A New Data Structure for the Nondominance Problem in Multi-Objective Optimization

Oliver Schütze

Department of Mathematics and Computer Science
http://math-www.uni-paderborn.de/~agdellnitz
University of Paderborn
Paderborn, Germany

Abstract. We propose a new data structure for the efficient computation of the nondominance problem which occurs in most multi-objective optimization algorithms. The strength of our data structure is illustrated by a comparison both to the linear list approach and the quad tree approach on a category of problems. The computational results indicate that our method is particularly advantageous in the case where the proportion of the nondominated vectors versus the total set of criterion vectors is not too large.

1 Introduction and Background

In most computational algorithms for the solution of a multi-criteria optimization problem[1]

$$\min F : Q \subset \mathbb{R}^n \to \mathbb{R}^k \tag{1.1}$$

the problem arises to sort out the *nondominated* vectors from a given finite (but large) set of criterion vectors $P \subset \mathbb{R}^k$. A vector $v \in \mathbb{R}^k$ is dominated by a vector $w \in \mathbb{R}^k$ if $w_i \leq v_i$ for all $i \in \{1, \ldots, k\}$ and $v \neq w$ (i.e. there exists a $j \in \{1, .., k\}$ such that $w_j < v_j$). A vector v is called nondominated in P if there is no vector $p \in P$ which dominates v.
The *nondominance problem* can be divided into two main classes. First, there is the *static* nondominance problem. Here one has to find the subset N of nondominated vectors of a given set P at once. For details we refer e.g. to [5] and [8], where the problem is solved up to $k = 4$.
Second, there is the *dynamic* nondominance problem which occurs in most multi-objective optimization techniques and which we want to address in this paper. We are given a set of nondominated vectors P, and, in addition to this set, there is a sequence of candidates (which is generated by the optimization procedure). For every vector v of this sequence the *archive* P has to be updated (see Figure 1).

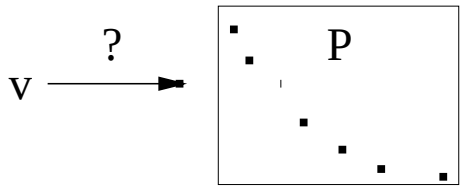


Fig. 1. Scheme of the dynamic nondominance problem: a given archive P of nondominated points has to be uptdated by arriving data.

There are several alternative approaches for the solution of this problem. First, if all the candidates (and hence all elements of the archive P) lie in bounded (hyper-) rectangles, it is suitable to use *kd-trees* ([1, 10]) or *range trees* ([2, 10]). *Priority trees* ([11]) are suitable for the case where these

[1] In this paper we assume that all objectives have to be minimized.

rectangles are unbounded on a single side. If there are no restrictions to the range of the objectives the intuitive *linear list* approach (see e.g. [14]) can be used. Another way of attacking the problem is proposed in [9], where a clever usage of the data structure *quad tree* (see [7]) is utilized. These techniques were refined in [14] and [13]. We refer to [12] for the extensions of the quad tree approach to multi-objective evolutionary algorithms. Furthermore, there exists the *composite point* approach which is presented in [4] and finally discussed in [6]. The data structure we are proposing here is - like all the methods mentioned above except the linear list approach - tree-based. A slightly different use of this approach due to a different kind of problem is presented in [3].

2 Attacking the Nondominance Problem

Let us assume that we have a dynamic nondominance problem, i.e. a sequence of candidates $v^j \in \mathbb{R}^k$ for which a given archive $P \subset \mathbb{R}^k$ has to be updated.
The basis for our approach is to store the nondominated vectors from P in the following tree:

DEFINITION A k-ary tree T is called a *dominance decision tree*, if for every node $p = (p_1, \ldots, p_k) \in T$ and for each existing i-th son $s = (s_1, \ldots, s_k)$ from p the following holds:

$$s_j \leq p_j \quad \forall j = 1, .., i-1 \tag{2.1}$$

$$s_i > p_i \tag{2.2}$$

A simple example of a dominance decision tree for three objectives is shown in Figure 2.

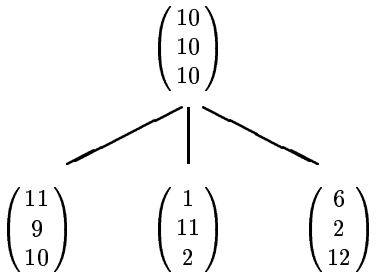


Fig. 2. Example of a dominance decision tree for $k = 3$.

