An Efficient Fitness Assignment Based on Dominating Tree

Chuan Shi[1, 2] Zhongzhi Shi[2] Bin Wu[1]
[1] School of Computer Science and Technology,
Beijing University of Posts and Telecommunications, 100876
[2] Institute of Computing and Technology, Chinese Academy of Sciences, 100080
{shic, shizz}@ics.ict.ac.cn, *wubin@bupt.edu.cn*

Abstract

It has seen a surge of research activity on multiobjective optimization using evolutionary algorithms in recent years. The majority of these algorithms use fitness assignment based on Pareto dominance. The fitness assignment not only decides the algorithm's performance, but also is one of the main time-consuming components. This paper proposes an efficient fitness assignment based on dominating tree (DT). The dominating tree is a binary tree with the dominating information of individuals, which can represent three-valued relationship existing in Pareto dominance. We apply the dominating tree as an effective fitness assignment that can improve general multiobjective evolutionary algorithms. The simulation results also prove it.

1. Introduction

Over the last twenty years, there has been an increasing interest in applying evolutionary algorithms to multiobjective optimization problems (MOP). This research is highly relevant to real world applications, since real world optimization problems often involve several conflicting objectives for which a tradeoff must be found. The presence of multiple conflicting objectives in an optimization problem means that no single solution is globally optimal, unless priorities can be assigned to the objectives. It is usually difficult or even impossible to assign priorities and it makes an algorithm returning a set of promising solutions is preferable. For this reason, most contemporary multiobjective evolutionary algorithms (MOEAs) are designed to return a set of promising solutions, from which a solution can be picked by human experts.

Many effective MOEAs have been suggested to solve the problem [1]. These MOEAs use Pareto dominance to guide the search, and return a set of nondominated solutions as result. Consider, without loss of generality, the minimization of the n objectives $f_k(\mathbf{x})$, $k = 1,\cdots n$, of a vector function $\mathbf{f}$ of a vector variable $\mathbf{x}$ in a universe Ω, where $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}),\cdots, f_n(\mathbf{x}))$. A vector $\mathbf{u} = \mathbf{f}(\mathbf{x}_\mathbf{u}) = (u_1,\cdots,u_n)$ Pareto dominates $\mathbf{v} = \mathbf{f}(\mathbf{x}_\mathrm{v}) = (v_1,\cdots,v_n)$, denotes as $\mathbf{u} \preceq \mathbf{v}$ if and only if

$$\forall i \in \{1,\cdots,n\} \quad u_i \le v_i \wedge \exists\, i \in \{1,\cdots,n\} \quad u_i < v_i \qquad (1)$$

A decision vector $\mathbf{x}_\mathrm{u} \in \Omega$ is said to be Pareto optimal if and only if there is no $\mathbf{x}_\mathrm{v} \in \Omega$ for which $\mathbf{f}(\mathbf{x}_\mathrm{v}) \preceq \mathbf{f}(\mathbf{x}_\mathrm{u})$. The set of all Pareto optimal decision vectors is called the Pareto optimal set. The corresponding set of the objective vector is called the nondominated set, or Pareto front.

Even though contemporary MOEAs work with several objectives simultaneously, they still transform all of the objectives into one fitness measure. This necessary, since what makes an evolutionary algorithm (EA) wok is the selection of highly fit individuals over less fit individuals. The process is also called fitness assignment, which evaluates the quality of individuals. The fitness assignment is the key component in MOEAs, which decides the algorithm's performance. The fitness assignment is usually a costly matter in terms of processing time. Most Pareto-based fitness assignment require that each solution is compared with a large number of other solutions, which make many MOEAs have computational complexity bounded by $O(MN^2)$ (where M is the number of objectives and N is the population size)[2],[3]. The N^2 factor means that the processing time becomes large for large population sizes. Since in some situations it is desirable to use large population sizes, especially when the number of conflicting objectives is large, it is important to reduce the process time.

