto optimal solutions [9,27]. The method first identifies a set of uniformly spaced solutions (P^*) from the true Pareto optimal front in the objective space and then calculates the minimum Euclidean distance from each solution in the obtained optimal set (Q) to the chosen solutions in P^*. The average of these distances is finally used as the metric (see Eq. (1)). For the test functions from ZDT1 to DTLZ4, the numbers of the uniformly spaced solutions are 500, 500, 537, 500, 500, 820, 1062, 1062 and 1062, respectively. The smaller Υ is, the better the convergence toward the true Pareto optimal front is

$$\Upsilon = \frac{\sum_{i=1}^{|Q|} d_i}{|Q|} \quad d_i = \min_{k=1}^{|P^*|} \sqrt{\sum_{m=1}^{M} \left(f_m^{(i)} - f_m^{*(k)}\right)^2}. \tag{1}$$

Deb has suggested the following metric Δ (see Eq. (2)) to measure the extent of spread achieved among the solutions obtained [8,27]. In the metric, d_i is the Euclidean distance between neighboring solutions, while $\bar{d}$ is the mean value of these

distance measures. The parameter d_m^e is the minimum Euclidean distance between the extreme solution of P^* in the mth objective function and the solutions in Q. Deb et al. have used the metric for the test function with two objectives [9]. Generally, the metric Δ takes a smaller value with a better distribution of solutions within the extreme solutions.

$$\Delta = \frac{\sum_{m=1}^{M} d_m^e + \sum_{i=1}^{|Q|} |d_i - \bar{d}|}{\sum_{m=1}^{M} d_m^e + |Q|\bar{d}} \tag{2}$$

4.1.2. Results and discussion

Figs. 8 and 9 summarize the experimental results of the four algorithms in the nine test functions in terms of the two respective performance measures. The distribution of results for each performance measure are represented by box plots [3], a tool that can efficiently visualize the distribution of simulation data [26,31,32]. In a box plot, the horizontal line within the box represents the median and the upper and lower ends of box are the upper and lower quartiles. '+' appendages represent the outliers. In the following discussion, we first present the experimental results and then go on to explain the reasons for such.

As shown in Fig. 8, the four algorithms perform in a similar manner in regard to the convergence metric Υ on most test functions and all can approximate the optimal front. It can be observed that the median values of DTEA are the smallest in

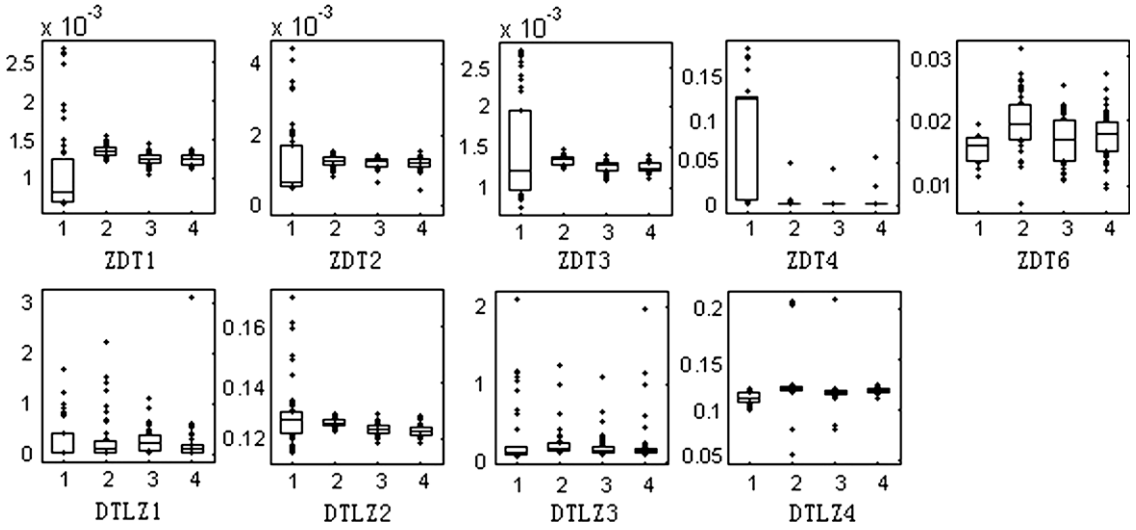

Fig. 8. Box plots based on the measures of metric Υ. 1–4 represent DTEA, SPEA2, NSGA-II and NSGA-DT, respectively.

