Heuristic search strategies for multiobjective state space search

PALLAB DASGUPTA, P P CHAKRABARTI and S C DESARKAR

Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur 721302, India

email addresses: [pallab,ppchak,scd]@cse.iitkgp.ernet.in

Abstract. The multiobjective search model is a framework for solving multicriteria optimization problems using heuristic search techniques, where the different dimensions of a multiobjective search problem are mapped into a vector valued cost structure and partial order search is employed to determine the set of non-inferior solutions. This new framework for solving multicriteria optimization problems has been introduced by Stewart and White, who presented a generalization of the well known algorithm A^* in this model. This paper presents several results on multiobjective state space search which helps in refining the scheme proposed by them. In particular, the following results have been presented.

- The concept of *pathmax* has been generalized to the multiobjective framework. It has been established that unlike in the conventional model, multidimensional *pathmax* (in the multiobjective model) is useful for non-pathological tree search instances as well.
- We investigate the utility of an induced total order on the partial order search mechanism. The results presented are as follows:
 - If an induced total order is used in the selection process, then in general it is not necessary to compute the entire set of heuristic vectors at a node.
 - In memory-bounded search, a multiobjective search strategy that uses an induced total order for selection can back up a single cost vector while backtracking and yet guarantee admissibility though multiple non-inferior candidate back-up vectors may be present in the space pruned while backtracking.
- In this paper we study multiobjective state space search using inadmissible heuristics. We show that if heuristics are allowed to overestimate, then no algorithm is guaranteed to find all non-inferior solutions unless it expands dominated nodes also.

The paper also addresses the task of multiobjective search under memory bounds, which is important in order to make the search scheme viable for

practical purposes. The paper presents a linear space multiobjective search strategy called MOR*0 and suggests several variants of the strategy which may be useful under different situations.

Keywords. Heuristic search; multicriteria optimization.

1. Introduction

Many real world optimization problems have multiple, conflicting and non-commensurate objectives. The task of adequately modeling such problems in a search framework that is designed for optimizing single scalar functions is by no means easy, and has been the subject of considerable debate in the past (Keeney & Raiffa 1976). One popular approach of solving such problems is to cast them into the conventional search framework after combining the multiple criteria into a single scalar criterion. However, in most multiobjective problems, the semantics of the desired solution is context dependent and can be dictated by individual preferences. Therefore, the task of constructing the combined evaluation function so as to preserve the semantics of the desired solution is difficult, and may require sufficient experience about solving that problem.

The other popular approach of solving multiobjective problems is to optimize one criterion at a time under given constraints on the others. This approach automatically preserves the semantics of the problem since it allows the multiple dimensions to retain their individual identities. However, one difficulty lies in determining a set of good constraints, in the absence of which search becomes unduly expensive. Moreover, repeatedly searching the same state space by progressively refining the constraints (until a satisfactory solution is found) increases the search complexity enormously.

The multiobjective search model was introduced by Stewart and White (Stewart 1988; Stewart & White 1991) as a unified framework for solving search problems involving multiple objectives. Since multiple non-commensurate criteria are involved, the solution space is partially ordered and will, in general, contain several *non-inferior* solutions. Multiobjective search addresses the task of determining the set of such solutions in the search space. Once the set of non-inferior solutions are found, standard procedures may be applied to choose the desired solution (Bogetoft 1986; Kok 1986; Korhonen 1986; Korhonen *et al* 1986; Mond & Rosinger 1985; Vansnick 1986).

In the multiobjective framework, the costs are modelled by vectors, such that each dimension of the cost vector represents a distinct non-commensurate optimization criterion. The following partial order is used to identify the non-inferior options.

DEFINITION 1

Dominance: Let $\mathbf{y}_1$ and $\mathbf{y}_2$ be two K -dimensional vectors. Then $\mathbf{y}_1$ dominates $\mathbf{y}_2$ iff:

1. $\mathbf{y}_1 \leq \mathbf{y}_2$ in all the K dimensions, and
2. $\mathbf{y}_1 < \mathbf{y}_2$ in at least one of the K dimensions.

A vector $\mathbf{y}_i$ is said to be “non-dominated” in a set of vectors Y if there does not exist another vector $\mathbf{y}_j \in Y$ such that $\mathbf{y}_j$ dominates $\mathbf{y}_i$. □

A multiobjective search strategy uses the above partial order to eliminate all clearly inferior alternatives and direct the search towards the set of non-dominated solutions. The multiobjective heuristic search problem is as follows:

DEFINITION 2

Multiobjective search problem:

Given:

1. A search space, represented as a locally finite directed graph.
2. A vector valued cost structure, with each dimension representing a distinct optimization criterion.
3. A heuristic evaluation function that returns a set of non-dominated vector valued costs for each candidate search avenue. Each vector is an estimate of the potential non-dominated solutions which may be obtained along that search avenue.

Find:

The set of non-dominated solutions in the search space. □

It should be noted that in the multiobjective search space a node may have multiple heuristic vectors. This is due to the fact that a node may lie on the path to multiple mutually non-dominated solutions. For every solution path that contains the given node, the multiobjective heuristic function may compute a vector valued estimate of the corresponding solution cost. It should be noted that some of these vectors computed at the node may not be distinct since the same vector may estimate the cost vectors of more than one solution. However if a single vector were to estimate the cost vectors of all the solutions then the heuristic value would become overly restrictive. Therefore, a typical multiobjective heuristic function computes a *set* of non-dominated heuristic vectors at every node.

In their work, Stewart and White (Stewart 1988; Stewart & White 1991) presented an algorithm *MOA** which is a generalization of the well known *A** algorithm (Nilsson 1980) to the multiobjective search framework. Like *A**, the algorithm maintains a list (called OPEN) of tip nodes of the different search paths. A node in OPEN is considered to be a candidate for expansion if one or more cost vectors of the node is non-dominated by other nodes in OPEN and by the solution vectors found so far. The algorithm terminates when there are no more non-dominated nodes in OPEN. Stewart and White have shown that when the heuristic function is admissible, the algorithm *MOA** terminates with the entire set of non-dominated solutions. Several well known results from the conventional search model have been extended to the multiobjective framework. For example, it has been shown that if *A* and *A'* are two versions of *MOA** such that *A* is at least as informed as *A'* then *A* expands no more nodes than *A'*. They have also established that monotonicity of the multiobjective heuristic function implies admissibility.

This paper addresses two broad topics. The first is to study multiobjective state space search under two different types of heuristics, namely non-monotone and inadmissible heuristics. The other is to develop search strategies for the multiobjective framework that operate under space constraints. Most of the analyses in this paper assumes that the state space is an implicitly specified tree.

In this paper we generalize the concept of *pathmax* to the multiobjective framework and show that in this model the utility of *pathmax* extends over a wider class of problem instances than in the conventional search model. For the problem of searching with inadmissible heuristics we show that in general only brute force search strategies are guaranteed to find all non-dominated solutions. We also analyze specific conditions under which a best-first search strategy such as *MOA** is admissible.

One of the major contributions of the present work has been to investigate the utility of using an induced total order to guide the selection process. In this paper we establish several results which show that the policy of using an induced total order called *K-order* is useful in many situations. The concepts of *pathmax* and *K-ordering* lead to an improved version of the algorithm *MOA**. The new algorithm is called *MOA***.

