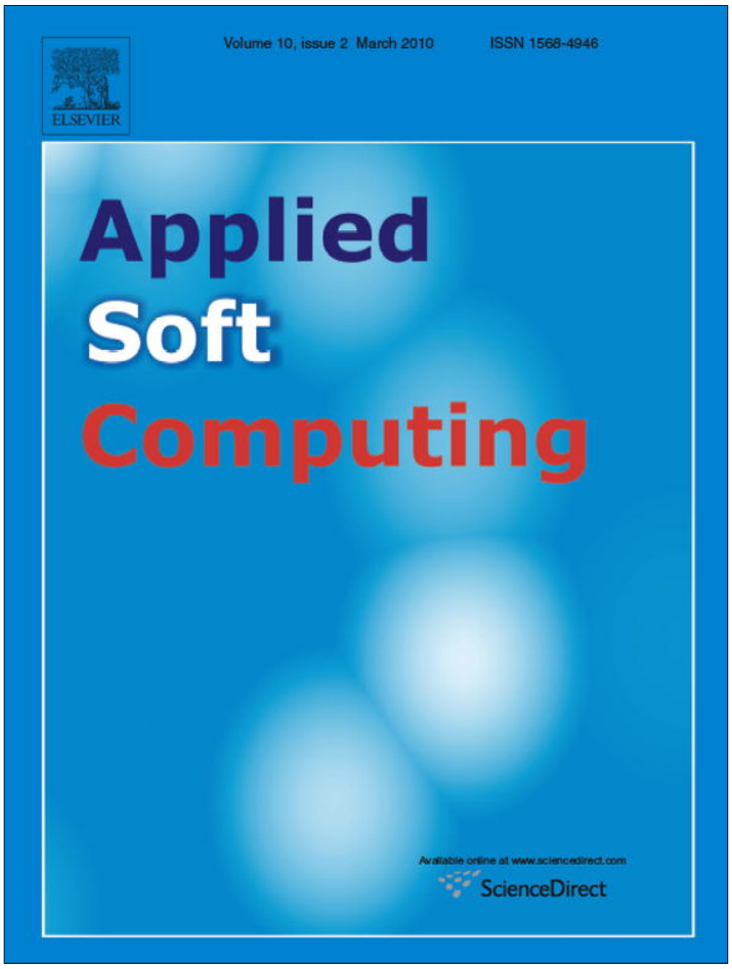

Volume 10, issue 2 March 2010
ISSN 1568-4946
ELSEVIER
Applied Soft Computing
Available online at www.sciencedirect.com
ScienceDirect

Applied Soft Computing 10 (2010) 468–480

Contents lists available at ScienceDirect

Applied Soft Computing

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

A fast multi-objective evolutionary algorithm based on a tree structure

Chuan Shi [a,*], Zhenyu Yan [b], Zhongzhi Shi [c], Lei Zhang [a]

[a] Beijing Key Laboratory of Intelligent Telecommunications Software and Multimedia, Beijing University of Posts and Telecommunications, Beijing 100876, China
[b] Department of Systems and Information Engineering, University of Virginia, 22903, USA
[c] Institute of Computing Technology, Chinese Academy of Sciences, 100080, China

ARTICLE INFO

Article history:
Received 15 January 2008
Received in revised form 30 June 2009
Accepted 13 August 2009
Available online 21 August 2009

Keywords:
Multi-objective evolutionary algorithm
Pareto dominance
Fitness assignment
Eliminating strategy

ABSTRACT

This paper proposes a fast evolutionary algorithm based on a tree structure for multi-objective optimization. The tree structure, named *dominating tree* (*DT*), is able to preserve the necessary Pareto dominance relations among individuals effectively, contains the density information implicitly, and reduces the number of comparisons among individuals significantly. The evolutionary algorithm based on *dominating tree* (DTEA) integrates the convergence strategy and diversity strategy into the *DT* and employs a *DT*-based eliminating strategy that realizes elitism and preserves population diversity without extra time and space costs. Numerical experiments show that DTEA is much faster than SPEA2, NSGA-II and an improved version of NSGA-II, while its solution quality is competitive with those of SPEA2 and NSGA-II.

1. Introduction

Solving multi-objective scientific and engineering problems is, generally, a very difficult goal. In these multi-objective optimization problems (MOPs), the objectives often conflict across a high-dimensional problem space and the optimization of MOPs may also require extensive computational resources. Classical optimization methods suggest converting the MOP to a single-objective optimization problem (SOP), in which many runs are required to find the multiple solutions. This makes an algorithm that returns a set of candidate solutions preferable to an algorithm that returns only one solution based on some weights of the objectives. For this reason, there has been an increasing interest in applying evolutionary algorithms (EAs) to MOPs in the past 20 years [1].

Many multi-objective evolutionary algorithms (MOEAs) have been proposed. These MOEAs use Pareto dominance to guide the search, and return a set of nondominated solutions as a result. Unlike in single-objective optimization, where finding the optimal solution as the ultimate goal, there are two goals in multi-objective optimization: (1) convergence to the Pareto optimal set and (2) maintenance of diversity in the Pareto optimal solutions set [2]. Many strategies and methods have been introduced to address the two sometime conflicting goals in MOPs [3]. A common problem with those methods is that they are often intricate. To achieve better solutions in terms of the two goals, the complicated strategies are usually used and a number of parameters need to be adjusted according to the experiences and prior knowledge of given problems. Besides, many MOEAs have the high computational complexity with $O(GMN^2)$ or more processing time (G is the number of generations, M is the number of objectives and N is the population size. These symbols maintain the same meanings throughout the following sections) [4].

In this paper, we propose a fast multi-objective evolutionary algorithm based on a tree structure. This data structure is a binary tree that preserves the three-valued dominance relations (i.e., dominating, dominated and nondominated) among the solutions of a MOP and thus we name it *dominating tree* (DT). With some unique properties, *DT* is able to contain the density information of individuals implicitly, and reduce the comparisons among individuals distinctly. The computational complexity experiments also confirm that *DT* is an efficient tool to manage the population. The evolutionary algorithm based on *dominating tree* (DTEA) integrates the convergence and diversity strategies, namely the two goals in a MOEA, into the *DT*, and as a consequence, the algorithm is easy to populate with only a few parameters. In addition, DTEA employs an eliminating strategy specially designed based on *DT*. This strategy not only maintains the diversity of population naturally, but also realizes elitism without extra costs. Six benchmark test functions and three well-known MOEAs (i.e., NSGA-II [2], SPEA2 [5], and the improved version of NSGA-II by Jensen [4]) are used to examine the

* Corresponding author.
E-mail addresses: shichuan@bupt.edu.cn (C. Shi), yan_zhen_yu@hotmail.com (Z. Yan), shizz@ics.ict.ac.cn (Z. Shi), zlei@bupt.edu.cn (L. Zhang).

1568-4946/$ – see front matter. Crown Copyright © 2009 Published by Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2009.08.018

efficiency and effectiveness of DTEA. The running time experiments show that DTEA is always much faster than the benchmark algorithms, especially when the population size is large. On the other hand, by examining the three criteria for evaluating the quality of solutions, we find that the solution quality of DTEA in general is not worse than NSGA-II and SPEA2 in terms of both converging to the true Pareto optimal front and maintaining the diversity of the population, although the computing time required by DTEA is much shorter.

The remainder of this paper is arranged as follows. Section 2 briefly reviews the state-of-the-art of MOEAs. Section 3 describes DTEA in details. Section 4 presents and discusses the experimental results. Finally, Section 5 concludes this paper.

2. An overview of MOEAs

This section gives a brief survey on contemporary MOEA research. The first part will give some notions to be used in the paper. The second part will briefly analyze the current state-of-the-art of MOEAs.

2.1. Definitions in MOP

Many researchers have given similar definitions [3,6,7] for MOP. We introduce the definitions of two important concepts here. Without loss of generality, we only consider minimization problems in this paper, and it is easy to convert a maximization problem into a minimization problem.

Definition 1. General MOP [3]: In general, an MOP minimizes $\boldsymbol{F}(\vec{x}) = (f_1(\vec{x}), \ldots, f_m(\vec{x}))$ subject to $g_i(\vec{x}) \le 0$, $i = 1, \ldots k$, $\vec{x} \in \Omega$ (Ω is the decision variable space). An MOP solution minimizes the components of an m-dimensional objective vector $\boldsymbol{F}(\vec{x})$, where $\vec{x} = (x_1, \ldots, x_n)$ is an n-dimensional decision variable vector from some universe Ω.

Definition 2. Pareto dominance [3]: A vector $\vec{u} = (u_1, \ldots, u_m)$ is said to dominate $\vec{v} = (v_1, \cdots, v_m)$ (denoted by $\vec{u} \preceq \vec{v}$), if and only if $\vec{u}$ is partially less than $\vec{v}$, i.e.,

$$\forall\, i \in \{1, \ldots, m\} \quad u_i \le v_i \wedge \exists \quad i \in \{1, \ldots, m\} \;\; u_i < v_i \tag{1}$$

Most contemporary research on MOP is based on Pareto dominance. (In some literatures, Pareto dominance is also defined with strict $\prec$ [7]. Here we do not discuss the difference.) A decision vector $\vec{x} \in \Omega$ is said to be Pareto optimal if and only if there is no $\vec{y} \in \Omega$ for which $\boldsymbol{F}(\vec{x}) \preceq \boldsymbol{F}(\vec{y})$. The set of all Pareto optimal decision vectors is called the Pareto optimal set. The corresponding set of the objective vectors is called the nondominated set, or Pareto front [6].