For a given archive P and a new candidate $v \in \mathbb{R}^k$ the following steps have to be performed:

1.) If there exist one vector $D \in P$ which dominates v, then STOP, else go to step 2.
2.) Detect and delete all elements $d \in P$ which are dominated by v.
3.) Insert v into the archive P.

In the following we describe how to realize these steps and how to take advantage of the structure of the dominance decision tree.
ad 1.) Assume that a vector v and an archive P – stored in a dominance decision tree with root r – are given. First, v has to be compared to the root r. If r dominates v, we stop and v has to be discarded. If v and r are mutually non-dominating, then the algorithm has to make the comparisons recursively in some subtrees of r. Due to (2.2) this comparison has to be made only in the i-th subtrees of r where $r_i \leq v_i$. The algorithm `DetectDomination` reads as follows:

Algorithm DetectDomination
Input: root r, vector $v \in \mathbb{R}^k$.
Task: returns 1, if there exists a vector $p \in P$ (given by root r) which dominates v, else 0.

```
DetectDomination (root r,node v)
    if r dominates v
        return 1
    for i = 1,...,k
        if r_i <= v_i AND the i-th son of r exists (denote it by r -> son_i)
            if DetectDomination(r -> son_i, v) == 1
                return 1
    return 0
```

ad 2.) Assume again that a vector v and a dominance decision tree P are given. First we have to discuss which nodes have to be checked for domination, i.e. in which subtrees of P the algorithm has to look for dominated points. With given $p \in P$ it follows by conditions (2.1) and (2.2) that the search has to be continued in the first i subtrees of k where $i \in \{1, \ldots, k\}$ is the smallest index where $v_i > p_i$. In order to see this let s be a vector from the l-th subtree where $i < l <= k$. Then by construction of the dominance decision tree:

$$s_i \overset{(2.1)}{\leq} p_i < v_i,$$

and hence s cannot dominate v.
We illustrate this by an example: let p, v_1, v_2 and v_3 be given by

$$p = \begin{pmatrix} 10 \\ 10 \\ 10 \end{pmatrix}, \; v_1 = \begin{pmatrix} 12 \\ 8 \\ 5 \end{pmatrix}, \; v_2 = \begin{pmatrix} 2 \\ 12 \\ 1 \end{pmatrix}, \; v_3 = \begin{pmatrix} 3 \\ 3 \\ 3 \end{pmatrix}.$$

In case of $v = v_1$ the search has only to be continued in the first subtree of v_1, whereas with the choice of $v = v_2$ the algorithm has to search in the first two subtrees of v_2. Eventually in case of $v = v_3$ the data structure has no advantage because all three subtrees have to be scanned.
A deletion of a node $p \in P$ can be done as follows: if one son s of p does exist, then it can be moved to the position of p. The other nodes of the subtree of p have to be reinserted – into the lifted subtree with root s. The deletion of the dominated nodes can be done via one postorder run through the tree:

Algorithm DeleteDominated
Input: root r, vector $v \in \mathbb{R}^k$.
Task: deletes every vector $p \in P$ which is dominated by v.

```
DeleteDominated (root r, vector v)
    for i = 1,...,k
        if the i-th son of r exists (denote it by r -> son_i)
            DeleteDominated (r -> son_i, v)
        if v_i > r_i
            break
    if v dominates r
        if r is leave
            delete r and STOP
        j := arg min { the j-th son of r exists (denote it by r -> son_j) }
        Move r -> son_j to the position of r
        for l = j+1,...,k
            if the l-th son of r exists (denote it by r -> son_l)
                TreeInsert (r -> son_l)
        delete r
```

The algorithm `TreeInsert` used above reads as follows:

Algorithm TreeInsert
Input: root r, root s.
Task: inserts every vector of the tree given by root s into the tree given by root r.

```
TreeInsert (root r, root s)
    for i = 1,...,k
        if the i-th son of s exists (denote it by s → son_i)
            TreeInsert (r, s → son_i)
    Insert (r, s)
    delete s
```

ad 3.) By its definition there is only one possible way for the insertion of a vector v into a given dominance decision tree P (given by root r):

Algorithm Insert
Input: root r, vector v.
Task: inserts v into the archive P (given by root r).

```
Insert(root r, vector v)

    i := arg min {v_i > r_i}
    if the i-th son of r exists (denote it by r → son_i)
        Insert(r → son_i,v)
    else
        r → son_i := v
```

Now the main algorithm for the update of an archive P (with root r) by a candidate v can be stated. Note that the root of the dominance decision tree can be changed in the algorithm `DeleteDominated`.