The problem of multiobjective heuristic search in restricted memory has been considered in this paper. We show that the main difficulty in extending known memory bounded search techniques from the conventional model arises from the existence of multiple candidate back-up cost vectors in the pruned space. We then establish that if an induced total order such as *K-order* is used to guide the partial order search then it is possible to construct a strategy that backs up a single cost vector while backtracking and yet guarantees admissibility. A linear space search strategy called *MOR*0* is presented on these lines. Several variants of the strategy are suggested to cater to specific situations.

The paper is organized as follows. Section 3 presents the generalization of *pathmax* to the multiobjective search scheme and establishes the related results. The policy of using an induced total order to guide partial order heuristic search is considered in § 4. In § 5 we consider the utility of incorporating *K-ordering* in the algorithm *MOA**. Section 6 addresses the problem of multiobjective state space search in bounded memory. The algorithm *MOR*0* is presented in the same section. In § 7 we analyze the problem of multiobjective search using inadmissible heuristics and present related results.

2. Preliminary notations and definitions

Most of the notations used in this paper are standard notations adopted in heuristic search literature. We therefore highlight only the characteristic terminologies of multiobjective heuristic search.

- $+$: If $\mathbf{a}$ and $\mathbf{b}$ are vectors (of equal dimension) then $\mathbf{a} + \mathbf{b}$ denotes the vector formed by summing $\mathbf{a}$ and $\mathbf{b}$ in each individual dimension.
- $-$: If $\mathbf{a}$ and $\mathbf{b}$ are vectors (of equal dimension) then $\mathbf{a} - \mathbf{b}$ denotes the vector formed by subtracting $\mathbf{b}$ from $\mathbf{a}$ in each individual dimension.
- $vmax(\mathbf{a}, \mathbf{b})$: If $\mathbf{a}$ and $\mathbf{b}$ are vectors (of equal dimension) then $vmax(\mathbf{a}, \mathbf{b})$ denotes a vector which is equal to the maximum of $\mathbf{a}$ and $\mathbf{b}$ in each individual dimension.
- $H(n)$: The non-dominated set of heuristic vectors computed at node n .
- $G(n)$: $G(n)$ is the vector valued cost of the path from the start node s to node n in the search tree.
- $F(n)$: The non-dominated set of cost vectors of node n computed as $h(n) + g(n)$, where $h(n) \in H(n)$ and $g(n) \in G(n)$. In the case of trees $g(n)$ is the same as $G(n)$.

The definition of admissible and non-monotonic heuristics are as follows.

DEFINITION 3

Admissible heuristics: A multiobjective heuristic function is said to be admissible if the set $H(n)$ of heuristic vectors computed at each node n satisfies the following property:

Admissibility property: For every solution path $P(s, \gamma)$ through node n (where s is the start node and γ is any goal node), there exists a heuristic vector $h(n)$ in $H(n)$ such that $h(n)$ either dominates the cost vector of the solution path from n to γ or is equal to it.

It should be noted that in one extreme case $H(n)$ may contain only a single vector which satisfies the above property, while in the other extreme case $H(n)$ may contain a distinct vector corresponding to each solution path through node n . $\square$

DEFINITION 4

Monotone heuristics: A multiobjective heuristic function is said to be monotone (or consistent) if for all nodes n_i and n_j , such that n_j is a successor of n_i , the set $H(n_i)$ of heuristic vectors computed at node n_i satisfies the following property:

Monotonicity property: For each heuristic vector $h(n_j)$ in $H(n_j)$, there exists a heuristic $h(n_i)$ in $H(n_i)$, such that $h(n_i)$ either dominates the vector $c(n_i, n_j) + h(n_j)$ or is equal to it. $c(n_i, n_j)$ denotes the cost vector of the edge from n_i to n_j . $\square$

3. Multidimensional pathmax

Pathmax is a standard concept in heuristic search which is used to strengthen admissible non-monotonic heuristics during search. The basic idea is to use the cumulative heuristic information available along the search path to compute the heuristic cost of a node, so that the heuristic cost of the node is consistent with the heuristic cost of its ancestors.

The idea of using *pathmax* to propagate the cost of the parent node to the child node was suggested by Mero (1984). *Pathmax* has been found to be useful on the following accounts.

- Dechter & Pearl (1985) and Gelparin (1977) have shown that by using *pathmax* it is possible to develop A^* -like algorithms that are superior to A^* in terms of the set of nodes expanded. However they have also shown that such extensions of A^* may expand a smaller set of nodes only in *pathological* problem instances, that is, in problem instances where *every* solution path contains at least one fully informed non-goal node. It has been shown (Dechter & Pearl 1985) that in non-pathological cases the same set of nodes will be expanded irrespective of using *pathmax*.
- Martelli (1977), and Bagchi & Mahanti (1983) have shown that it may be possible to reduce the number of nodes re-expanded by A^* in graph search problems considerably if a strategy similar to *pathmax* is used. Chakrabarti *et al* (1989) have also used this result in memory bounded search.

Therefore in the conventional search model, the utility of *pathmax* in *tree* search is limited to *pathological* problem instances only. In the case of graphs *pathmax* may be used to

reduce the number of node re-expansions in non-pathological cases as well. However the set of nodes expanded remain the same irrespective of using *pathmax*.

In this section, we generalize the concept of *pathmax* to the multiobjective search model. We shall show that in the multiobjective search framework *pathmax* may reduce the set of nodes expanded even in non-pathological cases.

3.1 The definition of *pathmax*

When the successor m of node n is generated, each heuristic vector $h(m)$ evaluated by the heuristic function at node m is updated using *pathmax* as follows:

For each vector $h(n)$ in $H(n)$

For each vector $h(m)$ evaluated at node m

1. Create a new vector $h'(m)$ as follows:

$$h'(m) \leftarrow v\max\{h(m), h(n) - c(n, m)\}$$

where $c(n, m)$ denotes the cost vector of the edge (n, m) , and the function $v\max$ is as defined in § 2.

2. Put $h'(m)$ in $H(m)$ and remove dominated vectors from $H(m)$, if any.

Thus *pathmax* is used in the same way as in the single objective search model, but for every heuristic vector and along each dimension of the vector.

3.2 Two basic properties of *pathmax*

In order to establish the validity of multidimensional *pathmax* in the multiobjective framework we prove the following two basic properties. The proofs consider trees only.

Theorem 1. *The set of heuristics remain admissible if *pathmax* is used.*

Proof. We first prove that if the set of heuristics at a node n is admissible then the set of admissible heuristics at each immediate successor node m remains admissible when *pathmax* is used.

If the set of heuristics at node n is admissible, then corresponding to every solution path to a goal node γ through node n , there exists $h(n)$ in $H(n)$ such that $h(n)$ either dominates $h^*(n, \gamma)$ or is equal to it, where $h^*(n, \gamma)$ denotes the cost vector of the path from n to γ . If the same solution path contains node m then $h(n) - c(n, m)$ either dominates or is equal to $h^*(m, \gamma)$. Now since $h(m)$ is admissible, it follows that $h(m)$ also dominates $h^*(m, \gamma)$ or is equal to it. Therefore, the heuristic vector $h'(m)$ computed using *pathmax* as follows will also dominate $h^*(m, \gamma)$:

$$h'(m) \leftarrow v\max\{h(m), h(n) - c(n, m)\}.$$

Thus $H(m)$ either contains $h'(m)$ or some vector which dominates $h'(m)$. It follows that the set of heuristics $H(m)$ computed using *pathmax* is admissible.