According to the definition of Pareto dominance, we can discover a distinct difference between MOPs and SOPs in terms of the relations among solutions. The solutions in the objective space of SOPs are scalar numbers and their relations have two possibilities: smaller than and larger than (including "equal to", for simplicity, it is often simply called larger than). However, the solutions of MOPs are vectors and their relations have three possibilities: $\vec{u} \preceq \vec{v}$, $\vec{v} \preceq \vec{u}$ and nondominated. This difference requires that MOEAs have more complicate fitness assignment rules.

2.2. State-of-the-art of MOEAs

A good MOEA must satisfy the following two aspects: (1) the resulting nondominated set converges to the true Pareto optimal front; (2) a uniformly distribution of the solutions is desirable [2,3]. These two goals are often the performance metrics for most MOEAs [2,5,6]. Many methods are designed to reach these two goals [3]. To address the first aspect, a Pareto-based fitness assignment method is usually designed to guide the search toward the true Pareto front [8]. The basic idea is to rank the solutions according to their dominance relations. Goldberg first proposed a popular method that the solution set is divided into different fronts with different ranks [9]. Fonseca and Fleming proposed a method where the rank of a solution is assigned the number of solutions associated with the current population dominating it [10]. One more ranking method was proposed in SPEA2 [5] where each individual is assigned a strength value. For the second aspect, some successful MOEAs provide the density estimation methods to preserve the population diversity [8]. Pareto niching and fitness sharing were popularly used in many MOEAs, for example, NSGA [11], NPGA [12] and MOGA [10]. In SPEA2 [5], a kth nearest neighbor density estimation method was applied to obtain the density index of each individual. NSGA-II defined a novel density-estimation metric that does not require any user-defined parameter [2]. Another popular tactic divided the objective space into cells using a hyper-grid [6,13]. Moreover, a new ε-eliminating diversity approach was also proposed [14,15].

Recently, some new evolutionary paradigms have been successfully applied to MOPs, for example, particle swarm optimization [16], artificial immune systems [17], estimation of distribution [18], and scatter search [19]. Zhang and Li also proposed a new MOEA based on decomposition. This method decomposes a MOP into a number of scalar optimization subproblems and then optimizes them simultaneously [20]. In addition, several effective techniques have also been adopted to obtain good solutions. The current research has shown that the elitism can improve the performance of the MOEAs significantly, and it helps to prevent the loss of good solutions once they have been found [2,5,21]. A dynamic population size, adjusted autonomously by the online characteristics of population tradeoff and density distribution information, has been found to be more efficient and effective than a constant population size in terms of avoiding premature convergence and unnecessary computational complexity [6,22]. Moreover, using unconstrained elite archives could avoid the retreating and shrinking estimated Pareto fronts [23].

Although a number of MOEAs have achieved good performances in some benchmark problems with these strategies and technologies, there are still several disadvantages with these algorithms. On the one hand, many MOEAs are intricate. To obtain good solutions, many MOEAs use separate techniques for convergence and diversity, although these two aspects are eventually integrated into the fitness evaluation of the individuals in most MOEAs [2,5,6]. In addition, many parameters in the MOEAs need to be adjusted according to the problem domain knowledge and experiences. For example, six strategies are used and four parameters need to be adjusted in DMOEA [6].

On the other hand, many MOEAs are time-consuming. The high computation demand of most published MOEAs is partially due to the fact that MOP is usually a hard computation problem (e.g., because of the three-valued relations) as compared to SOP. Another explanation lies in the fact that the MOEA research has often neglected the issue of computational complexity [4]. Fortunately, many researchers have begun to pay attention to this problem. Deb et al. proposed a fast-nondominated sort algorithm to reduce the computational complexity of the nondominated sort process from $O(MN^3)$ to $O(MN^2)$ in NSGA-II [2]. Jensen has systemically analyzed the computational complexity of many contemporary MOEAs, and presented some efficient algorithms for the nondominated sorting process [4]. Moreover, some data structures have been introduced to alleviate the difficulty. Quad-tree has been examined as an elite archive to store nondominated solutions [24]. The irreducible domination graph (IDG) was proposed to manage population [25]. Dominated and nondominated tree structures were introduced to facilitate the nondominated sorting in the elite archive [23,26]. These new algorithms and data structures speed up the fitness assignment process to some extent. However, it is still desirable to further investigate this important issue and design simple but efficient fitness assignment approaches.

3. Evolutionary algorithm based on dominating tree

3.1. Dominating tree

3.1.1. Structure of dominating tree

As we know, the fitness assignment is a key component in MOEAs, which highly impacts the algorithm's performance; and it is also a costly matter in terms of the processing time. Most Pareto-based fitness assignments require that each solution should be compared with a large number of other solutions, which imposes most MOEAs with the computational complexities bounded by $O(GMN^2)$ [4]. Please note that $O(N^2)$ factor means the complexity grows very fast as the population size increases. However, in many situations, it is desirable to use large population sizes for MOEAs, especially when the number of conflicting objectives is large. Thus, this dilemma needs to be addressed.

Through analyzing some of the popular fitness assignment processes, we can find there are many unnecessary comparisons in the process. The dominance relations among individuals can be visualized with a graph where each node represents an individual and each edge represents the relation between the adjacent nodes. For example, Fig. 1 illustrates the dominance relations among five nodes with a graph. We intuitively observe that there might be some redundant relations in Fig. 1(b). Reducing these redundant relations may be an efficient strategy to reduce the computational complexity of the fitness assignment.

Because the relation of Pareto dominance is transitive [7], some relations can be deduced based on the existing relations. Taking Fig. 1(b) for example, if we know N4 Pareto dominates N3, and N3 Pareto dominates N2, then we can deduce N4 Pareto dominates N2 without any direct comparison. Thus, the comparison between N4 and N2 can be avoided. We also notice another fact that we only need to achieve the Pareto optimal set of the population in each generation, although those dominated solutions are still useful for the evolutionary process. The reason is that a decision maker often makes decisions based on the optimal set without paying much attention to the other solutions. Also due to this reason, many relations among the solutions (e.g., the relations among the dominated solutions) are unnecessary to know, and thus these relations can be reduced. In the example in Fig. 1(b), it is obvious that N1 and N4 constitute the Pareto optimal set. We can observe that N4 dominates N3 and N1 dominates N2. Thus, the relation between N2 and N3 may not be of interests to us and it may not be worth spending the computing time either directly comparing or deducing it. Through the analysis above, we could find two principles to reduce the unnecessary computing time: (1) avoiding the redundant comparisons (i.e., inferring those relations by deduction); (2) only preserving the necessary relations (i.e., ignoring those relations that are not very of interests).

As we have mentioned, different from the relations of the solutions in SOPs, those in MOPs are three-valued. It can be summarized by a *Better* function:

Definition 3. *Better* function

$$Better(\vec{x}_1,\vec{x}_2)=\begin{cases}1 & \boldsymbol{F}(\vec{x}_1)\preceq\boldsymbol{F}(\vec{x}_2)\\ -1 & \boldsymbol{F}(\vec{x}_2)\preceq\boldsymbol{F}(\vec{x}_1)\\ 0 & \text{nondominated}\end{cases} \quad (2)$$

The *Better* function is a three-valued function. When the objective vector of $\vec{x}_1$ dominates that of $\vec{x}_2$, $Better(\vec{x}_1,\vec{x}_2)=1$; when the objective vector of $\vec{x}_1$ is dominated by that of $\vec{x}_2$, $Better(\vec{x}_1,\vec{x}_2)=-1$; when they are nondominated, $Better(\vec{x}_1,\vec{x}_2)=0$. As we know, the binary sort tree (BST) is an effective tool for storing two-valued relations among scalar numbers [27]. In a BST, a node's left sub-tree links to a node whose value is smaller than itself and its right sub-tree links to a node whose value is larger than itself. We consider the three-valued relations in MOPs may also be stored in a binary tree with extensions.

Following the idea of tree structure proposed by the authors' early work in Ref. [28], we design a special binary tree for this

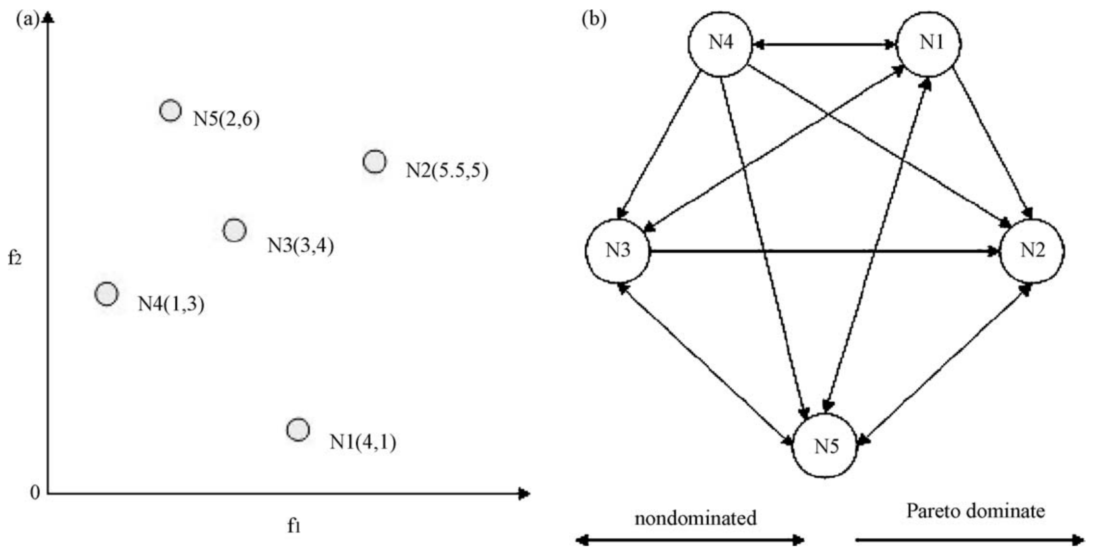


Fig. 1. (a) Illustration of five nodes in the two-dimensional objective space. (b) Illustration of the relations among five nodes using graph.