Algorithm Update
Input: root r, vector v.
Task: updates the archive P (given by root r) by the candidate v.

```
Update (root r, vector v)
    if( P is empty)
        P := {v}   (r := v)
        STOP
    if DetectDomination (r, v) == 1
        STOP
    DeleteDominated (r, v)
    Insert ((root of the archive P), v);
```

The algorithm presented above has the average case complexity of $O(n^2)$ for vector comparisons. However, since there is no algorithm with a provable complexity better than $O(n^2)$, we will discuss the particular advantages of the dominance decision tree approach in the following section.

3 Computational Results

Here we make a comparison of the approaches which need no restrictions to the range of the objective values. Regrettably, due to time limitations of the proceedings, it was not able for the author to involve the recently proposed composite point approach ([6]) to the comparison, though this would be very interesting and has to be done in the near future. Hence here we compare the

linear list approach, the quad tree approach and the dominance decision tree approach.
For the comparison we proceed as in [14] and take test points generated by an annulus as criterion vectors because by this category of problems the particular advantages of the three approaches can be demonstrated.
We choose a sequence of vectors $v^j \in \mathbb{R}^k$ which have to be inserted to the archive P given by the nondominated vectors of the set $\{v^1, .., v^{j-1}\}$. The components of every vector $v^j = (v_1, .., v_k)$ are of the following form:

$$v_i := -\frac{\tilde{r}_i}{\|w\|} w_i, \tag{3.1}$$

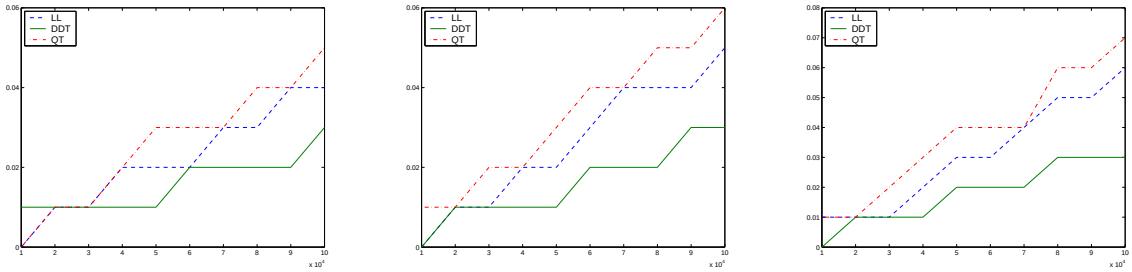
where $\tilde{r}_i \in [r, 1]$ and $w \in \mathbb{R}^k_+$ are chosen at random. This choice of criterion vectors allows to adjust not only the number k of "objectives" but also the proportion p_n of the nondominated vectors versus the total number of criterion vectors: it is easy to see that the larger the value of $r \in [0, 1)$ is the larger the value of p_n will typically be. Exactly this proportion is important for the comparison of the two tree based approaches: the computational results indicate that the dominance decision tree approach is advantageous in the case where the proportion p_n is "moderate". The larger the value of p_n the better is the performance of the quad tree approach and it gets eventually faster than the dominance decision tree approach. Of course it is barely possible to detect an exact "balance proportion" p_n^b where the two approaches have the same running time, but at least it seems to be possible to give some guidelines.
In Figure 3 and Table 1 we show that for $k = 3$ the proportion where the running time of the two approaches is basically the same is approximately $p_n^b = 1/3$ (with $r_b = 0.95$). That means that the dominance decision tree approach is faster when the optimization algorithm generates in average at most every third time a nondominated vector, otherwise the quad tree approach is faster.
For $k = 4$ the proportion p_n^b seems to be 0.5 (see Figure 4 and Table 2), but because of the higher dimension the limit radius $r_b \approx 0.85$ is lower than for k equals 3. A similar observation was made for $k = 5$ ($p_n^b \approx 0.5$ but $r_b \approx 0.7$). This means that the efficiency of the quad tree approach increases with growing k in comparison to the dominance decision tree approach.
In the case where k equals 2 we figured out that the linear list approach is faster than the dominance decision tree approach as well as the quad tree approach. This is possibly due to the overhead given by the tree based approaches.