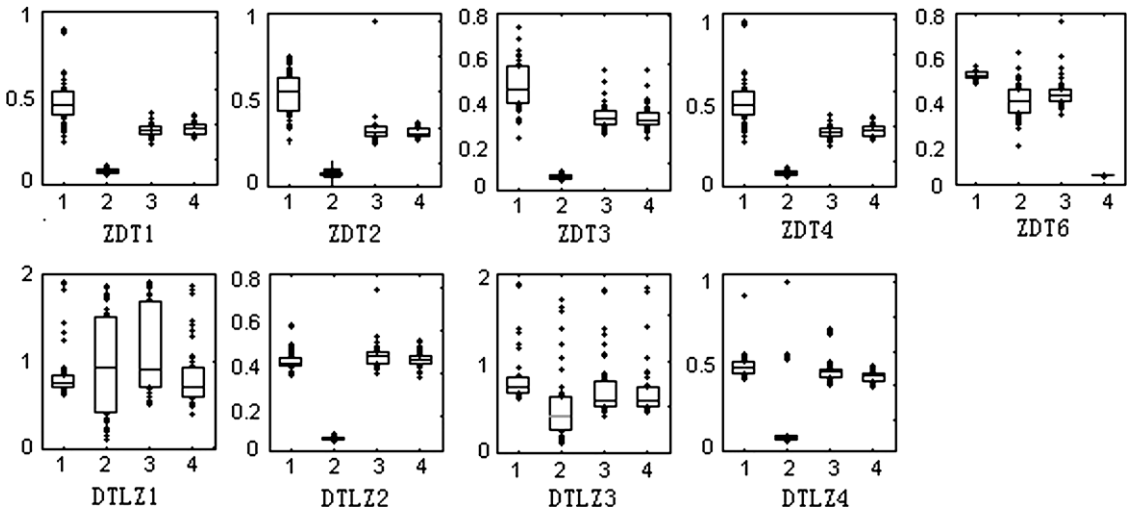

Fig. 9. Box plots based on the measures of metric Δ. 1–4 represent DTEA, SPEA2, NSGA-II and NSGA-DT, respectively.

ZDT1, ZDT2, ZDT3, ZDT6, DTLZ1, DTLZ3 and DTLZ4. For most functions, NSGA-II and NSGA-DT are slightly better than SPEA2. In all cases, NSGA-DT has similar results to NSGA-II. In ZDT1–ZDT4, the distribution of results in DTEA is wider than those seen in the other algorithms. However, in ZDT6-DTLZ4, we do not see any apparent difference in the distribution of results of the four algorithms.

Fig. 9 shows the result of the distribution metric Δ for the different algorithms. For most functions, SPEA2's distribution is the best one. NSGA-II and NSGA-DT also have very close distribution except for ZDT6. Although with respect to the functions with two objectives, DTEA's distribution is worse than those of the other algorithms, this is not the case for the functions with three objectives, where DTEA, NSGA-II and NSGA-DT have very similar results. Thus, compared to other algorithms, we can consider that the DT-based MOEAs may have an improving distribution (i.e., diversity of solutions) as the number of objectives increases. In order to verify this tendency, we further perform a diversity comparison experiments for a different number of objectives with the scalable test function DTLZ1 that has been illustrated in Section 4.2. The numbers of objectives are 2, 3 and 4, respectively, and the evaluated individuals are 20,000, 40,000 and 80,000, respectively. The running generations are 200. The metric Δ is also used to measure the diversity. The numbers of the uniformly spaced solutions for the three cases of objectives are 500, 1275 and 1676, respectively. Table 3 shows the comparison results for 50 runs. Compared to other algorithms, DTEA has a growing advantage in regard to the diversity maintenance as the number of objectives increases. NSGA-DT also obtains better diversities than NSGA-II. The reason for this is assumed to be that with the growth in the number of objectives, the non-dominated solutions rapidly increase. In this situation, the DT may be superior for managing the population as compared to other approaches.

Table 4 shows the average running time of the four MOEAs. The running time of DTEA is much less than those of the other algorithms in all test functions, SPEA2 is the slowest one, and NSGA-DT is significantly faster than NSGA-II. In summary, the two algorithms based on DT, namely DTEA and NSGA-DT, are significantly faster than the other two well-known algorithms.

For all of the nine test functions, SPEA2 performs best at maintaining diversity. However, its running time is significantly longer than the other algorithms. The reason for this is assumed to be that the density estimation used in SPEA2, the kth nearest neighbor method, while being an effective strategy for maintaining diversity, is also quite time-consuming. NSGA-II and NSGA-DT have very similar convergence and diversity. The reason is that NSGA-DT only replaces the non-dominated-sort process of NSGA-II with the DT, so it has a minor effect on the performance of NSGA-II. On the other hand, NSGA-DT is significantly faster than NSGA-II, due to its being enhanced by the DT. This control experiment demonstrates the time-saving feature of the DT-based individual evaluation and fitness assignment strategy. Compared to SPEA2 and NSGA-II, DTEA has slightly better convergence and at the same time slightly worse diversity for most functions. Although the diversity of DTEA is worse than the other algorithms in the functions with two objectives, its Δ values in ZDT1–ZDT6 are still smaller than 1, so the diversity is also acceptable. Please note that the diversity of DTEA is maintained by the DT implicitly without any special diversity maintenance strategy. It should also be noted that no attempts are made to design specific strategies for the DT-based methods, for example, no special operators, diversity strategy or selection of best parameters for the DT. From the experiments, it can be observed that for two-objective functions, the distribution of results in DTEA is wider than that in the other MOEAs, showing that DTEA may not be as stable as the other two algorithms in regard to two-objective functions. Please note that the DT's structure is dependent on the input sequence of nodes. As a result, the selection process of DTEA is influenced not only by non-domination ranks and density information, but also by the sequence of nodes. The sequence of nodes as a random factor may cause some small perturbations to DTEA.