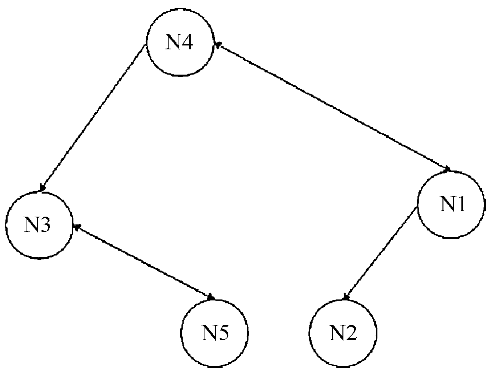


Fig. 2. Illustration of the relations among nodes in Fig. 1 with dominating tree.

purpose. An example that represents the relations among the nodes in Fig. 1 is shown in Fig. 2. In the figure, where the Pareto optimal set is {N1, N4}, the major relations are conserved and many unimportant relations (e.g., the relations among the dominated nodes) are omitted. As a novel binary tree, we name it *dominating tree.*

Definition 4. *Dominating tree* (*DT*). A *dominating tree* is a binary tree defined below.

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

The definition of a *DT* is similar to that of a BST. *DT* and BST also share some similar properties. For example, in both *DT* and BST, a node's child is the root of its sub-tree; correspondingly, the node is the parent of its child. *DT* also has some different features. A new term "sibling chain" is used in *DT*. The sibling chain of a *DT* is defined as a chain constituted by its root and the root's right-link nodes. Since each *DT* has only one root node, the sibling chain of a *DT* sometimes is simply called the sibling chain of the *DT*'s root node in the following sections. By tracing down the right-link filed, one can obtain the sibling chain. Taking Fig. 2 as an example, N1 and N4 constitute the sibling chain of the *DT* whose root node is N4. Moreover, N3 and N5 also constitute a sibling chain of the *DT* whose root node is N3. It should be noted that although with the same names, the left and right sub-trees of a node in a *DT* and a traditional BST have different meanings and thus their properties are also different, which will be further discussed in Section 3.1.3.

3.1.2. Construction algorithms of dominating tree

If we consider that the dominated relation is comparable to the smaller than relation in a BST and the nondominated relation is comparable to the larger than relation in a BST, and then a *DT* is very similar to a BST. Thus, the construction of a *DT* is similar to that of a BST. However, since we know that the relation among the nodes in a BST is two-valued, whereas that of a *DT* is three-valued, besides the similarities, they do have a number of different aspects. Unlike in BST, a new node to be inserted into a *DT* will face three choices. When the new node is dominated by the root, it will be inserted into the left sub-tree of the root. This is similar to the smaller than relation in a BST. When the new node is nondominated by the root, it will be inserted into the right sub-tree of the root. This is similar to the larger than relation in a BST. When the new node dominates the root, the new node should not only replace the root and let the root insert into its left sub-tree, but also continue to compare with the other nodes in its sibling chain. If there are nodes dominated by the new node, those dominated nodes should be deleted from the sibling chain and then be inserted into the left sub-tree of the new node.

Fig. 3 demonstrates the process of creating a *DT* (the complete tree is shown in Fig. 2). The input order of these nodes is from N1 to N5, and the inserting steps are shown in Fig. 3(a)–(e). After the five nodes have been inserted, the *DT* with five nodes is shown in Fig. 3(e). It is obvious that the *DT* is unbalanced because the count of N4 (which is 3) is larger than that of N1 (which is 2). To balance the tree, N4 with its left sub-tree moves along the sibling chain in the left direction as shown in Fig. 3(f) (i.e., N4 with its left sub-tree switch the position with N1 with its left sub-tree in this case). Although the action does not change the relations of the nodes (note that the nodes in the sibling chain are still nondominated to each other), it balances the *DT*, which is now the same as the complete tree shown in Fig. 2. An additional benefit of the balancing process is that the nodes in the same sibling chain will be sorted by their count descending order after the process. With this property, deleting a node becomes very easy. This will be further illustrated with the node deleting algorithm (i.e., Algorithm 2).

Based on the above examples, we show the *dominating tree* construction algorithms in Algorithm 1. *ConstructTree* is the main loop of creating a *DT*. There are three functions in the algorithms. *AddinTree* and *AddinSibling* consist of the necessary actions when a new node is inserted into a *DT*. *BalanceTree* performs the balancing process described above. More specifically, it sorts the nodes in the sibling chain in their count descending order by moving the nodes to the left or right direction along the same sibling chain.

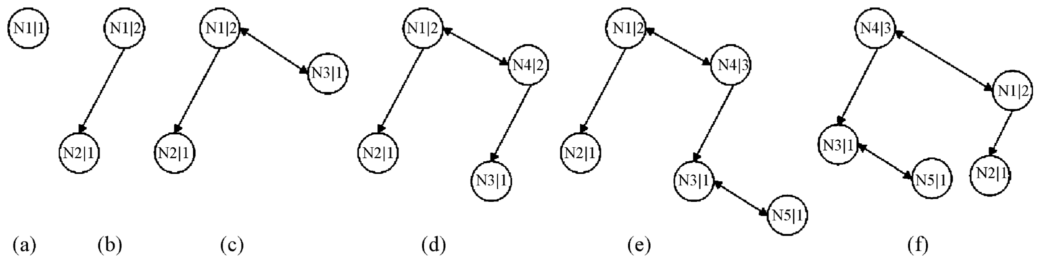


Fig. 3. Illustration of the creating process of the dominating tree in Fig. 2. The left number in the node is its id, and the right number is its count. The sequence is alphabetic.

Algorithm 1. Pseudocode of *dominating tree* construction algorithms

```
/*Creating a dominating tree. pTree is the root of the whole tree
(for simplicity, pTree only denotes the root, not a common
individual).
Input: all individuals in the population
Output: the dominating tree */
Link ConstructTree (Pop P){
  for each individual (pNewnode) in the population P
    AddinTree(pTree, pNewnode);
  return pTree;
}

/* Add pNewnode into the left sub-tree of the tree whose root
node is pRoot, when pNewnode is dominated by pRoot. */
AddinTree (Link pRoot, Link pNewnode){
  pRoot->count = pRoot->count + pNewnode->count;
  if ( pRoot->left-link == NULL )
    pRoot->left-link = pNewnode;
  else AddinSibling (pRoot, pRoot->left-link, pNewnode);
}

/* pNewnode is compared with pChild, a node in the sibling
chain of pParent's left sub-tree when pNewnode is inserted into
pParent's left sub-tree.*/
AddinSibling (Link pParent, Link pChild, Link pNewnode) {
  switch (Better (pNewnode, pChild))
  case 0:     //nondominated
    if (pChild->right-link == NULL)
      pChild->right-link = pNewnode;
    else AddinSibling (pParent, pChild->right-link,
         pNewnode);
  case 1:     //dominating
    pNewnode takes the place of pChild;
    AddinTree (pNewnode, pChild);
    for each node (pNode) in pNewnode's sibling chain {
      if (Better(pNewnode, pNode) ==1){
        delete pNode from the sibling chain;
        AddinTree (pNewnode, pNode);
      }
    }
    BalanceTree(pParent, pNewnode, L);
  case -1 :     //dominated
    AddinTree (pChild, pNewnode);
    BalanceTree (pParent, pChild, L);
}
/* Sort the nodes in the sibling chain of the parent'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 direction, or else to the right direction. */
BalanceTree (Link pParent, Link pMovenode, int direction) {
  if (direction == L)
    while pMovenode->count > tempNode->count
    /*tempNode is the pMovenode's left node which is in the
    same sibling chain */
     do switch the position of pMovenode and tempNode;
  if(direction == R)
     while pMovenode->count < tempNode->count
     /*tempNode is the pMovenode's right node which is in the
     same sibling chain */
     do switch the position of pMovenode and tempNode;
}
```

3.1.3. Properties of dominating tree

From the construction process, one can derive several important properties of the *DT*.

Lemma 1. *The sibling chain of a DT consists of and only consists of all Pareto optimal nodes in the DT.*

Rationale. If there is only one node, it is obvious that the lemma is true. If the lemma is true for *N* nodes, there are three possibilities for a new node (pNode) to be inserted according to the *dominating tree* construction algorithms. Case one: if pNode is dominated by the root, it is inserted into the left sub-tree of the root. Case two: if pNode is nondominated by the root, it should continue to be compared with the other nodes in the sibling chain of the *DT*. For this case, there are also three possibilities for pNode. 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, this dominated node will be deleted from the sibling chain and pNode will continue the comparison with the remaining nodes; if it is nondominated by all the others in the sibling chain, it becomes a member of the sibling chain. Case three: if pNode dominates the root, the root will be deleted from the sibling chain; and for the other nodes in the sibling chain, if they are dominated by pNode, they will also be deleted from the sibling chain. Therefore, the construction process guarantees that the sibling chain consists of and only consists of the Pareto optimal nodes.

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

The proof is similar to that of Lemma 1, and thus it is omitted here.

Lemmas 1 and 2 are the major relations recorded in a *DT*. It should be noted that impacted by the *BalanceTree* operation in Algorithm 1, the construction algorithms do not promise that a node is nondominated by all of the nodes in its right sub-tree. (Note that this is still consistent with the definition of *DT*, because the definition only guarantees that a node is nondominated by the root node of its right sub-tree.) For example in Fig. 3(f), N4 is nondominated by N1, but not nondominated by N2 which is in the right sub-tree of N1.

Lemma 3. *The root node of a dominating tree has the largest count value among the nodes (including itself) in its sibling chain.*

The proof is omitted here, since it is obvious according to *BalanceTree* function in Algorithm 1.

Lemma 3 guarantees that a *DT* is as balanced as possible. As a side benefit, we can get better average-case performance when a node is inserted or deleted (see Section 4.1).