4 Summary

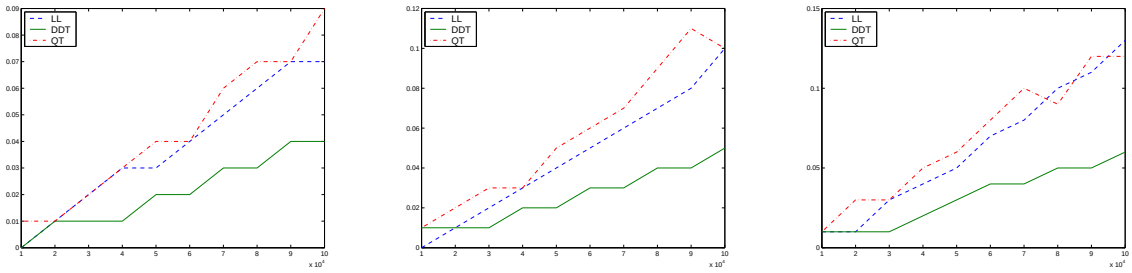
In this paper we have presented a new data structure for the computation of the dynamic nondominance problem and have shown its efficiency by a particular category of problems. Compared to the linear list approach and the quad tree approach the new method is advantageous in the case where the proportion of the number of nondominated points to the number of all criterion vectors is "moderate". In future work the data structure has to be compared to the composite point approach ([6]).

Achnowledgements

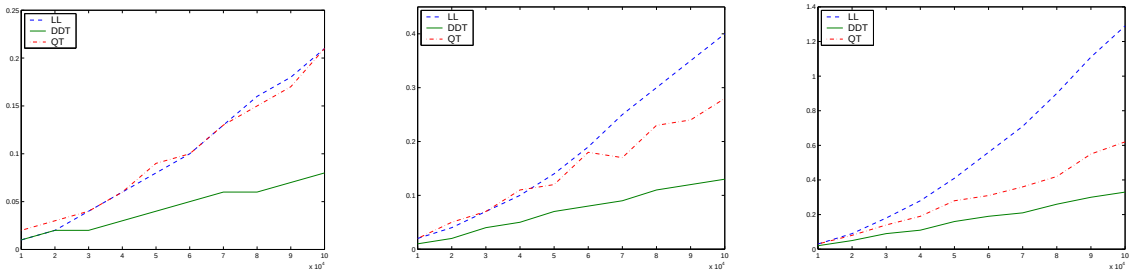
The author thanks Robert Elsässer for helpfull discussions on the contents of this paper.