We have shown that the set of heuristics at a node remain admissible provided the set of heuristics at its parent node is admissible. Since the set of heuristics at the start node remain the same, the result follows by induction. $\square$

Theorem 2. *Every node expanded by any admissible algorithm A using $pathmax$ is also expanded by A without $pathmax$ in the worst case.*

Proof. Each heuristic vector computed at a node without $pathmax$ either equals or dominates some heuristic vector computed using $pathmax$. Therefore if a node and its ancestors have one or more cost vectors that are non-dominated when computed using $pathmax$, then the node and its ancestors are non-dominated if $pathmax$ is not used. Since any admissible algorithm A will have to explore all non-dominated cost paths in the worst case, the result follows. It may be noted however that for certain individual instances anomalies may occur due to tie resolutions where the worst case set of nodes is not expanded. $\square$

3.3 The significance of $pathmax$

We now establish the result that in the multiobjective search framework $pathmax$ may be used to reduce the set of nodes expanded in non-pathological problem instances as well.

Theorem 3. *There exists non-pathological problem instances (that is, problem instances in which not all solution paths contain a fully informed non-goal node) where an arbitrarily large number of extra nodes will be expanded unless $pathmax$ is used.*

Proof. Consider the tree in figure 1. The cost vector of each edge is shown beside the edge. The heuristic vectors computed without $pathmax$ are shown beside the nodes. For simplicity, we consider only 2-dimensional costs and only one heuristic vector per node. It should be noted that no non-goal node along the solution path to node n_4 is totally informed (in either dimension).

When n_1 is expanded, the heuristic vector $(1,13)$ is computed at node n_3 and $c(n_1, n_3)$ is found to be $(1,1)$. From this $F(n_3)$ is computed as $(2,14)$. Likewise $F(n_2)$ is computed as $(2,10)$. Since $(2,10)$ dominates $(2,14)$, n_2 will be expanded earlier than n_3 to generate n_4 . Then $F(n_4)$ is computed as $(3,12)$. Now either n_4 or n_3 may be selected. Irrespective of which is selected earlier, the expansion of n_3 is certain. If $H(n_5)$ is computed without $pathmax$, then $F(n_5) = (4,9)$, and the expansion of n_5 is also certain since $(4,9)$ is non-dominated by the cost $(3,12)$ of node n_4 . On the other hand, using $pathmax$ $H(n_5)$ is computed as $(2,12)$. Therefore $F(n_5)$ becomes $(4,14)$ which is dominated by the cost $(3,12)$ of n_4 . Thus n_4 will be selected earlier and the solution of cost $(3,12)$ will be obtained. It follows that n_5 will never be expanded.

It is easy to see that the expansion of node n_5 may lead to the expansion of an arbitrarily large number of nodes in the subtree rooted at node n_5 , each of which contains some heuristic vector non-dominated by $(3,12)$ without $pathmax$. The result follows. $\square$

4. An induced total ordering: K-ordering

Efficiency of problem solving in the multiobjective search framework is largely dependent on effective partial order search strategies for finding the set of non-dominated solutions. A characteristic feature of partial order search is the presence of multiple non-inferior search

Figure 1. Multiobjective tree illustrating the advantage of *pathmax*.

paths each of which must be explored in order to guarantee admissibility. The set of nodes in OPEN that have one or more non-dominated cost vectors represent these non-inferior search paths and therefore must be expanded by admissible search strategies like MOA*. We formally define non-dominated nodes in OPEN as follows:

DEFINITION 5

Non-dominated node: A node n in OPEN is said to be non-dominated if at least one of the cost vectors in $F(n)$ is non-dominated by all cost vectors of every other node in OPEN and the cost vector of all solution nodes found so far.

It may be easily shown that the order in which the non-dominated nodes in OPEN are selected does not affect the total number of nodes expanded by a strategy like MOA* in the worst case. We shall present a proof of this result later in this paper. In this section we introduce the policy of using an induced total order to select a non-dominated node from OPEN. While this policy does not affect the number of nodes expanded by strategies such as MOA*, we shall show that there are several other benefits of using the policy.

A basic advantage of using an induced total ordering is that we then have a mechanism of identifying a non-dominated node from OPEN without having to identify the entire set of non-dominated nodes in OPEN. This is easily possible if the induced total order is defined in the following manner.

DEFINITION 6

K-ordering: Let y^1 and y^2 be two K -dimensional vectors. Then, based on *K-ordering*, $y^1 > y^2$ if

$$\exists j, 1 \leq j \leq K \text{ such that } y_j^1 > y_j^2 \text{ and } \forall i, i < j, y_i^1 = y_i^2$$

It essentially represents the policy of comparing two vector-valued costs on the basis of the first component; breaking ties on the basis of the second component; breaking further ties on the basis of the third component, and so on. The other relational operators (such as $=$ and $<$) are defined in a likewise manner.

The basic idea is derived from the single objective search strategies where the data structure OPEN is ordered on the basis of the costs of the nodes. It is easy to see that the

minimum cost vector in K -order is always non-dominated in a set of cost vectors. Since a node may have several cost vectors (in $F(n)$), we define the *representative cost vector* of a node based on the induced total order as follows.

DEFINITION 7

The representative cost vector of a node: The representative cost vector of node n is the minimum cost vector using K -ordering on the set of cost vectors $F(n)$ which is non-dominated by the cost vector of every solution found earlier.

If the representative cost vector of a node becomes dominated by the cost vector of some solution found during the course of the search, then the next non-dominated cost (in K -order) becomes the current representative cost of the node. The representative cost vector of a path in the state space is the same as the representative cost vector of its tip node.

In this paper, we shall show that using an induced total ordering such as K -ordering to guide the partial order search may be useful in several situations. The main results are as follows:

- In § 5, we show that if K -ordering is used to guide the search, then in some cases it may not be necessary to evaluate all the heuristic vectors at a node. This result is particularly useful in problems where generating the heuristics are costly.
- In § 5, we also observe that in two-objective problems (which are quite numerous) we can simplify the process of dominance checking by using K -ordering to guide the search.
- In § 6, we discuss the advantage of using K -ordering in memory bounded search. In particular, we show that though the space pruned by a memory bounded search strategy (while backtracking) may contain a large number of nodes having one or more non-dominated cost vectors, it is possible to back up a single cost vector from the pruned space and yet guarantee admissibility by using K -ordering to guide the search.

5. The algorithm MOA**

By incorporating the ideas of K -ordering and *pathmax*, it is possible to design an improved version of the algorithm MOA^* of Stewart & White (1991). The new algorithm follows directly from MOA^* with the exceptions that the heuristic vectors are computed using *pathmax* and the selection of nodes from the list OPEN is based on the K -ordering. We shall refer to this algorithm as MOA^{**} . The algorithm outline and the proof of its admissibility is presented in appendix A.