3.2. DT-based eliminating strategy

The fitness assignment process is a crucial component of MOEAs. Many effective fitness assignment approaches have been introduced (see Section 2.1). In fact, the *DT* can be employed as a fitness assignment approach. Different from partitioning multi-fronts in NSGA-II and calculating the strength value in SPEA2, the *DT* naturally records the dominance relations of solutions in the tree.

Lemma 1 guarantees that all of the Pareto optimal nodes in the current population are stored in the sibling chain of the tree, and Lemma 2 guarantees that a node dominates all of the nodes in its left sub-tree. With the two properties, in each generation, the *DT* can naturally archive the Pareto optimal nodes which usually should be least likely to be eliminated. According to Lemma 3, the leftmost node in the *DT* (e.g., N3 in Fig. 2) is always dominated by a

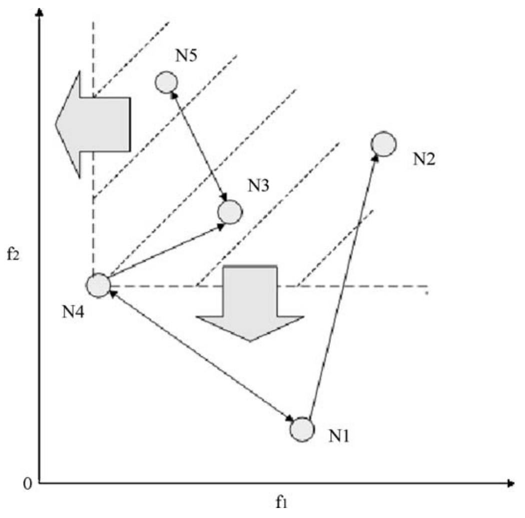


Fig. 4. Illustration of eliminating strategy. The shaded space is the Pareto dominated space of N4.

large number of nodes in the *DT* unless all nodes are nondominated and thus it can be regarded as the "worst" node of the *DT*. According to the essence of EA, survival of the fittest, this node should be more likely to be deleted. Therefore, the eliminating strategy will drop the leftmost node of the *DT* first in each generation. For example, in Fig. 2, the node to be deleted is N3. Algorithm 2 shows the pseudocode of the eliminating strategy. The eliminating strategy guarantees that the deleted node (i.e., the leftmost node) is dominated by a large number of the nodes in the tree and the *DT* is also as balanced as possible after the elimination. Please note that when all nodes are nondominated, the *DT* degrades into a list with the left-link fields of all nodes being NULL, and thus the node to be deleted is the first node of the *DT*'s sibling chain (i.e., the root node).

This eliminating strategy can also naturally maintain the diversity of population. A node with a big count means that there are many nodes dominated by it in the objective space, namely its Pareto dominated space[1] is more "crowded", and thus the nodes in its Pareto dominated space should be more likely to be eliminated. The larger a node's count is, the more "crowded" its Pareto dominated space is. This eliminating strategy (i.e., dropping the leftmost node) tends to eliminate the solutions in the "crowded" Pareto dominated space and consequently the solutions will tend to distribute evenly. As illustrated in Fig. 4, N4's Pareto dominated space is more "crowded" than the other nodes, because its count value is the largest one. Thus, the nodes in the N4's left sub-tree are more likely to be eliminated under this strategy. We also notice that the eliminating strategy may be not functional in the case when all nodes are nondominated. However, the experiments in Section 4.2.2 will show that the strategy still can maintain an acceptable diversity in this situation.

The *DT*-based eliminating strategy is obviously different from the methods used in other MOEAs. In NSGA-II, the crowded-comparison operator is used to guide the selection process [2], in which, between two solutions with different nondomination ranks, the solution with the lower rank is preferred. Otherwise, if both solutions have the same rank, then the solution located in a less crowded region is preferred. In the selection process of SPEA2, the nondomination ranks are considered first; when the archive is too small or too large, the density information is then considered. Compared to these complicated strategies, the *DT*-based eliminating strategy simply eliminates the leftmost node in the *DT*, because the node is not only dominated by many nodes, but also located in the "crowded" region. For the five nodes in Fig. 1, the first node to be eliminated should be N2 in both SPEA2 and NSGA-II, whereas it is N3 in the *DT*-based eliminating strategy.

Algorithm 2. Pseudocode of eliminating strategy.

```
/*Delete the worst node of the dominating tree
Input: the dominating tree
Output: the worst node */
Link DeleteWorstNode (Link pRoot){
  Link p = pRoot->left-link;
  pRoot->count--;
  if (p->left-link == NULL)
    pRoot->left-link = p->right-link;
    return p;
  else
    return DeleteWorst(pRoot);
}

Link DeleteWorst(Link pRoot) {
  Link p = pRoot->left-link;
  Link q = p->left-link;
  p->count--;
  if(q->left-link == NULL)
    p->left-link = q->right-link;
  else
    q = DeleteWorst (p);
  BalanceTree (pRoot, p, R);
  return q;
}
```

3.3. Main loop

Based on the *DT* construction algorithms and the *DT*-based eliminating strategy, we propose a new algorithm—evolutionary algorithm based on *dominating tree* (DTEA).

Algorithm 3 describes the pseudocode of DTEA. DTEA first randomly generates an initial population and then create a *DT* with the individuals in the population by following the construction algorithms. In each generation, a new-generated child will be inserted into the *DT* and the "worst" individual will be deleted from the *DT* by following the elimination strategy. This generating-eliminating process will be repeated until the stopping criterion is satisfied. At the beginning of the evolution, the sibling chain of the *DT* usually is small. In an extreme situation when there are no nondominated nodes in the population, each node in the *DT* only has its left sub-tree. In the middle stages of the evolution, the *DT* becomes more "balanced". At the end of the evolution, more individuals are nondominated, and thus the sibling chain of the *DT* usually becomes larger. In an extreme condition, the *DT* degrades to a sibling chain (i.e., a list) when all nodes are nondominated. However, when a new individual is inserted into the *DT* and the new individual is *better* than at least one of the individuals in the population, the *DT* will become "balanced" again. Those changes of a *DT* during the evolutionary process are illustrated in Fig. 5. As an algorithm framework, DTEA in Algorithm 3 does not explicitly give any specific individual generating methods. In fact, most existing crossover and mutation operators can be used to generate the offspring. In addition, the algorithm can also be extended to generate and delete more than one individual in one generation.

Algorithm 3. Pseudocode of the main loop of DTEA. *P* stands for the population, *T* stands for the *dominating tree*, *c* stands for the

[1] A node's Pareto dominated space is the space in which a node is dominated by it. It is similar to the forbidden region concept in Ref. [8].

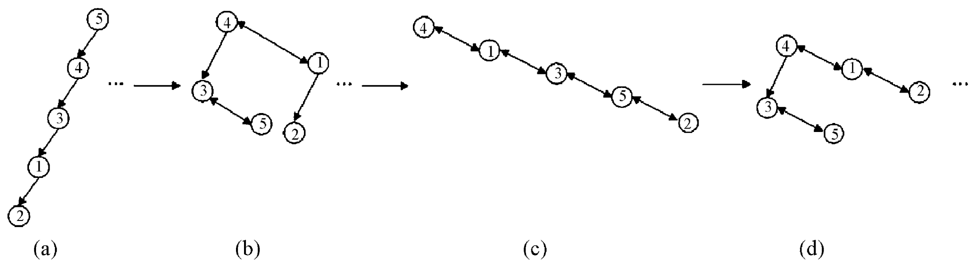


Fig. 5. The changes of dominating tree in the evolutionary process.

new-generated child, w stands for the worst node to be deleted, and t stands for the running generation.

```
DTEA
{
  generate $P_0$ at random;
  $T_0$ = ConstructTree( $P_0$ );
  set $t = 0$ ;
  while (not done){
    generate child $c_t$ from $P_t$ ;
    AddinTree( $T_t$ , $c_t$ );
    delete the worst node $w_t$ = DeleteWorstNode( $T_t$ );
    $P_{t+1} = P_t - \{w_t\} + \{c_t\}$ ;
    set $t = t + 1$ ;
  }
  return $T_t$ ;
}
```

3.4. Discussion of DTEA

As a novel MOEA based on *DT*, DTEA has the following original features.

1. DTEA integrates the convergence and diversity strategy into the *DT*. Many MOEAs use separate strategies for guaranteeing the convergence to the optimal front and maintaining the population diversity, which could make MOEAs intricate and time-consuming. The *DT*-based eliminating strategy not only naturally preserves the dominance relations of solutions, but also contains the density information implicitly. Thus, the DTEA can satisfy the two goals of MOEAs through an integrated strategy based on *DT*.
2. DTEA is a steady-state algorithm (the definition of steady-state algorithms can be seen in [29,30]). Because only one individual (or a very small portion of the population) is generated or eliminated in one generation, the possibility of an individual being eliminated is small. Once a good solution is found, it cannot be eliminated until it becomes the "worst" one. As stochastic algorithms, some early developed non-elitism MOEAs may suffer from losing good solutions during the evolutionary process [3]. The elite archive employed by many modern MOEAs maintains all Pareto optimal individuals (i.e., elites) of the current population, which can overcome the disadvantage [21,31], but it increases the time and space complexities. DTEA realizes elitism naturally with the steady elimination strategy without any extra expenses.
3. DTEA is simpler to implement than many MOEAs. There are no explicit diversity strategy and density estimation parameters as in other MOEAs.

Similar to DTEA, PAES also generates and deletes an individual in one generation [13]. PAES is identified as a (1 + 1) evolution strategy [32], using local search from a population to identify the approximate dominance ranking of the current and candidate solution vectors. There are several differences between the two algorithms. First, there is only one archive to store the elites in PAES. In DTEA, there is more than one archive to store the nondominated solutions of different layers, because each sibling chain in a *DT* can be seen as an elite archive to store the "local" nondominated solutions in different layers. Second, PAES only uses the mutation operator, since it is based on the evolutionary strategy. Whereas, many operators, including crossover and mutation, can be used in DTEA. Last but not least, the hyper-grid method is used in PAES to maintain the diversity; however, there is no special diversity strategy in DTEA.