(a) r=0.1 (b) r=0.2 (c) r=0.3



(d) r=0.4 (e) r=0.5 (f) r=0.6



(g) r=0.7 (h) r=0.8 (i) r=0.9

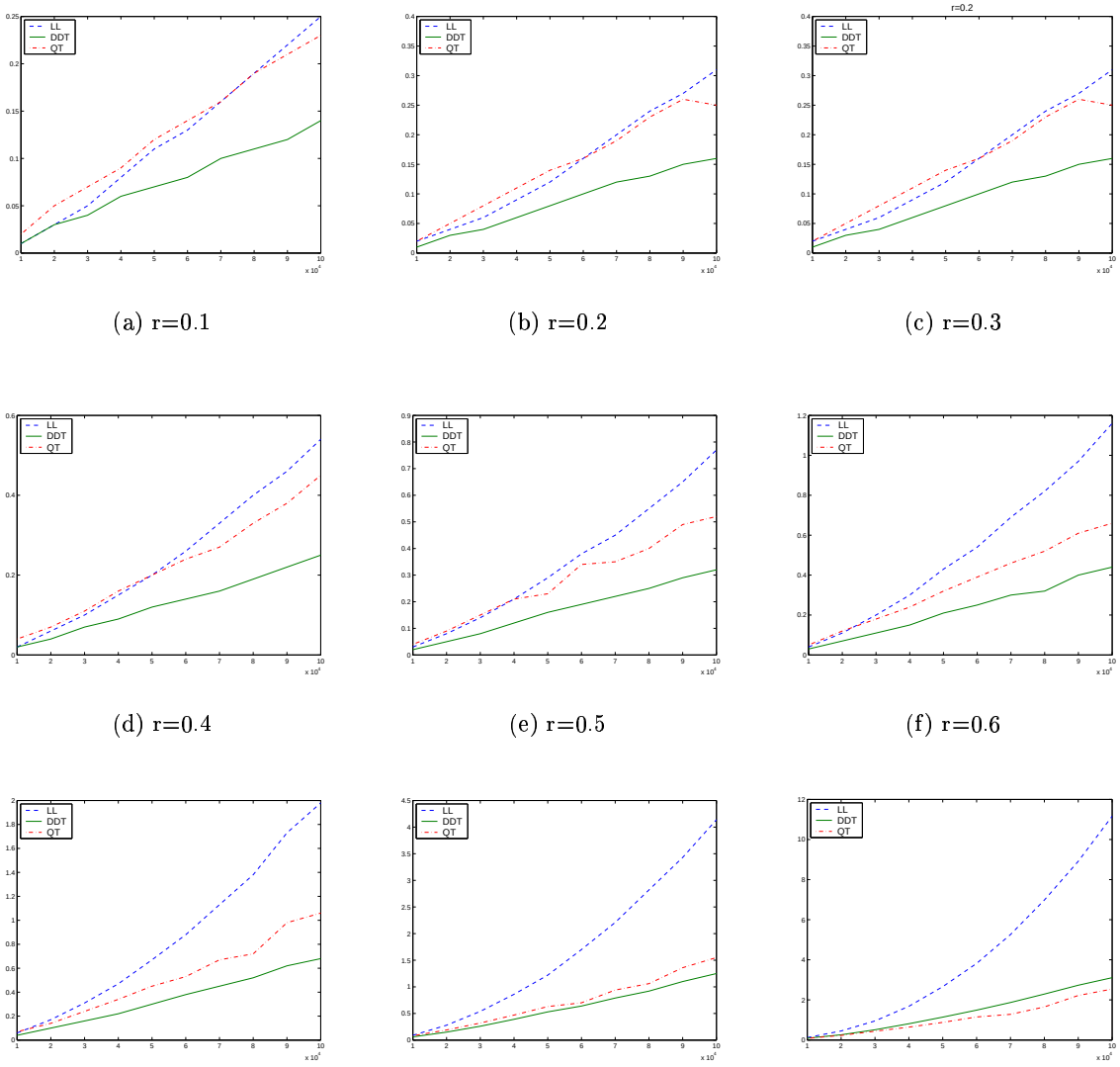
Fig. 3. Annulus generated test problem results for $k = 3$. In the Figures the number N of criterion points versus the running time of the three approaches is plotted for different values of the radius r. Here we have chosen $N = \{1000, 2000, \ldots, 10000\}$. For details see Table 1.

	N	2000	4000	6000	8000	10000
r = 0.3	T_{LL}	0.01	0.02	0.03	0.05	0.06
	T_{QT}	0.01	0.03	0.04	0.06	0.07
	T_{DDT}	0.01	0.01	0.02	0.03	0.03
	p_n	0.07	0.04	0.04	0.03	0.03
r = 0.5	T_{LL}	0.01	0.03	0.05	0.07	0.10
	T_{QT}	0.02	0.03	0.06	0.09	0.10
	T_{DDT}	0.01	0.02	0.03	0.04	0.05
	p_n	0.10	0.07	0.06	0.05	0.04
r = 0.7	T_{LL}	0.02	0.06	0.10	0.16	0.21
	T_{QT}	0.03	0.06	0.10	0.15	0.21
	T_{DDT}	0.02	0.03	0.05	0.06	0.08
	p_n	0.15	0.11	0.09	0.07	0.07
r = 0.9	T_{LL}	0.09	0.28	0.56	0.90	1.29
	T_{QT}	0.08	0.19	0.31	0.42	0.62
	T_{DDT}	0.05	0.11	0.19	0.26	0.33
	p_n	0.37	0.28	0.23	0.20	0.18
r_b = 0.95	T_{LL}	0.29	0.85	1.66	2.66	3.91
	T_{QT}	0.13	0.41	0.69	1.05	1.33
	T_{DDT}	0.15	0.40	0.65	0.95	1.31
	p_n	0.61	0.47	0.41	0.36	0.33