Since the new algorithm MOA^{**} utilizes the path information through *pathmax*, it follows from the properties of *pathmax* that the new algorithm will be superior to MOA^* in terms of the worst case number of nodes expanded. In this section we discuss the advantages of using K -ordering in the new algorithm.

We first show that the set of nodes expanded by MOA^{**} does not depend on the way in which the induced total ordering is defined. In other words we show that instead of using K -ordering if the non-dominated nodes are selected from OPEN in any other order, the total number of nodes expanded by MOA^{**} in the worst case remains the same.

Theorem 4. *A node n is expanded by MOA^{**} only if n and all its ancestors have one or more cost vectors that are either equal to or non-dominated by the cost vector of every solution node in the search space. If the node n and its ancestors also have at least one cost vector that is non-dominated by the cost vectors of every solution path not containing n , then the node n will definitely be expanded by MOA^{**} .*

Proof. We prove the sufficiency condition first. If a node n and all its ancestors have one or more non-dominated cost vectors then step 2 of MOA^{**} shows that the node is a candidate for expansion. However, if every non-dominated cost vector of n (or of one of its ancestors) equals the cost vector of other solution paths, then it is possible that those solution paths are found earlier and n is never expanded. Otherwise, it is easy to see that the node n will be expanded by MOA^{**} .

Now let us consider a node n selected for expansion in [step 3] of MOA^{**} . It is expanded if its representative cost vector is non-dominated by every solution node found so far. Let us assume that this cost vector is dominated by the cost vector of some non-dominated goal node γ in the search space which has not been found so far. This leads us to a contradiction because until γ is found, OPEN will contain either γ or one of its ancestors, each of which has one or more cost vectors that dominate the cost vector of γ (and therefore dominate the representative cost vector of n). The result follows by induction on the ancestors of n . $\square$

Based on theorem 4, we define the nodes that are *surely expanded* by MOA^{**} as follows.

DEFINITION 8

Surely expanded nodes: If a node n and each of its ancestors have one or more cost vectors that are non-dominated by every solution path which does not contain n , then n is **surely expanded** by MOA^{**} .

Since theorem 4 establishes the fact that the set of *surely expanded nodes* will have to be expanded by any admissible strategy irrespective of the order of selection of non-dominated nodes from OPEN, it follows that in the worst case, the set of nodes *surely expanded* by MOA^* and MOA^{**} are the same. Thus the optimality results proved by Stewart & White (1991) for MOA^* extends to MOA^{**} as well.

We now establish a property of MOA^{**} which will be useful in describing the advantage of using *K-ordering* in partial order search.

Theorem 5. *The non-dominated solution nodes are found by MOA^{**} in strictly increasing K -ordered sequence of their cost vectors.*

Proof. Let γ_1 and γ_2 be two non-dominated solutions in the search space such that the cost vector of γ_1 is less than the cost vector of γ_2 in K -order but γ_2 is found by MOA^{**} before γ_1 . This means that when γ_2 was selected from OPEN, there were no node in OPEN having one or more cost vectors that dominate the cost of γ_1 because then that node would have had a smaller representative cost vector. This leads us to a contradiction because OPEN must contain either γ_1 or one of its ancestors which will contain a cost vector that dominates the cost vector of γ_1 . $\square$

Based on the above results, the following advantages of using *K-ordering* becomes apparent.

- A. *Dominance checking in two-objective cases:* Since the solution nodes are found in a *K-ordered* sequence of their cost vectors, the list of solution vectors found so far is automatically sorted in the first component. In two-objective problems (which are found to occur frequently) the task of testing a cost vector for dominance by solutions can be done in constant time simply by comparing the second dimension of the vector with the second dimension of the latest solution cost vector in the list of solution vectors.
- B. *Computing all heuristic vectors is not necessary:* If the heuristic function can generate the heuristic cost vectors of a node in *K-order* then we may adopt the policy of computing only the representative cost vector of the node at the time of its generation rather than computing the entire set. Since the node may be expanded by virtue of this representative cost vector we save the computational effort of generating the other heuristic vectors of the node.

The above advantages of using *K-ordering* can easily be generalized to any induced total order on partial order heuristic search. In the next section we shall illustrate the advantage of using *K-ordering* in memory bounded partial order search and show that in the general situation the use of *K-ordering* is a key to perform the search efficiently within given memory constraints.

6. Memory bounded multiobjective search

The exorbitant space requirements of search strategies such as A^* have motivated researchers to develop schemes for searching within memory constraints. In the past decade several memory bounded heuristic search strategies have been developed for the general search model. The linear space iterative deepening algorithm IDA^* of Korf (1985) is one of the foremost among such strategies. The algorithm MA^* of Chakrabarti *et al* (1989) uses the given memory to retain the *best* portion of the state space and thereby reduce the number of node re-expansions. The algorithm IDA^*_{CR} of Sarkar *et al* (1991) and the algorithm DFS^* of Rao *et al* (1991) use suitable techniques to determine the cost cut-off of the next iteration in a way so as to reduce the number of node re-expansions. Other important contributions in the area of restricted memory search include the strategies MREC of Sen & Bagchi (1989), IE and SMA^* of Russel (1992), RBFS of Korf (1992, 1993) and IDA^*_{CRM} of Sarkar *et al* (1992).

In this section we show that the main difficulty in extending standard memory bounded search techniques from the general search model to the multiobjective search model lies in the cost back-up mechanism. We further show that the policy of using *K-ordering* can resolve this problem. This fact is established by incorporating *K-ordering* in a multiobjective search algorithm and showing that the algorithm is admissible and operates in linear space.

6.1 Cost back-up and K-ordering

A standard feature of all asymptotically optimal memory bounded search strategies is the act of backing up of costs while backtracking. When such algorithms backtrack from the current path to select a better path, they typically back up the cost of the current path before pruning it, so that the same path is selected only if all other paths of lower cost have been visited. Thus before removing a portion of the explicit search space from the memory the algorithm ensures that the *best* cost from the pruned space is backed up.

Whereas in the conventional search model it is possible to identify a single undisputed *best* cost from the pruned space by using the total order on the cost structure, the same is not possible in the multiobjective model due to the partial order on the cost vectors which allows the existence of multiple non-dominated cost vectors in the pruned space. Since the number of such vectors may be quite large, in general it will not be possible to back-up the entire set of non-dominated vectors due to space limitations. Therefore it is necessary to adopt some technique so that not many cost vectors need to be backed up and yet admissibility is guaranteed. We show that the policy of using an induced total ordering (such as K-ordering) to decide the direction of the search is one such technique to resolve the problem.

In the following sub-sections we shall show that if the policy of searching in a K-ordered best-first manner is adopted then it is possible to back-up only a single cost vector from the pruned space while backtracking. In order to implement this policy over the entire search mechanism (which includes the task of updating the cost vectors of regenerated nodes from the backed up cost vectors) we define a function called *Minf* which assigns updated cost vectors to all nodes in the explicit search space and ensures that the search proceeds in a K-ordered best-first manner.

Once the policy of using K-ordering is adopted, it becomes possible to extend standard memory bounded search techniques to the multiobjective search framework. We present a linear space algorithm called MOR*0 which incorporates the idea of *K-ordering* with standard memory bounded search techniques.