As a summary of the performance experiments, it can be seen that the DT-based algorithms (i.e., DTEA and NSGA-DT) obtain competitive solutions with significantly less computing time as compared to the two popular MOEAs: NSGA-II and SPEA2. Therefore, we consider the DT to be an effective data structure that can significantly speed up MOEAs while not compromising their performance.

Table 3
The diversity comparison for the different number of objectives. M represents the number of objectives. The first column for each algorithm represents the mean value of metric Δ, and the second column is the variance value.

M	DTEA		SPEA2		NSGA-II		NSGA-DT	
2	0.9325	0.2068	0.2208	0.3153	0.4816	0.2511	0.4781	0.2390
3	0.8101	0.1497	0.6165	0.4884	0.7768	0.3591	0.7472	0.3705
4	0.8875	0.2342	0.9302	0.1583	0.9184	0.1654	0.9081	0.1828

Table 4
Compare running time of different algorithms. The time unit is millisecond.

	ZDT1	ZDT2	ZDT3	ZDT4	ZDT6	DTLZ1	DTLZ2	DTLZ3	DTLZ4
DTEA	1122	1059	1126	307	268	1040	2260	844	2303
SPEA2	14,595	13,773	14,290	12,043	10,156	26,399	29,619	26,382	30,051
NSGA-II	3504	4518	3463	2227	2234	4745	6046	4808	7137
NSGA-DT	2806	1809	2655	2186	1115	3042	5892	3064	6141

4.2. Comparison of running time

In this subsection, we further compare the running time of different MOEAs by varying the population size and the number of objectives. In the following experiments, we exclude SPEA2, since SPEA2 is evidently slower than the other algorithms as shown in Section 4.1. Thus, the algorithms to be compared here are NSGA-II, NSGA-QS, NSGA-DT and DTEA.

$$
\begin{aligned}
\text{Minimize } & f_1(\vec{x}) = \frac{1}{2}x_1x_2\cdots x_{M-1}(1+g(\vec{x}_M)) \\
& f_2(\vec{x}) = \frac{1}{2}x_1x_2\cdots(1-x_{M-1})(1+g(\vec{x}_M)) \\
& \vdots \\
& f_{M-1}(\vec{x}) = \frac{1}{2}x_1(1-x_2)(1+g(\vec{x}_M)) \\
& f_M(\vec{x}) = \frac{1}{2}(1-x_1)(1+g(\vec{x}_M)) \\
\text{where } \quad & g(\vec{x}_M) = 100\left(|\vec{x}_M| + \sum_{x_i\in\vec{x}_M}((x_i-0.5)^2 - \cos(20\pi(x_i-0.5)))\right) \\
& 0 \leqslant x_i \leqslant 1, i = 1,\cdots,n
\end{aligned}
\tag{3}
$$

DTLZ1 is used as the test function (see Eq. (3)) here, because it is a standard multi-objective benchmark function and the number of objectives can be modified [10]. In addition, it takes very little time to calculate the objective functions and to execute its genetic operators, which means that the difference in running times should mainly be attributed to different comparison mechanisms. The number of decision variables is $|\vec{x}| = M + k - 1$, where k is a parameter. The notation $\vec{x}_M$ is used to denote the last k elements of $\vec{x}$. As suggested in [10], k is set to 5 in the experiments. The results that will to be shown are the average values in 10 runs. The experiment parameters are set as follows: $\eta_c = 20$ and $\eta_m = 20$. The crossover probability is 1 and the mutation probability is 0.1. The number of objectives (i.e., M) are 2 and 5, respectively. Ten different population size settings are used, ranging from 50 to 1400. The number of generations is 50 for NSGA-DT, NSGA-QS and NSGA-II. For a fair comparison, the generation of DTEA is calculated as $50N/2$ (where N is the population size), as DTEA only generates and deletes two individuals in one generation.

Fig. 10 shows the results in a logarithmic scale. The results for the case with two objectives are demonstrated in the figure on the left in Fig. 10. In examining this figure, it is clear in all cases that DTEA is the fastest one and NSGA-II is the slowest one. NSGA-QS is faster than NSGA-DT. As the population size increases, the difference among the four algorithms becomes more significant. All four of the lines seem to be approximately linear. Since the plots are in a log scale, this indicates that the processing time follows a $T = \beta N^{\alpha}$ relation. The parameter α can be estimated with linear regression. For NSGA-II, α is 1.91, which means NSGA-II has approximately a N^2 growth in running time. For NSGA-QS, α is 1.19, and it is 1.34 for NSGA-DT. For DTEA, α is 1.01 which is very close to linear growth.