There are some other interesting data structures used in MOEAs. Mostaghim et al. examined Quad-tree for storing Pareto-points [24]. Fieldsend et al. introduced the dominated/ nondominated tree to facilitate the use of an unconstrained elite archive [23,26]. Although our method has a similar name to Fieldsend's method and they both reduce the unnecessary comparisons, they are completely different in terms of the mechanism. In addition, they are also designed for addressing different problems. The *DT*, as a new data structure, not only can be used for the fitness assignment but also for the diversity maintenance, and it stores the dominance information of all individuals. Whereas, the dominated/nondominated tree and Quad-tree are used to store and sort the individuals only in the elite archive. Although the dominated/ nondominated tree can also be used as the base of MOEA [33], the *DT* has the unique property that it contains the density information implicitly. Another graph structure, IDG, has been proposed by Alberto et al. [25] to maintain the relations among the individuals in the whole population, whereas they have different data structures. Moreover, *DT* requires less time than IDG when inserting or deleting a node (see Section 4.1). The tree structure in Ref. [28] documented some of the authors' preliminary work on *DT* and it can be regarded as a precursor of the *DT* proposed in this paper, although it is very rough and contains many unsolved issues.

Recent work has shown that the restricting number of solutions in the elite front can result in shrinking [5] and oscillating/ retreating estimated Pareto front [26], so some research highlight the use of an active archive of elite to improve the optimization speed of these algorithms [6,23]. In DTEA, the archive of elite remains active until all nodes become nondominated for the first time; after that, the archive is restricted by the population size. It is easy to make the elite archive of DTEA unconstrained in the whole evolutionary process. Yen and Lu have designed the population growth and decline strategies to determine if an individual will survive or be eliminated based on some qualitative indicators [6]. The similar strategies can also be used to extend DTEA [34]. Considering the diversity maintenance, we can also add a density estimation into DTEA [34]. 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 according to their count values first and then their density values.

4. Experiments and discussions

In this section, we first observe the computational complexity of *DT* through experiments, and then validate the proposed DTEA and compare its effectiveness and efficiency with other benchmark MOEAs (i.e., NSGA-II and SPEA2). NSGA-II and SPEA2 are implemented in C according to their description in the literatures [2,5] and PISA [35]. We also implement the improved version of NSGA-II by Jensen [4], which is called NSGA-QS in the paper. The experiments are carried out on a 3 GHz and 512 M RAM Pentium IV computer running Windows 2000.

4.1. Computational complexity experiments

In this section, two groups of experiments are done to observe the computational complexity of *DT*. When a node is inserted into a *DT*, we calculate the number of steps that the node compares with other nodes. On the other hand, when the worst node is deleted, we calculate the number of steps that the algorithm spends on searching for the node. The experiments have the following two steps. First, we randomly generate some individuals whose objective values range from -100 to 100; and create a *DT* with these individuals. Second, we insert a random individual whose objective values also randomly range from -100 and 100 into the *DT*, and calculate the number of comparisons. At the same time, we delete the worst node, and calculate the number of searches. The second step (i.e., inserting and deleting) is repeated for 50 times, and then we calculate the average number of comparisons and searches. In order to further eliminate the randomness, we repeat the above process (including steps 1 and 2) 50 times.

Fig. 6(a) illustrates the relation between the number of comparisons and the population size for different number of objectives when a node is inserted into a *DT*. With the increase of the population size, an inserted node is expected to be compared with more nodes and thus the number of comparisons increases. When the number of objectives is 1 (i.e., $M = 1$, a single-objective problem), the objective vector becomes a scalar. In this situation, the comparison result of two solutions is either 1 or -1 (i.e., dominating or dominated) without considering the situation that the two vectors are identical. As a consequence, the *DT* degrades to a list with the right-link field being NULL, and inserting a new node is a straight insertion sort [27] with computational complexity $O(N)$ (N is the population size). The experiments also discover that the number of comparisons approximates $N/2$. As the number of objectives changes from 2 to 8, the number of comparisons increases and the increase rate is also growing. We consider the reason is that with the increase of the number of objectives, the size of nondominated solutions increases, so the inserted node is expected to be compared with more nondominated nodes. We also observe that the number of comparisons is larger than the straight insertion sort of a list (i.e., when $M = 1$) when the number of objectives is 8 and the population size is smaller than 400. It may indicate that the *DT* could be similar to the method proposed by Jensen [4] in the aspect that the algorithm has an inferior efficiency when the number of objectives is large and the population size is small. Since the plots in Fig. 6(a) all seem linear and these plots are in log-scales, we can empirically say that the expected processing time T follows $T = \beta N^{\alpha}$, where β and α are two constants and can be estimated by linear regression. We fit a linear regression model with the data and the results show the parameter α is 0.985, 0.339, 0.365, and 0.567 for M being 1, 2, 5 and 8, respectively.

Fig. 6(b) illustrates that the relation between the number of searches and the population size in terms of the different numbers of objectives when deleting the worst node. With the increase of the population size, the *DT* also requires more searches to find the worst node (i.e., the leftmost node). When the number of objectives increases, the number of nondominated nodes becomes large rapidly, and thus the left sub-tree of a *DT* becomes small, so it is easier to find the worst node. That is the reason why the number of searches decreases with the increase of the number of objectives. Especially when the number of objectives is larger than 5, the number of searches is very small, which indicates the *DT* is degenerating to a list with the left-link field being NULL. When the number of objectives is 1, we find the number of searches approximates N. The reason is same to that in Fig. 6(a). In this situation, the *DT* degrades to a list with the right-link field being NULL. Thus, the worst node is at the bottom of the list, so the algorithm needs to search all N nodes in order to find the target node. We also estimate the parameter α with linear regression for the four plots here. Parameters α are 0.986, 0.498, 0.237 and 0.218 when M are 1, 2, 5, and 8, respectively. This further confirms that the time consumed in the eliminating strategy becomes less when the number of objectives becomes larger.

From the experiments, we can find an interesting fact that the list is a special case of *DT* with the right-link field being NULL when $M = 1$ and the *DT* construction algorithms become a classical straight inserting sort algorithm of the list in this situation. We also find that the number of comparisons for inserting becomes larger

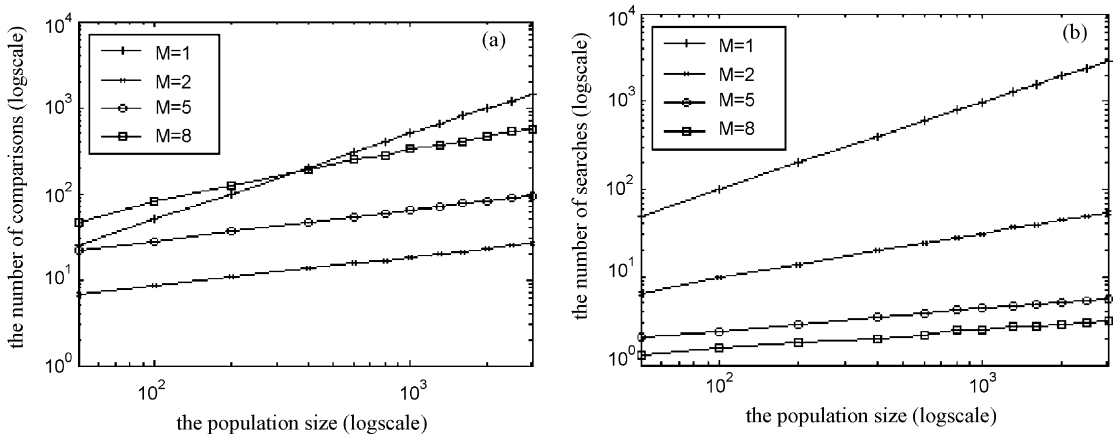


Fig. 6. (a) The relation between the number of comparisons and the population size when inserting a node. (b) The relation between the number of searches and the population size when deleting the worst node. *M* represents the number of objectives.

and the number of searches for deleting becomes smaller as the number of objectives increases. We will further investigate the impacts of the number of objectives in the experiments in Section 4.3.

Considering some popular fitness assignment processes in which each individual should be compared with all the other individuals with the complexity $O(N)$, we find that *DT* reduces the unnecessary comparisons significantly, and thus its complexity is much less than that of other methods. Compared to the experimental results of IDG proposed by Alberto and Mateo [25], *DT* is also more efficient when inserting or deleting a node. Jensen has systemically analyzed the computational complexity of some contemporary MOEAs [4]. Two classical MOEAs—SPEA2 and NSGA-II both have the complexity $O(MN)$ for the individual-based fitness assignment. And the complexities of the diversity strategy in SPEA2 and NSGA-II are $O(MN)$ and $O(M \ln N)$, respectively. Observing the experiments above, we can find that the computational complexities of DTEA are smaller than that of SPEA2 and NSGA-II. Even compared to the fast algorithms proposed by Jensen whose complexities are $O(\log^{M-1} N)$ in the fitness assignment and $O(M \ln N)$in the diversity strategy [4], DTEA is also competitive. We will further compare the efficiency of these algorithms in Section 4.3.

4.2. Comparison of effectiveness

In this study, six popular test functions are included, and these algorithms will be compared in three aspects: convergence to the Pareto front, maintenance of diversity and running time. NSGA-QS will not be included in these tests, since it only speeds up NSGA-II.

4.2.1. Test functions and performance assessment

The first two popular test functions QV [5] and KUR [2,5] have two objectives. The other test functions DTLZ1-DTLZ4 have three objectives, and they are widely used as the test functions with three objectives by many researchers [4,6,36].