In this context it must be noted that the admissibility of MOR*0 only establishes the result that it is *possible* to guarantee admissibility by backing up a single cost vector. While this result guarantees the feasibility of MOR*0 under all situations it does not necessarily imply that the policy of backing up more than one cost vector from the pruned space is inferior to the policy of backing up a single cost vector. In fact, when sufficient memory is available, the policy of backing up more than one cost vector can typically reduce the number of node re-expansions during search. Thus there is a scope for trade-off between the number of costs to be backed up and the advantage gained due to reduction in the number of node re-expansions. These issues are considered in § 6.4 where a variant of the proposed algorithm is suggested that backs up more than one cost vector from the pruned space while backtracking.

6.2 General philosophy of MOR*0

The algorithm MOR*0 is a generalization of restricted memory heuristic search techniques (in the conventional search model) with suitable features incorporated to adapt it to the

Figure 2. The structure of a node.

multiobjective framework. In this section we describe the general structure of the nodes and the basic features of the algorithm. In the following discussion we loosely use the term “minimum” to refer to “minimum in K -order”. Also when we say that a cost vector is *greater* or *less* than another vector we mean that the comparison is on the basis of K -order.

The structure of a node is shown in figure 2. $H(n)$ denotes the set of heuristic vectors of node n using *pathmax*. Each node n maintains a list $\{F_1(n), F_2(n), \dots, F_M(n)\}$ of cost vectors, where M denotes the number of successors of n and $F_i(n)$ denotes the minimum of the estimated cost vectors of solution paths through node n and its i^{th} child. $F(n)$ denotes the minimum of $\{F_i(n)\}$ and represents the minimum estimated cost vector of solution paths through n . $F(n)$ is called the *representative cost vector* of node n .

The basic features of the algorithm MOR*0 are as follows.

Use of GL: Algorithm MOR*0 uses a vector called GL (greatest lower bound), which holds the cost vector of the current *best* path. When a new path is selected, GL denotes the estimated promise of the path.

Use of NEXT_MIN: When MOR*0 extends the most promising path, it maintains the *next best promise* in a vector called NEXT_MIN. The selected path is extended until its cost vector exceeds the cost vector stored as NEXT_MIN. When this happens, MOR*0 backtracks up to the node where the cost vector in NEXT_MIN is backed up.

Cost back-up: When MOR*0 backtracks it backs up *only the minimum (in K -order) of the cost vectors in the current path*.

Backtracking and cost revision: If the cost vector of the current search path exceeds GL, and there exists a better alternative path having one or more cost vectors that are less than the cost vector of the current path, then MOR*0 backtracks as follows.

Let p be the tip node of the current path. Let q be the parent of p and p be its j th child. The value of $F(p)$ is backed up in $F_j(q)$ and $F(q)$ is revised to the minimum among all vectors $F_i(q)$ backed up at q . If this revision alters the

vector $F(q)$, then $F(q)$ is backed up to its parent, and so on until either for some node n in the path $F(n)$ is unaltered, or the root node is reached.

We refer to this process of backing up of cost vectors as *cost revision*.

Use of Minf: When a node n is expanded to generate its children m_i , the set of cost vectors of m_i is computed by adding $G(m_i)$ with the heuristic vectors in $H(m_i)$. However, in the cases where m_i is being re-generated $F(m_i)$ should be assigned a cost vector that is consistent with the cost vector that was backed up previously as $F_i(n)$. This assignment is done by a function *Minf* which takes two aspects into account.

1. $F(m_i)$ must be assigned a value greater than or equal to $F_i(n)$.
2. The value assigned to $F(m_i)$ must be consistent with the cost vectors computed at node m_i , that is, it must be dominated by at least one cost vector of node m_i .

The function *Minf* determines the minimum vector which is greater than or equal to $F_i(n)$ and which is dominated by at least one of the cost vectors of m_i and assigns this vector to $F(m_i)$.

Test for dominance: If the vector $F(m)$ of node m is dominated by the cost vector of some solution found previously, then the dominated heuristic vectors in $H(m)$ are removed. If $H(m)$ becomes empty, then the $F(m)$ is set to infinity, and the algorithm backtracks. Otherwise, $F(m)$ is assigned a new vector by applying *Minf* on $F_i(n)$ and the new $H(m)$, where n is the parent of m , and m is the i th successor of n .

Since the search proceeds in K-order, it is easy to see that the first dimension of the representative cost vector of a node is greater than or equal to the first dimension of the cost vector of every solution found so far. Therefore, while testing for dominance against the cost vector of a solution, we ignore the first dimension.

Termination condition: When a non-dominated solution node is found, its path to the source is traced and the cost vector of the path is saved. Then the cost vector of the node is set to infinity, so that the algorithm backtracks and proceeds along alternative paths for other solutions. When all the cost vectors of a path becomes dominated by the cost vectors of solutions, infinity is backed up along that path. The algorithm terminates, when infinity is backed up along all paths.

6.3 Algorithm MOR*0

The outline of the algorithm in recursive form is given below. The *min* operator refers to the minimum in K-order.

MOR*0(node:n, cost:GL, cost:NEXT_MIN)

1. If $F(n) > GL$ return $F(n)$
2. Test for dominated heuristics
 - 2.1 Remove all dominated heuristics in $H(n)$
 - 2.2 If $H(n)$ is empty, return ∞
 - 2.3 Recalculate $F(n) \leftarrow \text{Minf}(GL, G(n), H(n))$
 - 2.4 If $F(n)$ increases then return $F(n)$
3. IF n is a goal node THEN

- 3.1 Put n in SOLUTION_GOALS and its cost in SOLUTION_COSTS
- 3.2 Output the solution path and return ∞
4. Expand n , generating all its successors.
 - 4.1 For each successor m_i of n
 - 4.1.1 Evaluate the set $H(m_i)$ of non-dominated heuristics
 - 4.1.3 Calculate $G(m_i) \leftarrow G(n) + c(n, m_i)$
 where $c(n, m_i)$ is the cost of the arc (n, m_i) .
 - 4.1.4 Use the function Minf to evaluate the value of $F(m_i)$
 - 4.1.5 Set $F_i(m_i) \leftarrow F(m_i) \forall i$
 - 4.1.6 Set $F_i(n) \leftarrow F(m_i)$
 - 4.2 Set $F' \leftarrow \min\{F_i(n), \forall i\}$.
5. Let m_j be the successor with cost F' . Set $F'' \leftarrow \min\{F_i(n), \forall i, i \neq j\}$.
 - 5.1 $F_j(n) \leftarrow MOR*0(m_j, GL, \min(NEXT_MIN, F''))$
 - 5.2 Set $F' \leftarrow \min\{F_i(n), \forall i\}$.
 - 5.3 Set $MIN \leftarrow \min(NEXT_MIN, F')$
 - 5.4 If $F' = MIN$ THEN
 - 5.4.1 Set $GL \leftarrow NEXT_MIN$
 - 5.4.2 Go to [Step 5]
 - Else return F'

6.3a *The function: Minf* Whenever the i th successor m of node n is generated, $F(m)$ is assigned the minimum vector which is greater than or equal to $F_i(n)$ and which is dominated by at least one of the cost vectors of m . The outline of the function is given below:

Minf($F_i(n), G(m), H(m)$)

Let $f_j(n)$ denote the j th dimension of $F_i(n)$.