The experimental results of the case with five objectives are demonstrated in the figure on the right in Fig. 10. In all of the cases, DTEA is still the fastest one for the different populations. When the population size is small, the running times of NSGA-DT and NSGA-QS are longer than that of NSGA-II. As the population size increases, NSGA-QS and NSGA-DT become faster than NSGA-II at the points when the population sizes are larger than 200 and 800, respectively. The linear regression analysis shows that α in NSGA-II, NSGA-QS and NSGA-DT are 1.89, 1.39 and 1.53, respectively. The trend of DTEA does not

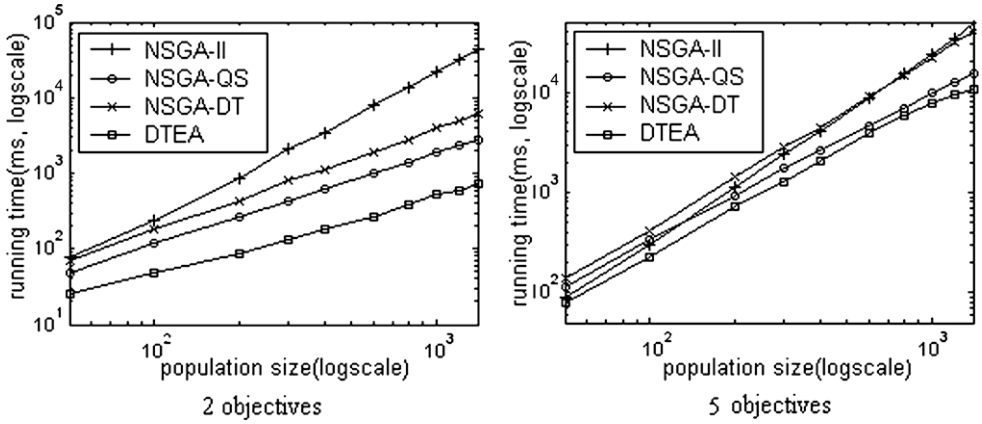

Fig. 10. The processing times of the four algorithms on the DTLZ1 problem for two and five objectives.

seem to be linear, especially when the population is quite large (e.g., larger than 800). This shows that an increase in the running time of DTEA is less than that of the other three algorithms as the population size increases. In summary, DTEA is always the fastest one and NSGA-DT runs faster than NSGA-II when the population size is considered to be large.

From the experiments, it can be seen that the advantages of NSGA-DT over NSGA-II are not as significant as in the other cases when the number of objectives is large and, at the same time, the population size is small. The reason for this is that when the number of objectives is large, more individuals in the population tend to be non-dominated. In this situation, a new non-dominated individual should be compared to each of the non-dominated individuals in the tree, leading NSGA-DT to need more time in initializing the DT. Many other approaches also face the same difficulty, for example IDG [1], Quad-tree [16], Pareto tree [4] and the fast algorithms proposed by Jensen [14]. Compared to these approaches, DTEA has the advantage of having a running time that is less sensitive to the number of objectives. When the non-dominated solutions become large, it does take more time for DTEA to insert a new non-dominated individual. However, under the same conditions, the size of the DT's left sub-tree becomes smaller. Thus, it takes less time for DTEA to delete the leftmost node. This is one reason why the number of objectives has less effect on DTEA. It should be noted that for a fair and strict comparison, the most efficient versions of SPEA2 and NSGA-II were performed. Since DTEA only generates and deletes two individuals in one generation, it can be considered to be a steady-state algorithm (see [17] for the definition of steady-state algorithms). If the comparison had been between DTEA and the steady-state versions of NSGA-II and SPEA2, DTEA would have shown an even more significant time-saving advantage, since more comparisons and generations are required by the steady-state versions of NSGA-II and SPEA2.

5. Improvements

As presenting the DT as a general data structure for MOEAs, we do not discuss any specific implementation strategies that can improve the performance of the DT-based algorithms in this paper. As illustrated in the performance experiments, the DT-based algorithms were not found to be obviously better than SPEA2 and NSGA-II in terms of the quality of results, although they were generally much faster. There are several possible ways to further enhance the performance of the DT-based algorithms (e.g., DTEA). Recent work has shown that the restricted number of solutions in the elite archive can result in shrinking and oscillating/retreating the estimated Pareto front [33]. For this reason, Fieldsend et al. highlighted the use of an active elite archive to improve the optimization speed of these algorithms [11]. In DTEA, the elite archive (i.e., the sibling chain of the DT) is active prior to the first time that all of the nodes become non-dominated. After this, it may be restricted (i.e., the elite size is equal to the population size). It is effortless to make the elite archive unconstrained in the whole evolutionary process by designing a strategy to control the population's growth and decline. Moghaddam et al. proposed an adaptive elite archive updating process by saving the dissimilar solutions when the maximum archive size is exceeded [15]. Yen and Lu have designed the population's growth and decline strategies to determine if an individual will survive or be eliminated based on some qualitative indicators [29]. Similar strategies can also be used in DTEA [21].