The main result of this paper is development of an effective fitness assignment approach. Through careful analysis, we find that there are many unnecessary comparisons in the process of the fitness assignment. Reducing the unnecessary comparisons may be a shortcut to reduce its computational complexity. Contrasting with binary search tree (BST), we propose a dominating tree

0-7695-3019-2/07 $25.00 © 2007 IEEE
DOI 10.1109/ICDMW.2007.68

to effectively reduce the unnecessary comparisons. As we known, the nodes in BST are partial order, and their relationships are two-valued: $<$ or $\geqslant$. However, the solutions in MOP may be nondominated with each other, which make their relationships three-valued: dominating, dominated, and nondominated. The dominating tree is designed to represent the three-valued relationship. The left link of a node in dominating tree points to a dominated node and its right link pointes to a nondominated node. The construction algorithms of dominating tree keep some useful properties, which indicate it can be applied in evolutionary multiobjective optimization (EMO). We propose the fitness assignment based on dominating tree, and apply it to replace the non-dominated-sort procedure of NSGA-II [6]. The simulation results show that the improved NSGA-Ⅱ is significantly faster than NSGA-Ⅱ and SPEA2 [3]. On the other hand, the quality of the results obtained by the improved NSGA-Ⅱ is competitive with that of NSGA-Ⅱ and SPEA2.

2. Dominating Tree and its Properties

2.1 Structure of Dominating Tree

Comparing with solutions in SOPs, solutions in MOPs are vectors, and their relationships are decided by Pareto dominance. For MOP, two solutions, $\mathbf{x}_\mathbf{u}$ and $\mathbf{x}_\mathbf{v}$, can be evaluated by the following function 2 ($\mathbf{u}=\mathbf{f}(\mathbf{x}_\mathbf{u})=(u_1,\cdots,u_n)$, $\mathbf{v}=\mathbf{f}(\mathbf{x}_\mathbf{v})=(v_1,\cdots,v_n)$). If $Better(\mathbf{u},\mathbf{v})=1$, $\mathbf{x}_\mathbf{u}$ is better than $\mathbf{x}_\mathbf{v}$; if $Better(\mathbf{u},\mathbf{v})=0$, $\mathbf{x}_\mathbf{u}$ is nondominated with $\mathbf{x}_\mathbf{v}$, which means some objective functions of $\mathbf{x}_\mathbf{u}$ is smaller than that of $\mathbf{x}_\mathbf{v}$ but others are larger.

$$Better(\mathbf{u},\mathbf{v})=\begin{cases}1 & \mathbf{u}\preceq\mathbf{v}\\ -1 & \mathbf{v}\preceq\mathbf{u}\\ 0 & \text{nondominated}\end{cases} \quad (2)$$

The *Better* function formally defines the relationship of solutions in SOP and MOP, and it also points out the important difference between these two problems. The objective space solution in SOP is real number, and their relationships of solutions are a two-valued: $<$ or $\geq$; which constitutes the partial order. However, the objective space solution in MOP is vector, their relationships are a three-valued: $\mathbf{u}\preceq\mathbf{v}$, $\mathbf{v}\preceq\mathbf{u}$ or nondominated. The function also explains the reason why MOP is more complex than SOP. Since the relationship of solutions in SOP is order partial, many sorting algorithms can be used in the fitness assignment of SOP, for example bubble sorting, shell sorting etc. The effective BST is also used to store the relationship of real number. The computational complexity of these algorithms is usually smaller than $O(N^2)$. However, as far as we knew, there are few studies on sorting the vectors and storing their relationships. As shown in Figure 1, the common method is to compare a vector with others and store their results with linear link, which causes N^2 process time. Can we design an efficient method to compare individuals and store their relationships?

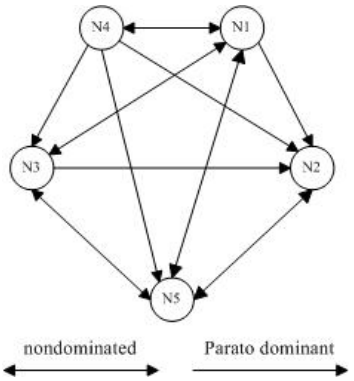

Figure 1. Illustration of the relationships among nodes obtained with linear link.

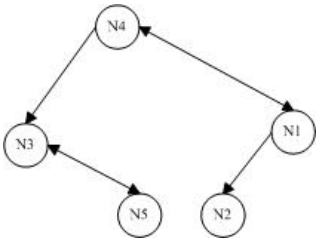

Figure 2. Illustration of the relationships among nodes in Figure 1 with dominating tree.