1. Construct a set $\hat{F}(m)$ in the following way:
 - 1.1 For each heuristic $h(m) \in H(m)$ create a new vector $f(m)$:
 - 1.1.1 $f(m) \leftarrow G(m) + h(m)$
 Let $f_j(m)$ denote the j th dimension of $f(m)$.
 - 1.1.2 For $j = 1$ to $j = K$
 If $f_j(n) \geq f_j(m)$ Then Set $f_j(m) \leftarrow f_j(n)$
 Else Go to [Step 1.1.3]
 - 1.1.3 Put $f(m)$ in $\hat{F}(m)$
2. Return the minimum vector (in K-order) from $\hat{F}(m)$.

End.

The following example illustrates the working of the algorithm *MOR*0* and justifies the use of the function *Minf*. To demonstrate the basic idea we consider a simple case where each node has a single cost vector.

Example 1. We show the operation of *MOR*0* on the tree of figure 3. Since $H(s)$ is (3,2,2), the algorithm starts with GL as (3,2,2). Node s is expanded to generate node n_1 of cost (4,3,4) and node n_2 of cost (10,10,10).

Then node n_1 is selected. The value of GL now becomes (4,3,4), and $NEXT_MIN$ is assigned the value (10,10,10). Node n_1 is expanded to generate node n_3 of cost (6,7,7) and

Figure 3. Tree illustrating the utility of *Minf*.

node n_8 of cost (7,8,8). The cost of the path has exceeded GL, but it is still less than any other alternative since NEXT_MIN is (10,10,10).

Therefore n_3 is selected. The value of GL becomes (6,7,7), and NEXT_MIN becomes (7,8,8). Node n_3 is expanded to generate node n_4 of cost (7,8,10) and node n_5 of cost (6,9,10).

Node n_5 is selected. The value of GL becomes (6,9,10), and NEXT_MIN remains (7,8,8) since the alternative cost $F(n_4)$ is worse than (7,8,8). Now node n_5 is expanded to generate nodes n_6 and n_7 both of cost (14,14,14). Now the cost of the path has not only exceeded GL, but also the value of NEXT_MIN. The algorithm therefore backtracks to the node which holds the alternative promised by NEXT_MIN, i.e. the algorithm backtracks to node n_1 since $F_2(n_1)$ is equal to NEXT_MIN. The best cost from the current path, that is, (7,8,10) is backed up at $F_1(n_1)$.

Now n_8 is selected. The value of GL becomes (7,8,8), and NEXT_MIN becomes (7,8,10) which incidentally is the backed up cost from n_3 . Node n_8 is now expanded to generate nodes n_9 and n_{10} both of cost (13,13,13). The algorithm backtracks again to node n_1 , and selects n_3 on the basis of the backed up cost (7,8,10) which is the current value of NEXT_MIN. The cost (13,13,13) is backed up from node n_8 . The value of GL becomes (7,8,10), and NEXT_MIN becomes (13,13,13).

In the next step, n_3 is re-expanded. If the costs are evaluated without using *Minf*, then $F(n_4)$ is evaluated as (7,8,10) and $F(n_5)$ is evaluated as (6,9,10) (the same values as those evaluated when they were generated for the first time). In that case node n_5 will be selected and re-expanded.

However, using *Minf*, $F(n_4)$ is evaluated as (7,8,10) and $F(n_5)$ is evaluated as (7,9,10). Therefore, n_4 is selected and expanded. Subsequently $F(n_{11})$ is evaluated as (7,8,10), and

the solution is obtained. It is easy to see that this cost dominates all other costs in the explicit search space, and therefore the algorithm terminates without expanding any other node. The unnecessary re-expansion of node n_5 is therefore avoided by virtue of using the function *Minf*. $\square$

6.3b Admissibility of MOR*0: In this section, we prove the admissibility of MOR*0 and establish the equivalence of MOR*0 and MOA** in terms of the set of nodes expanded by them. In the following analysis when the terms *greater*, *less*, *minimum*, *maximum* etc. are used with respect to vectors, we mean that the comparison is on the basis of *K-order*.

*Lemma 1. Whenever a node containing the backed-up cost vector of a previously generated (and consequently pruned) path is selected, MOR*0 will expand at least one new node.*

Proof. MOR*0 continues the extension of a path P until the representative cost vector of the extended path exceeds the cost vector of the next best alternative. When this happens, MOR*0 backtracks up to the node which has this next best promise (now the best). Let this node be t and let u be the child of t in the extended path P' . Since MOR*0 backs up the F -value while backtracking, $F_i(t) = F(u)$, u being the i th successor of t . The backed up cost vector $F_i(t)$ must have exceeded the previous backed-up vector (otherwise the algorithm will not backtrack). Clearly this backed up cost vector is the representative cost vector of the tip node n of some path generated by extending the path P . When the path P is selected again the node n is guaranteed to be expanded. $\square$

Lemma 1 shows that the same path will not be repeatedly re-expanded without any progress. Using this it is now easy to establish the termination of MOR*0. In order to prove that MOR*0 is admissible we first show that it expands the same set of nodes as MOA**.

*Lemma 2. MOR*0 always expands the node n with the minimum F -value $F(n)$. Ties are resolved in favor of nodes at a greater depth.*

Proof. The proof follows from the way NEXT_MIN is maintained and the policy of backtracking to the *deepest* node having a backed-up cost vector equal to NEXT_MIN. $\square$

*Lemma 3. Except for differences due to tie resolutions between nodes having representative cost vectors equal to non-dominated solution cost vectors, the representative cost vectors of new nodes expanded by MOR*0 occur in the same sequence as the representative cost vectors of nodes expanded by MOA**.*

Proof. MOA** always selects the node having the minimum representative cost vector in the explicit search space. We show that each new node expanded by MOR*0 is the one having the minimum representative cost vector among all nodes generated up to that point.

When MOR*0 generates a node n for the first time, $F(n)$ is the same as the representative cost vector from its set of cost vectors. Since $F_i(n)$ denotes the representative cost vector

of the tip node of some path through n and its i th successor and $F(n)$ is the minimum among all such $F_i(n)$, it follows that $F(n)$ is actually the minimum among the cost vectors of all leaf nodes in the subtree below node n .

MOR*0 always selects the node n with the minimum $F(n)$ and extends the path until its cost increases. Clearly $F(n)$ is the minimum representative cost vector among all nodes generated so far. During the course of regeneration there can be no node having a representative cost vector less than $F(n)$ (this is guaranteed by the use of *pathmax* and the function *Minf*). Also no node with a representative cost vector greater than $F(n)$ will be selected until the path having the cost vector $F(n)$ is traced. Thus the first new node that is expanded after selection of node n is the one having the minimum representative cost vector among all nodes generated so far.