As for diversity maintenance, it is possible to add a density estimator into DTEA [21]. In DTEA, the nodes in the sibling chain are sorted according to their count values. We can evaluate the density values of the nodes in the same sibling chain and sort them, first according to their count values, then by their density values. Because it only compares the density values of the nodes in the same sibling chain, the computational complexity of the improved algorithm is still lower than that in many MOEAs.

6. Conclusion

This paper introduced a general data structure-dominance tree (DT) that can store the three-valued relations of vectors and also discussed its applications in EMO. The DT can be characterized as a binary tree: (1) storing three-valued relations: dominating, dominated and non-dominated; (2) maintaining only the necessary relations among vectors, and thus significantly reducing unnecessary comparisons; and (3) implicitly containing the density information of vectors. To demonstrate the potential applications of DT as an effective fitness assignment strategy in EMO, we replaced the non-dominated-sort process of NSGA-II with a DT-based method. The experiments showed that while the DT version of NSGA-II did not compromise the original NSGA-II's performance, it achieved a much faster running speed than the original version. In addition, this paper further presented DTEA, a MOEA specially built on the favorable properties of DT. In DTEA, a single strategy based on DT realizes both the convergence and diversity goals in MOEA. Moreover, the DT-based eliminating strategy not only maintains the diversity naturally, but also realizes elitism without any extra effort. In the performance experiments, DTEA showed its potential in producing statistically competitive results compared with SPEA2 and NSGA-II in the nine popular multi-objective optimization benchmark functions. In addition, DTEA was significantly faster than SPEA2, NSGA-II and the improved version of NSGA-II by Jensen.

As we know, EAs may suffer from premature convergence if the population size is too small, and therefore a large population is usually desirable. However, large population sizes are often infeasible in real world applications due to the high computational complexity of the existing fitness assignment methods and likely also due to the possible convergence slowdown caused by population sizes. Thus, we believe that a fitness assignment strategy based on a DT can both directly contribute to the reduction of computational complexity and indirectly contribute to the enhancement of solution quality, since the increasing population sizes in the DT-based methods result in a lower incremental rate of running time compared with most existing fitness assignment methods.

This paper addresses the properties, effectiveness and efficiency of a DT for EMO. However, the application of a DT may not be limited to EMO. As a general data structure, a DT may be suitable for many applications that require efficient storage of three-valued information (e.g., relations similar to dominating, dominated and non-dominated in multi-objective optimization).

Acknowledgements

The authors are grateful to the editors and referees for their comments, which have greatly improved the presentation of this paper. This work is supported by the National Science Foundation of China (No. 60402011, 60775035, 60805041), 863 National High-Tech Program (No. 2007AA01Z132), National Basic Research Priorities Programme (No. 2003CB317004, 2007CB311004) and National Science and Technology Support Plan (No. 2006BAC08B06, 2006BAH03B05). We also wish to thank Jeremy Schiff for his useful suggestions.

Appendix A

A.1. Proof of Property 1

Let $\vec{u},\ \vec{v},\ \vec{w}$ are three vectors and $\vec{u}=\{u_1,\ldots,u_n\},\ \vec{v}=\{v_1,\ldots,v_n\},\ \vec{w}=\{w_1,\ldots,w_n\}$. If Pareto dominance is not anti-reflexive, there exits $\vec{u}$ and $\vec{u}\preceq\vec{u}$. According to the definition of Pareto dominance, there exits u_i and $u_i<u_i$, but this is impossible. So Pareto dominance is anti-reflexive.

If $\vec{u}\preceq\vec{v}$ and $\vec{v}\preceq\vec{w}$, then $u_i\leqslant v_i$ and $v_i\leqslant w_i$ $(i=1,\ldots,n)$, so $u_i\leqslant w_i$. Because there exits u_j and $u_j<v_j$, $u_j<w_j$. According to the definition of Pareto dominance, $\vec{u}\preceq\vec{w}$. So Pareto dominance is transitive.