As a basal data structure, the binary search tree is used to sort the real number and store their relationships. The left link of a node in BST points to the smaller node, and its right link points to the larger node. When BST is used for heap sorting algorithm, the average complexity is $O(N\lg N)$, which confirms that it is an efficient algorithm. The BST is used for the two-valued relationship. For the three-valued relationship, the similar data structure may be used. The left link of a node in the binary tree points to the dominated node, and its right link points to the nondominated node. Taking Figure 2 for example, the relationships of five nodes can be contained in the binary tree.

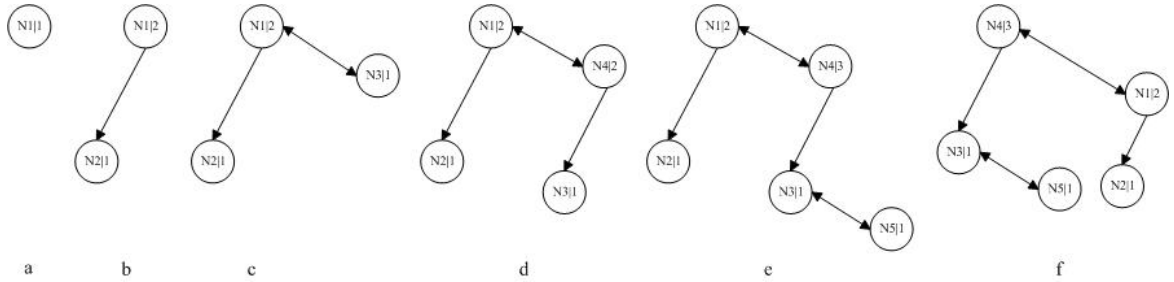

Figure 3. Illustration of the creating process of Figure 2. The left number in the node is its id, and the right number is its count. The sequence is alphabetic.

The tree is a novel binary tree; therefore, we call it a dominating tree. A dominating tree can be designed as follows.

Definition 1 (**dominating tree**): 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 subtree and the right subtree of that node.
2. Each node in the dominating tree has four fields: id, count, left-link, and right-link, where id registers which individual the node represents, count registers the size of its left subtree (including itself), left-link links to its left subtree whose root is dominated by that node, and right-link links to its right subtree whose root is nondominated with that node.

Compared to BST, DT has the similar definition except the new notion "sibling chain" that refers to the chain constituted by the root and its right-link nodes. Following the right-links, one can obtain the sibling chain. Taking Figure 2 for example, N3 is the left child of N4, and N4 is the parent of N3 and N5. N1 is the right sibling of N4. N1 and N4 is the sibling chain of the DT whose root is N4. Moreover, N3 and N5 also constitute a sibling chain.

If we consider the dominated relationship corresponds to the smaller than relationship in BST and the nondominated relationship correspond to the larger than or equal relationship in BST, and then a DT is equivalent to a BST. However, the relationship of nodes in BST is two-valued, whereas that of DT is three-valued. Thus although the construction of a DT is similar to that of a BST, they do have a difference.

2.2 Construction Algorithms of Dominating Tree

Since the creating process of a DT is similar to that of a BST, the construction algorithm of a DT is also a recursive algorithm. Unlike in BST, a new node inserted into the DT has three choices. When the new node is dominated by the root, it is inserted into the left subtree of the root, which is similar to the smaller than relationship in BST. When the new node is nondominated with the root, it is inserted into the right subtree of the root, which is similar to the larger than or equal relationship in BST. When the new node dominates the root, it is impossible that the new node is dominated by the nodes in the root's sibling chain[1]. So the new node should not only take the place of the root and let the root become its left subtree, but should also continue to compare with other nodes in its sibling chain. If there are nodes dominated by the new node, they should be deleted from the sibling chain and then be inserted into the left subtree of the new node.