Tie resolutions will occur among nodes having this minimum cost vector. If this cost vector is not equal to the cost vector of a non-dominated solution then each of these nodes will be expanded by both algorithms. However, if the cost vector equals that of a non-dominated solution, then depending on the tie resolutions one algorithm may select the solution earlier and prune the other nodes of equal cost. In the worst case situation, the solution node may be selected last. $\square$

Lemma 3 establishes the equivalence of MOR*0 and MOA** in terms of the worst case set of nodes expanded by them. The admissibility of MOR*0 follows from this result.

Theorem 6. *Algorithm MOR*0 is admissible, that is, it terminates with all non-dominated solutions.*

Proof. The proof follows from lemmas 1 & 3. $\square$

6.4 Variants of MOR*0:

The main drawback of MOR*0 is addressed by the following lemma.

Lemma 4. *If the number of nodes expanded by MOA** is N , then algorithm MOR*0 expands $O(N^2)$ nodes in the worst case.*

Proof. In the worst case after expanding each new node MOR*0 may have to backtrack right up to the source node and regenerate the entire portion of the search space that it had pruned earlier before expanding the next new node. Under such a situation it is easy to see that MOR*0 will expand $O(N^2)$ nodes. $\square$

Lemma 4 reflects a standard drawback of many recursive search strategies. Similar results have been obtained for algorithms such as IDA^* (Korf 1985), prompting researchers to develop techniques such as IDA^*_{CR} (Sarkar et al 1991) and DFS^* (Rao et al 1991) to reduce the number of node re-expansions in recursive search strategies. In this section we shall suggest extensions of similar techniques to improve the performance of MOR*0. We also discuss a scheme (in the following sub-section) that is applicable only to the multiobjective framework.

6.4a MOR*0 with multiple back-up cost vectors: When the algorithm MOR*0 backtracks, it backs up only the minimum cost vector (in K-order) from the pruned space. In case some solution is found during the course of search that dominates the backed up cost vector then the vector becomes useless. Since no other backed up cost vector is available, in the worst case the algorithm may have to explore the same portion of the search space again. The policy of backing up multiple non-dominated cost vectors has the following advantage:

- If all cost vectors are backed up and all of them are subsequently found to be dominated by one or more solutions, then it is possible to prune the portion of the search space below that node forever and save a lot of possible node re-expansions. Also if more than one cost vector is backed up, then whenever the representative cost vector becomes dominated the node may be represented by the next minimum backed up cost vector.

Backing up multiple non-dominated cost vectors also has the following disadvantages.

- Since algorithm MOR*0 proceeds K-ordered best-first, at each node it needs to compute only the minimum non-dominated cost vector. Computing multiple cost vectors may be expensive for certain heuristic functions.
- The space constraints may prohibit the backing up of multiple cost vectors.

Therefore depending on the situation, there is a scope for trade-off between the number of costs to be backed up and the advantage gained due to reduction in the number of nodes expanded.

6.4b MOR*0 with controlled re-expansions: In this section we suggest the extension of an established technique from the conventional search model for controlling node re-expansions in backtracking heuristic search strategies. In the conventional model the following result has been established (Korf 1985) for strategies such as IDA*:

- If for some constant b (such that $b^K = N$ for some integer K), b^i new nodes are expanded in the i th iteration, then the worst case complexity of the strategy reduces from $O(N^2)$ to $O(N)$.

Based on this result, the algorithm IDA*_CR was developed by Sarkar *et al* (1991). In each iteration i , IDA*_CR performs a depth-first branch and bound with a cutoff value x , which is set in way so as to ensure that at least b^i new nodes are expanded. To determine the cutoff value for the next iteration, IDA*_CR uses a bucketing technique for grouping the node costs that exceed the cut-off of the current iteration.

A similar technique may be used to reduce the number of node re-expansions in MOR*0. We may define an iteration to be the set of operations between successive assignments of new values to GL. An iteration ends when there are no more paths with cost less than or equal to the current value of GL (in K-order). By using a suitable bucketing technique it is possible to group the representative cost vectors of nodes which exceeded GL in an iteration. The extended algorithm then simply assigns GL a new vector such that in the i th iteration GL exceeds (or equals) the representative cost vectors of at least b^i new nodes. After a new vector is assigned to GL, this algorithm searches the space below the tip node of the current path with the new GL as cut-off. The remaining operations are identical to that of MOR*0.

6.4c *The algorithm MOR**: One way of controlling excessive node regeneration is to prune nodes only when the given memory is exhausted. In the conventional search model the algorithm *MA** (Chakrabarti *et al* 1989) uses such an idea. The algorithm *MOR** is an extension of *MA** to the multiobjective framework with suitable modifications. The algorithm makes use of *K-order* to prune away the nodes having the maximum representative cost vector in *K-order*. The operation of the algorithm is similar to that of *MOR*0* except that the pruning and cost back-up mechanism is initiated only when the given memory (which is specified through a parameter called *MAX*) is exhausted. The outline of this algorithm is given in appendix B.

The admissibility of *MOR** follows from the admissibility of *MOR*0*. In addition, the algorithm has the following property. We state it here without a proof.

- The algorithm *MOR** never uses more memory than $MAX + |C^*|$, where *MAX* is the given parameter depicting the additional amount of memory available and $|C^*|$ is the number of nodes in the longest non-dominated path in the search space.

7. Searching with inadmissible heuristics

The admissibility and optimality properties of best-first search algorithms such as *A** and *MOA** are subject to the admissibility of the heuristic function. Though an inadmissible heuristic function may not be able to guarantee admissibility, search algorithms using such heuristics are sometimes found to converge to the solution much faster than weaker admissible heuristics. The problem of searching with inadmissible heuristics has been well studied within the conventional search model (Bagchi & Mahanti 1983; Dechter & Pearl 1985; Pearl 1984). This section presents preliminary results obtained by us on multiobjective search with inadmissible heuristics.

In order to avoid exhaustive search of the state-space, the following basic policies are adopted by strategies such as *MOA** and *MOA***.

- The node which is selected from *OPEN* for expansion must be non-dominated in *OPEN*.
- If every cost vector of a node is dominated by the cost vectors of already obtained solutions, then the node will never be selected for expansion.

If the above policies are not adopted then it may be easily shown that the search turns out to be exhaustive in many situations. We now show that if the heuristic function is inadmissible, then admissibility of the algorithm cannot be guaranteed without relaxing the above policies.

Theorem 7. *If the heuristic function is inadmissible then no algorithm is guaranteed to find all non-dominated solutions unless it expands dominated nodes also.*

Proof. We prove the result by constructing a problem instance where the expansion of a dominated node becomes mandatory. Let us consider the tree in figure 4. At first node *s* is expanded to generate node n_1 having the single cost vector (6,4) and node n_2 having the single cost vector (4,6). Since $F(n_1)$ and $F(n_2)$ are mutually non-dominated either of them may be selected first by the search strategy. Let us consider both the cases.

Figure 4. Multiobjective tree with inadmissible heuristics.

1. If n_1 is selected, then the goal node n_3 of cost vector (3,5) is generated. Since $F(n_3)$ dominates $F(n_2)$, n_3 is selected next and the solution of cost vector (3,5) is recorded. This cost vector dominates $F(n_2)$. Therefore any strategy that adopts the policy of not expanding dominated nodes will not expand n_2 and so will never find the other non-dominated goal node n_4 of cost (5,3).
2. If n_2 is selected prior to n_1 , then by a similar reasoning the non-dominated solution n_3 will never be found unless the dominated node n_1 is expanded.