In summary, Pareto dominance is anti-reflexive and transitive. □

A.2. Proof of Property 2

Because $Better(\vec{u},\vec{v})=1,\ \vec{u}\preceq\vec{v}$. So $Better(\vec{v},\vec{u})=-1$.
Because $Better(\vec{u},\vec{v})=-1,\ \vec{v}\preceq\vec{u}$. So $Better(\vec{v},\vec{u})=1$.
Because $Better(\vec{u},\vec{v})=0$, there are no $\vec{u}\preceq\vec{v}$ or $\vec{v}\preceq\vec{u}$. So $Better(\vec{v},\vec{u})=0$. □

A.3. Proof of Property 3

$Better(\vec{u},\vec{v})=1,\ \vec{u}\preceq\vec{v}.Better(\vec{v},\vec{w})=1,\ \vec{v}\preceq\vec{w}$. According to Property 1, $\vec{u}\preceq\vec{w}$, so $Better(\vec{u},\vec{w})=1$. □

A.4. Proof of Property 4

If $Better(\vec{u},\vec{w})=-1$, then $Better(\vec{w},\vec{u})=1$ according to Property 2. Because $Better(\vec{u},\vec{v})=1,\ Better(\vec{w},\vec{v})=1$. But this is impossible because $Better(\vec{v},\vec{w})=0$. So $Better(\vec{u},\vec{w})\neq-1$. □

A.5. Proof of Property 5

The base operation is the comparison of two scalar numbers. The objective space is M dimensions. There are N nodes; i nodes are in the left sub-tree and $N-i-1$ nodes are in the right sub-tree. The possibility of a new node dominating the root is α; the possibility of the node dominated by the root node is β; the possibility of non-dominance is γ and $\alpha+\beta+\gamma=1$. When there are N nodes and i nodes are in the left sub-tree of the root, $Q(N,i)$ is the average number of comparisons, and $\lambda_{N,i}$ is the size of the current Pareto optimal set, namely the size of the sibling chain of the dominance tree. $P(i)$ is the average number of comparisons when the size of the tree is i. $Q(N,i)$ can be calculated as follows. (When the new node dominates the root, we omit the operation that the nodes in the sibling chain dominated by the new node are deleted from the sibling chain and inserted into the left sub-tree of the new node, since these operations are constant.)

$$Q(N,i)=\alpha M\lambda_{N,i}+\beta(P(i)+M)+\gamma(P(N-i-1)+M) \tag{4}$$

In the analysis, we do not consider the boundary limit of the objective space and the effect of the operators, and another hypothesis is that the N nodes are randomly sorted, namely the possibility of i being $0,1,\ldots$ or $N-1$ is equal.

The M-dimension objective space Ω can be denoted as $\Omega=\{(x_1,\ldots,x_M)|-a<x_i<a,i=1,\ldots,M\}$. A solution $\vec{u}_0$ in Ω is denoted as $\vec{u}_0=(u_1,\ldots,u_M)$ and $\vec{u}_0\in\Omega$. $\vec{u}_0$'s Pareto dominating space Ω_{di} is denoted as $\Omega_{di}=\{(v_1,\ldots,v_M)|(v_1,\ldots,v_M)\in\Omega\wedge v_i\leqslant u_i,i=1,\ldots,M\}$. Without regard to the effect of the operators, a new node distributes in Ω with the same possibility, so the possibility of the node being in Ω_{di} can be defined as follows. ($\xi(\Omega)$ denotes the volume of Ω.)

$$\rho(\Omega_{di})=\frac{\xi(\Omega_{di})}{\xi(\Omega)}=\frac{\prod_{i=1,\ldots,M}(u_i+a)}{(2a)^M} \tag{5}$$

Without considering the boundary limit, $a \to \infty$, so $\rho(\Omega_{di}) \to 1/2^M$. The similar analysis shows that when $a \to \infty$, $\rho(\Omega_{dd}) \to 1/2^M$ and $\rho(\Omega_{dn}) \to 1 - 1/2^{M-1}$. So, according to the hypothesis, $\alpha = \beta = 1/2^M$, and $\gamma = 1 - 1/2^{M-1}$.

According to the hypothesis, $P(N)$ can be calculated as follows:

$$P(N) = \frac{1}{N}\sum_{i=0}^{N-1} Q(N,i) = \frac{1-\alpha}{N}\sum_{i=0}^{N-1} P(i) + \frac{M\alpha}{N}\sum_{i=0}^{N-1}\lambda_{N,i} + M(1-\alpha) \tag{6}$$

According to Lemmas 1, $\lambda_{N,i} = \lambda_{Nj}(i \neq j)$. We denote it as λ_N.

$$P(N) = M(1-\alpha+\alpha\lambda_N) + \frac{1-\alpha}{N}\sum_{i=0}^{N-1} P(i) \tag{7}$$

and $P(1) = M$. According to Eq. (7),

$$\sum_{i=0}^{N-1} P(i) = \frac{N}{1-\alpha}(P(N) - M(1-\alpha+\alpha\lambda_N)) \tag{8}$$

and

$$\sum_{i=0}^{N-1} P(i) = P(N-1) + \sum_{i=0}^{N-2} P(i) = P(N-1) + \frac{N-1}{1-\alpha}(P(N-1) - M(1-\alpha+\alpha\lambda_{N-1})) \tag{9}$$