Figure 3 demonstrates the process of creating the DT in Figure 2. The input order of these nodes is N1, N2, N3, N4, and N5, and the result of inserting each node is shown from Figure 3 (a) to Figure 3 (e) respectively. After the five nodes have been inserted, the DT is shown as Figure 3 (e). It is obvious that the DT is unbalanced since the count of N4 (which is 3) is larger than that of N1 (which is 2) in Figure 3 (e). To balance the tree, N4 with its left subtree moves along the sibling chain in the left direction as in Figure 3 (f). The action doesn't change the relationships of the nodes, but it makes the DT more balanced. Figure 3 (f) is the same as Figure 2. The balanced DT has the advantage of less cost for adding or deleting a node. In addition, it also enables the dominating tree to have some useful properties.

```
/*Creating a dominating tree. pTree is the root of the whole tree.
Input: all nodes in the population
Output: the dominating tree */
Link ConstructTree (Pop P){
  For each node (pNewnode) in the population P
    AddinTree(pTree, pNewnode);
  return pTree;
```

[1] If there is a node in the sibling chain dominating the new node, the node also dominates the root, since the relationship of Pareto dominance is transitive. However, the node is nondominated with the root, since they are in the same sibling chain.

```
}

/* Add pNewnode into the left subtree 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 )
   then pRoot->left-link = pNewnode;
   else     AddinSibling     (pRoot,     pRoot->left-link,
              pNewnode);
}

/* pNewnode is compared with pChild whose parent
node is pParent when pNewnode is inserted into
pParent's left subtree.*/
AddinSibling (Link pParent, Link pChild, Link
pNewnode){
   switch (Better (pNewnode, pChild))
   case 0:      //nondominated
      if (pChild->right-link == NULL)
      then pChild->right-link = pNewnode;
      else AddinSibling (pParent, pChild->right-link,
             pNewnode);
   case 1:     //dominating
      pNewnode takes the place of pChild;
      AddinTree (pNewnode, pChild);
      while there exists a pNode (a node in the
      pNewnode's sibling chain) and Better(pNewnode,
      pNode) ==1
      do{
            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 sibling chain of the parent's left subtree in
their count descending order. If the count of pMovenode
(a node in the sibling chain) increases, it moves along
the sibling chain in the left direction, or else in the right
direction. */
BalanceTree (Link pParent, Link pMovenode, int
direction){
   if(direction == L)
      while the count of pMovenode is larger than that of
      its left  nodes which is also in the same sibling
      chain
      do pMovenode with its left subtree moves along
      the sibling chain in the left direction;
   if(direction == R)
      while the count of pMovenode is smaller than that
      of its right nodes which is in the same sibling chain
      do pMovenode with its left subtree moves along
      the sibling chain in the right direction;
}
```

Figure 4. Pseudocode of dominating tree construction algorithms.

Based on the demonstration above, we have the dominating tree construction algorithms as shown in Figure 4. *ConstructTree* is the main loop of creating a *DT* according to nodes in the population. There are three functions used in the algorithm. *AddinTree* and *AddinSibling* demonstrate the action of a new node inserted into a DT. *BalanceTree* moves the node with the changing count along the sibling chain in the left or right direction, which makes the DT balanced. The algorithms fully utilize the properties of Pareto dominance and reduce the comparisons among nodes.

2.3 Properties of Dominating Tree

There are several useful properties of the dominating tree that can be derived from the construction process. For simplicity, we omit the detailed proof here.

Lemma 1. The sibling chain of a dominating tree only conserves all Pareto optimal nodes in the tree.

Lemma 2. The root of a dominating tree dominates all nodes in its left subtree.

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

3. Fitness Assignment Based on Dominating Tree

The high computational complexity of fitness assignment leads to long processing times for large population sizes, so it should be reduced if possible [5]. The creating process of a dominating tree is in fact the process of the fitness assignment. The dominating tree preserves the dominating relationships of solutions in the tree naturally. In NSGA-Ⅱ, a nondominated sorting approach is used for each individual to create a Pareto rank, which divides solutions into different fronts with different ranks [2]. In SPEA2, each individual in both the main population and elitist archive is assigned a strength value. On the basis of the strength value, the final rank value is determined by the summation of the strengths of the points that dominate the current individual [3].

To test the applicability of DT in MOEA, we replace non-dominated-sort process in NSGA-Ⅱ with a new fitness assignment strategy based on dominating tree. To satisfy NSGA-Ⅱ, the nodes in the dominating tree also are assigned rank value as its front. The node's rank can be assigned as follows.

Definition 2 (**Rank Assignment in DT**): the ranks of nodes in a dominating tree can be assigned as following steps:

1. The root's rank is assigned as 1.
2. If a node's rank is k, its right-link node's rank is k; and its left-link node's rank is k+1.