To fairly compare these three algorithms, they all use the simulated binary crossover (SBX) and polynomial mutation [37]. The population size is 100; the archive size is 100; the distribution indexes for crossover and mutation operators are $\eta_c = 20$ and $\eta_m = 20$, respectively; the crossover probability is 1; and the mutation probability is $1/n$ for all test functions, where n is the number of variables. The numbers of the evaluated individuals are set as follows: 15,000 for QV and KUR; 30,000 for DTLZ1 and DTLZ2; 50,000 for DTLZ3; and 20,000 for DTLZ4. The number of generations of SPEA2 and NSGA-II is equal to the number of the total evaluated individuals divided by the population size. The number of generations of DTEA is equal to the number of the evaluated individuals divided by two, since the SBX only generates two offspring and DTEA deletes the two worst individuals in one generation. In DTEA, two parent individuals are randomly selected from the current population for crossover. The experimental results are the average results in 30 runs.

The measure criteria of solutions are an important matter in multi-objective optimization. Because of its complication, there are no consensus measure criteria accepted by all researchers [38]. Two popular measure criteria are used in this paper.

Let $\mathbf{X},\mathbf{Y} \subseteq V$ be two sets of objective vectors, the function C maps the ordered pair $(\mathbf{X}, \mathbf{Y})$ to the interval [0,1]:

$$C(\boldsymbol{X},\boldsymbol{Y}) = \frac{|\{b \in \boldsymbol{Y}; a \in \boldsymbol{X} : a \preceq b\}|}{|\boldsymbol{Y}|} \qquad (3)$$

$C(\mathbf{X},\mathbf{Y})$ is used to compare the convergence of two Pareto optimal sets [6,21]. The value $C(\mathbf{X},\mathbf{Y}) = 1$ means that all solutions in $\mathbf{Y}$ are Pareto dominated by solutions in $\mathbf{X}$. The opposite, $C(\mathbf{X},\mathbf{Y}) = 0$ represents the situation when none of the solutions in $\mathbf{Y}$ are covered by the set $\mathbf{X}$. Note that both $C(\mathbf{X},\mathbf{Y})$ and $C(\mathbf{Y},\mathbf{X})$ have to be considered, since $C(\mathbf{Y},\mathbf{X})$ is not necessarily equal to $1 - C(\mathbf{X},\mathbf{Y})$.

Let S be the Pareto optimal solutions in the final population; d_i is the Euclidean distance between two consecutive solutions in S in the objective space; the parameter $\bar{d}$ is the average of these distances. The diversity is computed as follows:

$$\Delta = \sum_{i=1}^{|S|} \frac{|d_i - \bar{d}|}{|S|} \qquad (4)$$

Δ measures the extent of spread achieved among the obtained solutions [39]. A good distribution would make all distance d_i equal to $\bar{d}$ and thus make Δ equal to 0.

4.2.2. Results and discussions

Table 1 shows the comparison of the three MOEAs in terms of the convergence metric C. DTEA performs better in QV and KUR obviously, whereas it performs slightly worse in DTLZ1. For other functions, they have close performance. Table 2 shows the results in terms of the distribution metric Δ. These three algorithms have very close diversity. The diversity of DTEA is not worse than other two algorithms except for KUR. All the results are close to 0, which means all the three algorithms obtain good diversities. Table 3 shows the running time of these three MOEAs. The running time of DTEA is significantly less than that of the other two algorithms for all test functions. Between the two benchmark algorithms, since the efficient density estimation algorithm and fast-nondominated

Table 1
Convergence comparison of different algorithms using C. D is DTEA; S is SPEA2; and N is NSGA-II. For each result, the first row is the average value, and the second row is the variance.

	QV	KUR	DTLZ1	DTLZ2	DTLZ3	DTLZ4
C (D, S)	1	0.594	0.017	0.089	0.019	0.168
	0	0.002	0.597	0.112	0.210	0.310
C (D, N)	1	0.554	0.007	0.119	0.021	0.079
	0	0.003	0.607	0.212	0.185	0.124
C (S, D)	0	0.004	0.718	0.103	0.061	0.182
	0	0.001	0.567	0.135	0.223	0.296
C (N, D)	0	0.004	0.562	0.082	0.058	0.195
	0	0.001	0.590	0.311	0.199	0.154

Table 2
Distribution comparison of different algorithms using Δ. For each result, the first row is the average value, and the second row is the variance.

	QV	KUR	DTLZ1	DTLZ2	DTLZ3	DTLZ4
DTEA	0.004	0.042	0.137	0.235	0.249	0.287
	0.004	0.003	0.444	0.032	0.354	0.023
SPEA2	0.004	0.015	0.116	0.246	0.268	0.253
	0.001	0.001	1.463	0.015	1.196	0.117
NSGA-II	0.004	0.016	0.130	0.233	0.253	0.284
	0.001	0.001	0.817	0.019	0.301	0.105

Table 3
Compare running time of different algorithms. The time unit is millisecond. For each result, the first row is the average value, and the second row is the variance.

	QV	KUR	DTLZ1	DTLZ2	DTLZ3	DTLZ4
DTEA	657	239	621	1084	909	725
	0.022	0.015	0.012	0.011	0.018	0.004
SPEA2	7247	3099	10342	10638	17619	8332
	0.124	0.151	0.005	0.015	0.025	0.092
NSGA-II	1651	773	2042	2554	3511	1759
	0.112	0.092	0.008	0.009	0.021	0.052

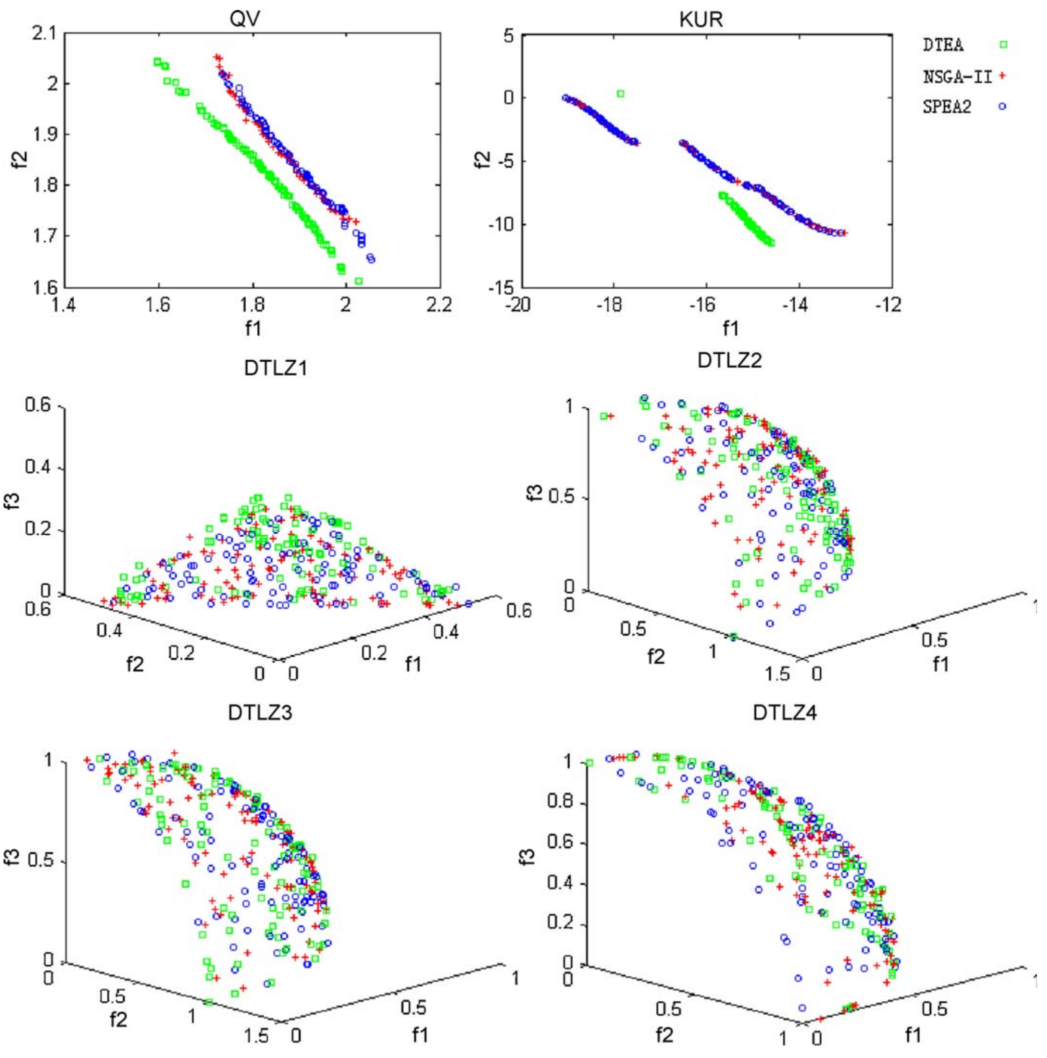


Fig. 7. Visualization of the Pareto optimal set of the test functions.

sort algorithm are used in NSGA-II, NSGA-II is obviously faster than SPEA2.