According to Eqs. (8) and (9),

$$P(N) = \frac{N-\alpha}{N}P(N-1) + M\alpha\lambda_N + \frac{1-N}{N}M\alpha\lambda_{N-1} + \frac{M(1-\alpha)}{N} \tag{10}$$

The size of the Pareto optimal set of N nodes is no smaller than that of $N-1$ nodes existing in the N nodes, so $\lambda_{N-1} \leqslant \lambda_N \leqslant \lambda_{N-1} + 1$.

$$\begin{aligned} P(N) &\leqslant (1-\frac{\alpha}{N})P(N-1) + \frac{M\alpha\lambda_N - \alpha M + M}{N} + \alpha M \leqslant P(N-1) + \frac{M(\alpha\lambda_N - 2\alpha + 1)}{N} + \alpha M \\ &\leqslant M(\alpha\lambda_N - 2\alpha + 1)(\frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{N}) + \alpha MN + M \leqslant (\alpha\lambda_N - 2\alpha + 1)M\ln N + \alpha MN + M \end{aligned} \tag{11}$$

λ_N is the size of Pareto optimal set of the N nodes. It has a positive correlation to the number of the nodes N and the number of the objectives M. We estimate the average size of λ_N. In the ideal condition, there are $2\alpha N$ nodes in the root's left sub-tree and $N - 2\alpha N$ nodes in its right sub-tree and each dominance tree has the same rule. So $N(1-2\alpha)^{\lambda_N} \geqslant 1$, namely $\lambda_N \leqslant -\ln N/\ln(1-2\alpha)$. With Taylor Series, $\ln(1-2\alpha) \approx -2\alpha$, so $\lambda_N \leqslant \ln N/2\alpha$. Because $\alpha = 1/2^M$, $\lambda_N \leqslant 2^{M-1}\ln N$. This inequation shows that the size of the Pareto optimal set of the population is the exponential increment with the number of objectives and logarithmic increment with the size of the population. Because $\lambda_N \leqslant 2^{M-1}\ln N$ and $\alpha = 1/2^M$:

$$P(N) \leqslant (\ln N/2 - 1/2^{M-1} + 1)M\ln N + MN/2^M + M \tag{12}$$

Omitting the tiny variable, the computational complexity of a new individual inserted into the dominance tree is $O(M\ln^2 N + MN/2^M)$. □

A.6. Proof of Property 6

We denote d as the number of searching the leftmost node, and this is also the length of the leftmost link of the dominance tree. According to the analysis in Property 5, the following inequation can be drawn: $N(2\alpha)^d \geqslant 1$, namely $d \leqslant -\log_{2\alpha} N$. Because $\alpha = \frac{1}{2^M}$, $d \leqslant \frac{1}{M-1}\log_2 N$. So the computational complexity of deleting the leftmost node is $O(\ln N/M)$. □

References

[1] I. Alberto, P.M. Mateo, Representation and management of MOEA populations based on graphs, European Journal of Operational Research 159 (1) (2004) 52–65.
[2] K. Atashkari, N. Nariman-Zadeh, A. Pilechi, A. Jamali, X. Yao, Thermodynamic Pareto optimization of turbojet engines using multiobjective genetic algorithm, International Journal of Thermal Sciences 2338 (2005) 1–11.
[3] J.M. Chambers, W.S. Cleveland, B. Kleiner, P.A. Turkey, Graphical Methods for Data Analysis, Wadsworth & Brooks/Cole, Pacific Grove, CA, 1983.
[4] X.M. Chen, Pareto Tree Searching Genetic Algorithm: Approaching Pareto Optimal Front by Searching Pareto Optimal Tree, Technical Report NK-CS-2001-002, 2001.
[5] C.A.C. Coello, Evolutionary multiobjective optimization: a historical view of the field, IEEE Computational Intelligence Magazine 1 (1) (2006) 28–36.
[6] C.A.C. Coello, Twenty years of evolutionary multi-objective optimization: what has been done and what remains to be done, Computational Intelligence: Principles and Practice 4 (2006) 73–88.
[7] K. Deb, R.B. Agrawal, Simulated binary crossover for continuous search space, Complex Systems 9 (1995) 115–148.
[8] K. Deb, Multiobjective Optimization using Evolutionary Algorithms, Wiley, UK, 2001.
[9] K. Deb, A. Pratab, S. Agarwal, T. MeyArivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Transaction on Evolutionary Computation 6 (2002) 182–197.