The node's rank value is used as its front. The rank assignment can be realized with the depth-first search algorithm or breadth-first search algorithm whose run-time complexity is $O(N)$.

4. Experiment and Discussion

In this section, we validate the proposed dominating tree through experiments. The improved version of NSGA-II by dominating tree is called NSGA-DT. Two well-known MOEAs -NSGA-II and SPEA2- are compared. The experiments include effectiveness and efficiency aspects. NSGA-II and SPEA2 are implemented in C according to their description in the literatures [2], [3]. The experiments are carried out on a 3GHz and 512M RAM Pentium Ⅳ computer running Windows 2000.

4.1 Test Functions and Performance Assessment

Five two-objective functions (ZDT1-ZDT4 and ZDT6) and four three-objective functions (DTLZ1—DTLZ4) will be used in the performance experiments. These functions are widely used as the benchmark problem in many researches [4]. These algorithms will be compared in three aspects: convergence to the Pareto front, maintenance of diversity and running time. Each of the algorithms runs for 20 times to obtain average results.

To fairly compare these three algorithms, they all use the simulated binary crossover (SBX) and polynomial mutation. The reasonable parameters are settled. The population size is 200; the archive size is 200; the distribution indexes for crossover and mutation operators are $\eta_c = 20$ and $\eta_m = 20$ respectively; the crossover probability is 0.9; the mutation probability is $1/n$ for all test functions, where n is the number of variables. The running generations are settled as follows: ZDT1-ZDT6 are 100; DTLZ1-DTLZ4 are 200.

We use two popular measure criterions in this paper. The first metric Υ measures the extent of convergence to a known set of Pareto optimal solutions [6]. The smaller the value of the metric, the better the convergence toward the Pareto optimal front. In all simulations performed here, we present the average $\overline{\Upsilon}$ and variance σ_Υ of this metric calculated for solutions set obtained in multiple runs. The second metric Δ measures the extent of spread achieved among the obtained solutions [6]. A good distribution would make Δ equal to 0. We also calculate $\overline{\Delta}$ and σ_Δ for solutions obtained in multiple runs.

4.2 Result and Discussion

Table 1 shows the mean and variance of the convergence metric Υ obtained using three MOEAs. On most test problems, the three algorithms have close results; and they all converge to the true optimal front. NSGA-DT finds the best convergence in ZDT2, ZDT4, DTLZ1, DTLZ2 and DTLZ4; it also finds the worst convergence in ZDT3. Similarly, NSGA-II obtains the best result in ZDT1, ZDT3, ZDT6 and DTLZ3, and the worst result in DTLZ1 and DTLZ4. Observing from the variance of these three algorithms, we find they all are very steady.

Table 2 shows the mean and variance of the distribution metric Δ using different algorithms. The three algorithms also have very close and uniformly distribution. NSGA-DT has the best distribution in ZDT4 and DTLZ1, but the worst distribution in ZDT6. The distribution of NSGA-II is best in ZDT2, ZDT6, DTLZ2, DTLZ2 and DTLZ4. We also observe that these algorithms are steady from the distribution's variance.

Table 3 shows the running time of three MOEAs. The running times of NSGA-DT are much smaller than that of other algorithms on all test functions; SPEA2 is the slowest one. Since the efficient density estimation algorithm and fast-nondominated-sort algorithm are used in NSGA-II [6], NSGA-II is obviously faster than SPEA2. Because of the application of dominating tree, NSGA-DT is much faster than NSGA-II.

Through the performance experiments, we can find that NSGA-DT, as the application of dominating tree, not only approximate the true Pareto optimal front, but also maintain a good diversity on all problems. The running time show that the dominating tee reduces the unnecessary comparisons, and fasten the fitness assignment indeed.

5. Conclusions

In this paper, a new data structure- dominating tree-has been introduced to as a fitness assignment approach. The dominating tree is a binary tree with dominating information among individuals. The construction algorithm indicates that the dominating tree has some useful properties, which make it useful in EMO. To show DT as an effective fitness assignment that can improve general MOEAs, we propose the rank assignment in DT, and apply it to replace the non-dominated-sort procedure of NSGA-II. The performance experiments indicate that the DT improved version of NSGA-Ⅱ (NSGA-DT) is found to be competitive with SPEA2 and NSGA-II in terms of converging to the true Pareto front and maintaining the diversity of individuals. Moreover, NSGA-DT is much faster than other two algorithms.