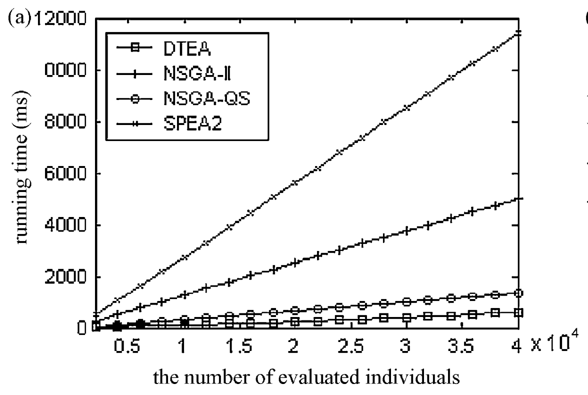
In order to further demonstrate the results obtained by these algorithms, 100 individuals are randomly selected from the final result Pareto optimal sets and are visualized in Fig. 7. The final result sets are constructed through following steps: the solutions of 30 runs are combined to form a new set; and then the Pareto optimal set in the new set is calculated. We can observe that the optimal solutions found by DTEA dominate those found by NSGA-II and SPEA2 in QV and KUR. However, DTEA could not find one discontinuous front in KUR. In addition, DTEA shows better diversity in QV. As for the four three-objective test functions, generally speaking, the three algorithms all converge to the true Pareto optimal front and maintain good diversities.

The performance experiments demonstrate the effectiveness of DTEA. DTEA is not only able to find the true (or approximate) Pareto optimal front, but also maintain a good distribution of solutions without any special diversity strategy. Moreover, it is much faster than the other two algorithms. Please note that we do not use any specially designed operators, diversity maintenance strategy, and specially tuned parameters settings for DTEA in the experiments. Whereas, compared to the two well-known MOEAs: SPEA2 and NSGA-II, DTEA obtains the competitive solutions with much less time. As mentioned in Section 3.2, the eliminating strategy in DTEA may not be functional in maintaining the diversity when all nodes are nondominated. However, the experimental results show that DTEA maintains good diversities in most testing problems. We consider there may be two reasons. In the situation when the *DT* degrades into a list, once a new-generated node dominates one of the nodes in the sibling chain, the list changes back into a balanced *DT*, in which case the eliminating strategy becomes functional again in maintaining the diversity. In addition, DTEA is a steady-state algorithm that only deletes very limited number of individuals (e.g., two in this experiment) in one generation. We consider the steady-state strategy may be beneficial for maintaining the diversity.

4.3. Comparison of efficiency

In this section, we further compare the running time of different algorithms including NSGA-II, NSGA-QS, SPEA2, and DTEA. The experiments include three aspects: the running time in the evolutionary process, the running time for different population sizes and the running time for different number of objectives.

The test function is the problem SPH-*m* defined in Eq. (5) [40]. SPH-*m* is a scalable function and it is easy to extend the number of objectives. The results shown here are the average results in 30 runs. There are some common parameters settings for the three experiments: $\eta_c = 20$; $\eta_m = 20$; the crossover probability is 1; and



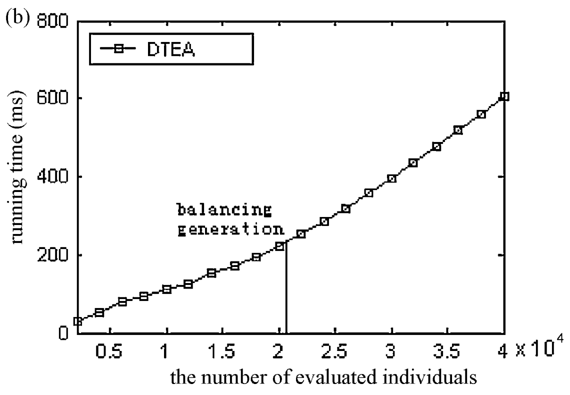


Fig. 8. (a) The relation of running time and generations. (b) The relation of running time and generations in DTEA.

the mutation probability is $1/n$.

$$f_j(\mathbf{x}) = \sum_{1\le i\le n, i\ne j} (x_i)^2 + (x_j - 1)^2 \quad 1 \le j \le M \quad x_i \in [-10, 10]^n \tag{5}$$

4.3.1. Running time in evolutionary process

The first experiment is to observe the running time in the evolutionary process. The parameters are set as follows: the population size is 200; the number of variables is 10; the number of objectives is 2. The number of total evaluated individuals is 40,000. We calculate the average running time of every 2000 individuals evaluated. Fig. 8 shows the relation of the running time and the generations. For a fair comparison, the number of the evaluated individuals (instead of the generations) is shown in the x-axis.

It is obvious that DTEA is the fastest one and it is even faster than NSGA-QS. SPEA2 is the slowest one. NSGA-QS speeds up NSGA-II indeed. As the number of evaluated individual increases, the differences of the four algorithms become more and more significant. Fig. 8(b) further investigates the trend of the running time of DTEA as the number of evaluated individual increases. We can observe a turning point in the curve in Fig. 8(b) and the running time of DTEA has different growing ratios before and after the turning point. We also find that the turning point is the generation at which all individuals in the *DT* are nondominated for the first time. We name this generation the *balancing generation* and define it as following:

Definition 5. *Balancing generation.* The *Balancing generation* is the generation at which all individuals in the *dominating tree* are nondominated for the first time.

The *DT* degrades into a list at the point of *balancing generation*. The *balancing generation* is 10,613 in this example (the number of evaluated individuals is 21,226, since 2 for each generation). We show the *balancing generations* of all those test functions in Section 4.2 in Table 4. From Tables 3 and 4, we can find that the *balancing generation* affects the *DT*'s efficiency. Fig. 9 illustrates the effects more clearly. We can observe a trend that when the ratio of the *balancing generation* vs. the total generations is larger, the speedup of DTEA vs. both NSGA-II and SPEA2 are also larger. We consider the reason is that after the *balancing generation*, there are more nondominated individuals in the *DT* and based on the *DT* construction algorithms, the number of comparisons increases when a new individual is inserted. Thus, the *DT* shows better efficiency when the *balancing generation* is larger. On the other hand, we also may say that *DT* is more efficient before the *balancing generation* than after it.

Table 4
The balancing generation of each test function for DTEA.

	QV	KUR	DTLZ1	DTLZ2	DTLZ3	DTLZ4
Total generations	7500	7500	15,000	15,000	25,000	10,000
Balancing generation	1052	1269	3179	625	6451	1016

4.3.2. Running time for different population sizes

The second experiment is to observe the running time for different population sizes. The parameters in the experiment are set as follows: the number of variables is 10; and the number of objectives is 2. There are 11 different settings of population sizes ranging from 50 to 1000. The generation setting is 50 for SPEA2, NSGA-QS and NSGA-II. For a fair comparison, the generation of DTEA is calculated as $50 \times N/2$, where N is the population size. The result is demonstrated in Fig. 10(a), where the plot is in a logarithmic scale and the average computing time is a function of the population size.

It is easy to observe from Fig. 10(a) that DTEA is the fastest one in all cases; and SPEA2 is the slowest one. NSGA-QS is faster than NSGA-II. As the population size increases, DTEA is much faster than the other algorithms. The lines in Fig. 10(a) all seem linear, which also indicates that the processing time follows a $T = \beta N^{\alpha}$ relation. We estimate parameters α with linear regression. For NSGA-II and SPEA2, α is 1.97, which confirms the $O(N^2)$ processing time of these algorithms. For NSGA-QS, α is 1.22 and for DTEA, α is 1.07, which is close to $O(N)$. When the population size becomes larger, the

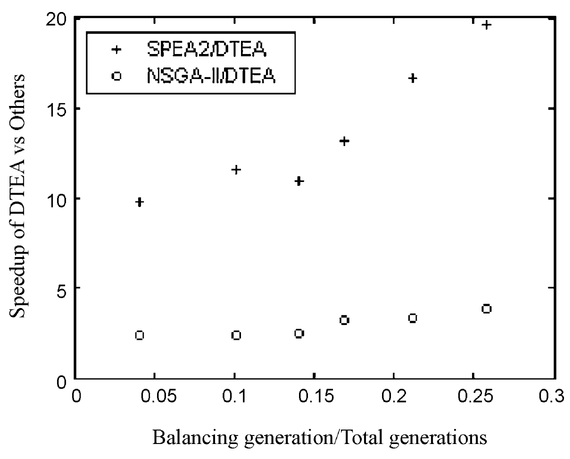


Fig. 9. The relation of DTEA's speedup and the balancing generation. From left to right, the six "+" (or "○") represents the relation of DTLZ2, DTLZ4, QV, KUR, DTLZ1 and DTLZ3, respectively.

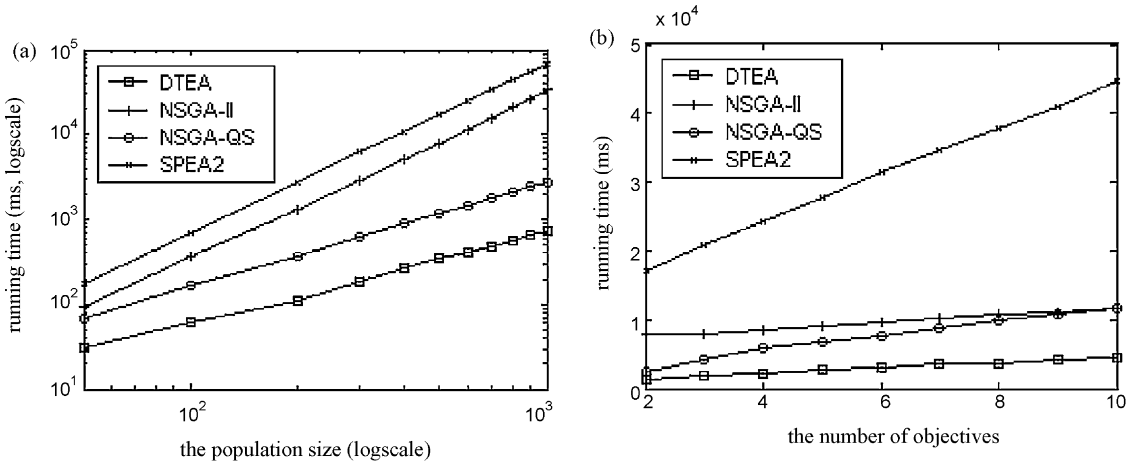


Fig. 10. (a) The relation of running time and the population size. (b) The relation of running time and the number of objectives.

balancing generation also becomes larger, so the increase of the running time of DTEA is slower than the other methods with the increase of the population size.