Table 1. Annulus generated test problem results for $k = 3$

	N	2000	4000	6000	8000	10000
r = 0.3	T_{LL}	0.05	0.11	0.21	0.30	0.39
	T_{QT}	0.06	0.12	0.19	0.28	0.38
	T_{DDT}	0.03	0.07	0.11	0.16	0.20
	p_n	0.18	0.14	0.12	0.10	0.10
r = 0.5	T_{LL}	0.08	0.21	0.38	0.55	0.77
	T_{QT}	0.09	0.21	0.34	0.40	0.52
	T_{DDT}	0.05	0.12	0.19	0.25	0.32
	p_n	0.26	0.20	0.17	0.15	0.14
r = 0.7	T_{LL}	0.17	0.47	0.88	1.38	1.98
	T_{QT}	0.14	0.34	0.53	0.72	1.06
	T_{DDT}	0.10	0.22	0.38	0.52	0.68
	p_n	0.42	0.32	0.28	0.25	0.22
r_b = 0.85	T_{LL}	0.20	0.71	1.80	3.36	5.25
	T_{QT}	0.13	0.29	0.56	0.76	1.20
	T_{DDT}	0.11	0.30	0.51	0.74	1.05
	p_n	0.68	0.56	0.50	0.46	0.43
r = 0.9	T_{LL}	0.45	1.68	3.82	6.98	11.16
	T_{QT}	0.23	0.64	1.15	1.65	2.54
	T_{DDT}	0.26	0.81	1.49	2.29	3.11
	p_n	0.85	0.77	0.71	0.66	0.63

Table 2. Annulus generated test problem results for $k = 4$.



(a) r=0.1 (b) r=0.2 (c) r=0.3

(d) r=0.4 (e) r=0.5 (f) r=0.6

(g) r=0.7 (h) r=0.8 (i) r=0.9

Fig. 4. Annulus generated test problem results for $k = 4$. For details see Figure 3 and Table 2.

References

1. J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, pages 509–517, 1975.
2. J. L. Bentley and J. H. Friedmann. Data structures for range searching. *Computing Surveys*, 4:398–409, 1979.
3. M. Dellnitz, R. Elsässer, T. Hestermeyer, O. Schütze, and S. Sertl. Covering Pareto sets with multilevel subdivision techniques. to appear, 2002.
4. R. M. Everson, J.E. Fieldsend, and S. Singh. Full elite sets for multi-objective optimization. In I. C. Parmee, editor, *Adaptive Computing in Design and Manufacture V*, Springer, 2002.
5. F. P. Preparata and M. I. Shamos. *Computational Geometry - An Introduction.* Springer Verlag, 1988.
6. J. Fieldsend, R.M. Everson, and S. Singh. Using unconstrained elite archives for multi-objective optimisation. to appear, 2003.
7. R. A. Finkel and J. L. Bentley. Quad trees, a datastructure for retrieval on composite keys. *Acta Informatica*, 4:1–9, 1974.
8. P. Gupta, R. Janardan, M. Smid, and B. Dasgupta. The rectangle enclosure and point-dominance problems revisited. *Int. J. Comput. Geom. Appl.*, 5:437–455, 1997.
9. W. Habenicht. Quad trees, a datatructure for discrete vector optimization problems. *Lecture Notes in Economics and Mathematical Systems*, 209:136–145, 1983.
10. M. de Berg and M. van Kreveld and M. Overmars and O. Scharzkopf. *Computational Geometry: algorithms and applications.* Springer Verlag, 1997.
11. E. M. McCreight. Priority search trees. *SIAM Journal on Computing*, 14:257–276, 1985.
12. S. Mostaghim, J. Teich, and A. Tyagi. Comparison of data structures for storing pareto-sets in moeas. *Int. J. Comput. Geom. Appl.*, 5:437–455, 2002.
13. M. Sun and R. E. Steuer. InterQuad: An interactive quad tree based procedure for solving the discrete alternative multiple criteria problem. *European Journal of Operational Research*, 89:462–472, 1996.
14. M. Sun and R. E. Steuer. Quad trees and linear lists for identifying nondominated criterion vectors. *INFORMS J. Comp.*, 8:367–375, 1996.