Table 1. Convergence comparison of different algorithms using Υ. First rows are mean, the second rows are variance.

	ZDT1	ZDT2	ZDT3	ZDT4	ZDT6	DTLZ1	DTLZ2	DTLZ3	DTLZ4
NSGA-DT	0.001800	0.000241	0.006068	6.834584	0.380954	0.062340	0.117837	0.192634	0.120307
	0.000229	0.000807	0.001702	3.070909	0.038333	0.088464	0.003187	0.082492	0.002533
NSGA-II	0.001152	0.000327	0.001454	8.260305	0.351938	0.571498	0.119575	0.173664	0.130049
	0.000115	0.000589	0.000068	5.866980	0.030285	0.508453	0.002103	0.044093	0.027837.
SPEA2	0.001984	0.001252	0.001618	8.341894	0.399221	0.253707	0.121365	0.277417	0.121587
	0.000167	0.001006	0.000091	4.454580	0.038211	0.314592	0.002137	0.130644	0.001067

Table 2. Convergence comparison of different algorithms using Δ. First rows are mean, the second rows are variance.

	ZDT1	ZDT2	ZDT3	ZDT4	ZDT6	DTLZ1	DTLZ2	DTLZ3	DTLZ4
NSGA-DT	0.002647	0.000768	0.006777	0.464903	0.012766	0.184153	0.253905	0.325358	0.248115
	0.000181	0.001172	0.000538	0.675500	0.002407	0.097828	0.008718	0.074161	0.011867
NSGA-II	0.002654	0.000532	0.006781	0.610860	0.009561	0.500472	0.251486	0.321142	0.230279.
	0.000126	0.001063	0.000488	1.617882	0.001242	0.270827	0.024443	0.051273	0.076928
SPEA2	0.001219	0.000834	0.005585	0.842312	0.010338	0.572517	0.254560	0.505960	0.255493
	0.000070	0.000679	0.000461	1.746986	0.000742	0.889044	0.009572	0.349350	0.011421

Table 3. Compare running time of different algorithms. The time unit is millisecond.

	ZDT1	ZDT2	ZDT3	ZDT4	ZDT6	DTLZ1	DTLZ2	DTLZ3	DTLZ4
NSGA-DT	1420	934	1328	867	654	2387	2923	2329	3098
NSGA-II	1885	2234	1839	2223	1910	3343	4045	3310	4185
SPEA2	8657	8756	8593	8614	8368	20468	23417	20384	23715

Acknowledgement

This work is supported by the National Science Foundation of China (No. 60435010, 90604017, 60675010), 863 National High-Tech Program (No.2006AA01Z128), National Basic Research Priorities Programme (No. 2003CB317004) and the Nature Science Foundation of Beijing (No. 4052025). It is also supported by the National Natural Science Foundation of China under Grant 60402011.

References

[1] C.A.C.Coello, "Evolutionary multiobjective optimization: a historical view of the field," *IEEE Computational Intelligence Magazine*, vol. 1, no. 1, pp. 28-36, 2006.
[2] K. Deb, A. Pratab, S. Agarwal, and T.MeyArivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol.6, pp. 182-197, Apr.2002.
[3] E.Zitzler, M.Laumanns and L.Thiele, "SPEA2: improving the strength Pareto evolutionary algorithm," TIK-Report 103, 2001, ETH Zentrum, Gloriastrasse 35, CH-8092 Zurich, Switzerland.
[4] G. Yen and H. Lu, "Dynamic multiobjective evolutionary algorithm: adaptive cell-based rank and density estimation," *IEEE Trans. Evol. Comput.*, vol.7, no3, pp. 253-274, 2003.
[5] M.T.Jensen, "Reducing the Run-Time complexity of Multiobjective EAs: The NSGA-II and Other Algorithms," *IEEE Trans. Evol. Comput.*, vol.7, no.5, pp.503-515, 2003.
[6] K.Deb, A.Samir, etc, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II," KanGAL Report No.200001, Kanpur, PIN 208 016, India.