4.3.3. *Running time with different number of objectives*

The third experiment is to observe the running time with different number of objectives. The experiment parameters are set as follows: the population size is 200; and the number of variables is 20. The number of objectives ranges from 2 to 10. The generation is 300 for SPEA2, NSGA-QS and NSGA-II, and 30000 for DTEA.

The experimental result is demonstrated in Fig. 10(b). For each case, DTEA is still the fastest one, and SPEA2 is the slowest one. With the increase of the number of objectives, the increase rate of running time is relatively slow for NSGA-II and DTEA, and fast for SPEA2 and NSGA-QS. The running time of NSGA-QS becomes larger than NSGA-II when the number of objectives is 10. Generally speaking, with the increase of the number of objectives, the solutions become nondominated rapidly and the *balancing generation* becomes small, so the time of inserting a node into *DT* increases quickly. However, in this situation, the left sub-tree of the *DT* become smaller; so it costs less time to delete the worst node. This is the reason why the running time of DTEA increases slowly. We also observe that NSGA-QS has the inferior performance when the number of objectives is large, which is consistent with Jensen's report [4]. As we know, many data structures and approaches have an inferior performance when a large number of solutions become nondominated, such as IDG [25], Quad-tree [24] and Jensen's fast algorithms [4]. Whereas, the experiments show that DTEA has the advantage of maintaining high efficiency in this situation because the time for deleting the worst node could be saved.

4.3.4. *Discussion of running time*

Through the three experiments above, we find that DTEA is always much faster than NSGA-QS, SPEA2 and NSGA-II. SPEA2 is always the slowest one. NSGA-QS speeds up NSGA-II.

The *balancing generation* affects the efficiency of the *dominating tree*. The larger the *balancing generation* is, the less time-consuming it is to construct a *DT*. In fact, the *balancing generation* reflects the selection pressure: the population is under more selection pressure before the *balancing generation* and less pressure after that. When the population size is large, the *balancing generation* will also be large and thus the increase rate of the running time of the *DT*-based algorithms is significantly lower than that of the other algorithms as shown in the experiments. As a consequence, DTEA demonstrates more significant advantages in computing time when the population size is large.

5. Conclusions

In this paper, a fast multi-objective evolutionary algorithm based on *dominating tree*, named DTEA, is proposed. The *dominating tree* is a binary tree that can store the three-valued relation existing in MOPs, namely, dominating, dominated, and nondominated. The construction algorithms guarantee that the *dominating tree* is able to effectively handle the whole population of MOEA. DTEA integrates the convergence strategy and diversity strategy into the *dominating tree*. The eliminating strategy in DTEA based on the *dominating tree* can maintain the diversity without extra expenses. The experiments show that DTEA can produce statistically competitive results compared to SPEA2 and NSGA-II on six popular multi-objective optimization benchmark problems. Moreover, DTEA is much faster than the other three benchmark MOEAs, and this advantage becomes more significant when the population size is large. As we know, the large population is desirable for MOEAs to avoid premature convergence. However, because of the huge time complexity of the current fitness assignment methods in MOEAs, the large population size is usually unfeasible in real world applications. As a consequence, we consider that DTEA may have advantages in both requiring less time complexity and obtaining better solution quality because we can increase the population size of DTEA with much less increase rate of computing time than the other MOEAs.

There are still many interesting researches to be done to exploit DTEA further. We have illustrated the effect of the *balancing generation* on the efficiency of DTEA. The *balancing generation* may also affect the performance and the speed of convergence. Future research may explore this relation and the factors that affect the *balancing generation*.

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 (Nos. 60905025, 60775035, 60805041), 863 National High-Tech Program (No. 2007AA01Z132) and National Basic Research Priorities Programme (Nos. 2003CB317004, 2007CB311004).

References

[1] C.A.C. Coello, Evolutionary multiobjective optimization: a historical view of the field, IEEE Computational Intelligence Magazine 1 (1) (2006) 28–36.

[2] 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.
[3] D.A.V. Veldhuizen, G.B. Lamont, Multiobjective evolutionary algorithms: analyzing the state-of-the-art, Evolutionary Computation 18 (2) (2000) 125–147.
[4] 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.
[5] E. Zitzler, M. Laumanns, L. Thiele, SPEA2: improving the strength Pareto evolutionary algorithm, TIK-Report 103, ETH Zentrum, Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001.
[6] 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.
[7] S.Y. Zeng, L.S. Kang, L.X. Ding, An orthogonal multi-objective evolutionary algorithm for multi-objective optimization problem with constraints, Evolutionary Computation 12 (1) (2004) 77–98.
[8] C.M. Fonseca, P.J. Fleming, An overview of evolutionary algorithms in multiobjective optimization, Evolutionary Computation 3 (1995) 1–16.
[9] D.E. Goldberg, Generic Algorithms in Search Optimization and Machine Learning, Addison Wesley, Massachusetts, 1989.
[10] 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.
[11] N. Srinivas, K. Deb, Multiobjective optimization using nondominated sorting in genetic algorithms, Evolutionary Computation 2 (3) (1995) 221–248.
[12] J. Horn, N. Nafpliotis, D.E. Goldberg, A niched Pareto genetic algorithm for multiobjective optimization, in: Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE Press, 1994, pp. 82–87.
[13] J.D. Knowles, D.W. Corne, Approximating the nondominated front using the Pareto archived evolution strategy, Evolutionary Computation 8 (2000) 149–172.
[14] 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.
[15] Y. Xin, Y. Xu, Recent advances in evolutionary computation, Journal of Computer Science & Technology 21 (1) (2006) 1–18.
[16] S. Agrawal, B.K. Panigrahi, M.K. Tiwari, Multiobjective particle swarm algorithm with fuzzy clustering for electrical power dispatch, IEEE Transactions on Evolutionary Computation 12 (5) (2008) 529–541.
[17] M.G. Gong, L.C. Jiao, H.F. Du, L.F. Bo, Multiobjective immune algorithm with nondominated neighbor-based selection, Evolutionary Computation 16 (2) (2008) 225–255.
[18] Q.F. Zhang, A.M. Zhou, Y. Jin, RM-MEDA: a regularity model-based multiobjective estimation of distribution algorithm, IEEE Transactions on Evolutionary Computation 12 (1) (2008) 41–63.
[19] A.J. Nebro, F. Luna, E. Alba, B. Dorronsoro, J.J. Durillo, A. Beham, AbYSS: adapting scatter search to multiobjective optimization, IEEE Transactions on Evolutionary Computation 12 (4) (2008) 439–457.
[20] Q. Zhang, H. Li, MOEA/D: a multi-objective evolutionary algorithm based on decomposition, IEEE Transaction on Evolutionary Computation 11 (6) (2007) 712–731.
[21] E. Zitzler, K. Deb, L. Thiele, Comparison of multiobjective evolutionary algorithms: empirical results, Evolutionary Computation 18 (2) (2000) 173–195.
[22] 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.
[23] J.E. Fieldsend, R.M. Everson, S. Singh, Using unconstrained elite archives for multiobjective optimization, IEEE Transaction on Evolutionary Computation 7 (2003) 305–323.
[24] 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.
[25] I. Alberto, P.M. Mateo, Representation and management of MOEA populations based on graphs, European Journal of Operational Research 159 (1) (2004) 52–65.
[26] R.M. Everson, J.E. Fieldsend, S. Singh, Full elite sets for multi-objective optimization, in: Proceedings of 5th International Conference on Adaptive Computing in Design and Manufacture (ADCM 2002), 2002, pp. p343–354.
[27] Robert Sedgewick, Algorithms in C++, Parts 1–4-Fundamentals, Data Structures, Sorting, and Searching, in: Person Education, third edition, 2002, pp. 515–521.
[28] 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.
[29] Z. Yan, L.S. Kang, R. Mckay, SEEA for multi-objective optimization: reinforcing elitist MOEA through multi-parent crossover, steady elimination and swarm hill climbing, in: Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning, 2002, pp. 21–26.
[30] K. Rajeev, R. Peter, 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.
[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] T. Baeck, Evolutionary Algorithms in Theory and Practice, Oxford University Press, 1996.
[33] J.E. Fieldsend, S. Singh, A multi-objective algorithm based upon particle swarm optimization, an efficient data structure and turbulence, in: Proceedings of 2002 Workshop on Computational Intelligence, UK, (2002), pp. 37–44.
[34] C. Shi, Q.Y. Li, Z.Y. Zhang, Z.Z. Shi, An improved multiobjective evolutionary algorithm based on dominating tree, in: PRICAI 2006, 2006, 691–700.
[35] S. Bleuler, M. Laumanns, L. Thiele, E. Zitzler, PISA—a platform and programming language independent interface for search algorithms, in: Conference on Evolutionary Multi-Criterion Optimization (EMO 2003), 2003, 494–508.
[36] M. Farina, K. Deb, P. Amato, Dynamic multiobjective optimization problems: test cases, approximations, and applications, IEEE Transaction on Evolutionary Computation 8 (5) (2004) 425–442.
[37] K. Deb, R.B. Agrawal, Simulated binary crossover for continuous search space, Complex Systems 9 (1995) 115–148.
[38] 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.
[39] K. Deb, A. Samir, et al., A fast elitist nondominated sorting genetic algorithm for multi-objective optimization: NSGA-II, KanGAL Report No. 200001, Kanpur, PIN 208 016, India.
[40] M. Laumanns, G. Rudolph, H.P. Schwefel, Mutation control and convergence in evolutionary multiobjective optimization, in: Proceedings of the 7th International Mendel Conference on Soft Computing (MENDEL 2001), Brno, Czech Republic, June, 2001.