[10] K. Deb, L. Thiele, M. Laumanns, E. Zitzler, Scalable multi-objective optimization test problems, in: Proceedings of 2002 Congress on Evolutionary Computation, 2002, pp. 825–830.
[11] J.E. Fieldsend, R.M. Everson, S. Singh, Using unconstrained elite archives for multiobjective optimization, IEEE Transaction on Evolutionary Computation 7 (2003) 305–323.
[12] C.M. Fonseca, P.J. Fleming, An overview of evolutionary algorithms in multiobjective optimization, Evolutionary Computation 3 (1995) 1–16.
[13] C.M. Fonseca, P.J. Fleming, Multiobjective optimization and multiple constraint handling with evolutionary algorithms – part I: a unified formulation, IEEE Transaction on Systems Man, and Cybernetics – Part A: Systems and Humans 28 (1) (1998) 26–37.
[14] M.T. Jensen, Reducing the run-time complexity of multiobjective EAs: the NSGA-II and other algorithms, IEEE Transaction on Evolutionary Computation 7 (5) (2003) 503–515.
[15] R.T. Moghaddama, A. Rahimi-Vaheda, A. Hossein Mirzaeib, A hybrid multi-objective immune algorithm for a flow shop scheduling problem with bi-objectives: weighted mean completion time and weighted mean tardiness, Information Sciences 177 (2007) 5072–5090.
[16] S. Mostaghim, J. Teich, A. Tyagi, Comparison of data structures for storing Pareto-sets in MOEAs, in: Proceedings of World Congress on Computational Intelligence, 2002, pp. 843–849.
[17] K. Rajeev, P. Rockett, Improved sampling of the Pareto-front in multiobjective genetic optimizations by steady-state evolution: a Pareto converging genetic algorithm, Evolutionary Computation 10 (3) (2002) 283–314.
[18] R. Sedgewick, Algorithms in C++, Parts 1-4-Fundamentals, Data Structures, Sorting, and Searching, third ed., Person Education, 2002. pp. 515–521.
[19] C. Shi, Y. Li, L.S. Kang, A new simple and highly efficient multiobjective optimal evolutionary algorithm, in: Proceedings of 2003 IEEE Conference on Evolutionary Computation, Australia, 2003, pp. 1536–1542.
[20] C. Shi, M. Chen, Z.Z. Shi, A fast nondominated sorting algorithm, in: Proceedings of 2005 IEEE Conference on Neural Network and Brain, China, 2005, pp. 1605–1611.
[21] C. Shi, Q.Y. Li, Z.Y. Zhang, Z.Z. Shi, An improved multiobjective evolutionary algorithm based on dominating tree, in: PRICAI 2006: Trends in Artificial Intelligence, China, 2006, pp. 691–700.
[22] C. Shi, Q.Y. Li, Z.Z. Shi, A quick multi-objective evolutionary algorithm based on dominating tree, Journal of Software (Chinese) 18 (3) (2007) 505–516.
[23] C. Shi, Z.Z. Shi, B. Wu, An efficient fitness assignment based on dominating tree, in: Workshops on ICDM, 2007, pp. 247–252.
[24] C. Shi, Z.Y. Yan, Z.Z. Shi, L. Zhang, A fast multiobjective evolutionary algorithm based on a tree structure, Applied Soft Computing, accepted for publication.
[25] N. Srinivas, K. Deb, Multiobjective optimization using nondominated sorting in genetic algorithms, Evolutionary Computation 2 (3) (1995) 221–248.
[26] K.C. Tan, T. Lee, E. Khor, Evolutionary algorithms with dynamic population size and local exploration for multiobjective optimization, IEEE Transaction on Evolutionary Computation 5 (2001) 565–588.
[27] P.K. Tripathi, S. Bandyopadhyaya, S.K. Pala, Multi-objective particle swarm optimization with time variant inertia and acceleration coefficients, Information Sciences 177 (2007) 5033–5049.
[28] D.A. Van Veldhuizen, G.B. Lamont, Multiobjective evolutionary algorithms: analyzing the state-of-the-art, Evolutionary Computation 18 (2) (2000) 125–147.
[29] G. Yen, H. Lu, Dynamic multiobjective evolutionary algorithm: adaptive cell-based rank and density estimation, IEEE Transaction on Evolutionary Computation 7 (3) (2003) 253–274.
[30] S.Y. Zeng, L.S. Kang, L.X. Ding, An orthogonal multi-objective evolutionary algorithm for multi-objective problem with constraints, Evolutionary Computation 12 (1) (2004) 77–98.
[31] E. Zitzler, L. Thiele, Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach, IEEE Transaction on Evolutionary Computation 3 (4) (1999) 257–271.
[32] E. Zitzler, K. Deb, L. Thiele, Comparison of multiobjective evolutionary algorithms: empirical results, Evolutionary Computation 18 (2) (2000) 173–195.
[33] E. Zitzler, E. Laumanns, L. Thiele, SPEA2: Improving the Strength Pareto Evolutionary Algorithm, TIK-Report 103, ETH Zentrum, Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001.
[34] E. Zitzler, L. Thiele, M. Laumanns, C.M. Fonseca, V.G.D. Fonseca, Performance assessment of multiobjective optimizers: an analysis and review, IEEE Transaction on Evolutionary Computation 7 (2) (2003) 117–132.