Thus, unless the search strategy expands a dominated node, it will not be able to find all non-dominated solutions. $\square$

The above result also shows that depending upon the policy adopted by an algorithm to decide which nodes to expand first, different sets of solution nodes will be discovered by the search. In this section we analyze the conditions for a given solution path to be discovered by MOA**. In the following discussion when the terms *greater*, *less*, *minimum*, *maximum* etc. are used with respect to vectors, we mean that the comparison is on the basis of *K-order*.

DEFINITION 9

[*pmax*]: We define a function called “pmax” as follows. Given a path P , $pmax[P]$ returns the maximum of the representative cost vectors of all nodes on the path P . We shall also refer to the vector returned by $pmax[P]$ as “*pmax* of P ”. $\square$

DEFINITION 10

[*pmax-ordering*]: We define a strict ordering on the search paths as follows. We shall refer to this ordering as “*pmax-ordering*”.

Given any two paths P_1 and P_2 :

1. If $pmax[P_1] < pmax[P_2]$ then $P_1 < P_2$ based on *pmax-ordering*.
2. If $pmax[P_2] < pmax[P_1]$ then $P_2 < P_1$ based on *pmax-ordering*.
3. If $pmax[P_1] = pmax[P_2]$ then
 - 3.1 Let $n_1 \in P_1$ and $n_2 \in P_2$ be the shallowest nodes in P_1 and P_2 respectively such that the representative cost vector of n_1 equals $pmax[P_1]$ and the representative cost vector of n_2 equals $pmax[P_2]$.
 - 3.2 If n_1 and n_2 are the same node then
 - If the node is a tip node then

$P_1 < P_2$ or $P_2 < P_1$ (arbitrarily, based on the tie-breaking rules).
 else
 3.2.1 Let P'_1 be the subpath of P_1 from n_1 onwards
 and P'_2 be the subpath of P_2 from n_2 onwards
 3.2.2 If $P'_1 < P'_2$ on pmax-ordering
 then $P_1 < P_2$ on pmax-ordering.
 3.2.3 If $P'_2 < P'_1$ on pmax-ordering
 then $P_2 < P_1$ on pmax-ordering.
 else when n_1 and n_2 are different nodes then
 $P_1 < P_2$ or $P_2 < P_1$ (arbitrarily, based on the tie-breaking rules). □

DEFINITION 11

Eligible node: We say that a node n is “eligible” for expansion if it satisfies at least one of the following conditions.

1. Let P_{min} be the minimum solution path based on “pmax-ordering”. If n belongs to P_{min} then n is eligible.
2. A node n is eligible for expansion if it has at least one cost vector $f(n)$ which is non-dominated by the cost vector of every eligible solution path P such that $pmax[P]$ is less than $f(n)$. A path is eligible if all its nodes are eligible. □

Based on the above definitions, the following properties are easy to establish for the algorithm MOA** (see appendix A).

- The necessary and sufficient condition for a node n to be expanded is that all nodes on the path $P(s, n)$ from the start node s to the node n are *eligible*.
- If two paths P_1 and P_2 are eligible, then P_1 is generated prior to P_2 if and only if $P_1 < P_2$ in *pmax-ordering*.
- MOA** finds the solution paths in the *pmax-ordered* sequence of the paths.

If we also relax the assumption that any cost vector of a finite path dominates every cost vector of an infinite path (assumption A.3), then infinite paths may have finite cost vectors. If such a path is *eligible* up to infinite depth, then it is easy to see that MOA** will not terminate. Taking all these aspects into account, the general conditions of admissibility of MOA** may be stated as follows.

*Admissibility conditions of MOA***

MOA** terminates with all non-dominated solutions iff:

1. There is no infinite path which is eligible up to infinite depth, and
2. Every non-dominated solution path is eligible, and
3. The pmax-ordering of the solution paths describe the same sequence as the representative cost vector of the paths based on K-ordering.

The third condition may be relaxed by modifying step 5 of MOA** as follows:

5. [IDENTIFY SOLUTIONS]

If n is a goal node then

- 5.1 Put n in SOLUTION_GOALS and its cost in SOLUTION_COSTS.

5.2 Remove dominated solutions (if any) from SOLUTION_COSTS.

5.3 Go to [Step 2].

The introduction of step 5.2 becomes necessary because if the third condition is relaxed then the solution nodes may not arrive in the K-ordered sequence of their cost vectors and it is possible that some solution node entered in SOLUTION_GOALS is dominated by the cost vector of some solution found later.

8. Conclusion

In this paper we have addressed two major topics under multiobjective state space search. The first has been to refine the basic search scheme proposed by Stewart & White (1991) by incorporating algorithmic improvements. We have extended the standard technique of *pathmax* to the multiobjective framework and have shown that in a way it is more significant in the multiobjective model than in the conventional model. We have also incorporated ideas such as using an induced total order (such as K-order) on the partial order search.

The other major issue addressed in this paper has been to develop memory bounded search strategies. We have shown that the major difficulty in extending standard memory bounded search schemes to this model is due to the presence of multiple candidate back-up cost vectors in the pruned space, and that the problem can be resolved by using a total order on the search. We have presented the algorithm MOR*0 to establish this fact.

References

- Bagchi A, Mahanti A 1983 Search algorithms under different kinds of heuristics: A comparative study. *J. Assoc. Comput. Mach.* 30: 1–21
- Bogetoft P 1986 General communication schemes for multiattribute decision making. *Eur. J. Oper. Res.* 26: 108–122
- Chakrabarti P P, Ghose S, Acharya A, DeSarkar S C 1989 Heuristic search in restricted memory. *Artif. Intell.* 41: 197–221
- Dechter R, Pearl J 1985 Generalized best-first search strategies and the optimality of A*. *J. Assoc. Comput. Mach.* 32: 505–536
- Gelparin D 1977 On the optimality of A*. *Artif. Intell.* 8: 69–76
- Keeney R L, Raiffa H 1976 *Decisions with multiple objectives: Preferences and value trade-offs*. (New York: John Wiley)
- Kok M 1986 The interface with decision makers and some experimental results in interactive multiple objective programming methods. *Eur. J. Oper. Res.* 26: 96–107
- Korf R E 1985 Depth-first iterative deepening: An optimal admissible tree search. *Artif. Intell.* 27: 97–109
- Korf R E 1992 Linear-space best-first search: Summary of results. In *Proc. of AAAI-92*, pp. 533–538
- Korf R E 1993 Linear-space best-first search. *Artif. Intell.* 62: 41–78
- Korhonen P J 1986 A hierarchical interactive method for ranking alternatives with multiple qualitative criteria. *Eur. J. Oper. Res.* 24: 265–276
- Korhonen P J, Moskowitz H, Wallenius J 1986 A progressive algorithm for modeling and solving multiple-criteria decision problems. *Oper. Res.* 34: 726–731
- Martelli A 1977 On the complexity of admissible search algorithms. *Artif. Intell.* 8: 1–13

- Mero L 1984 A heuristic search algorithm with modifiable estimate. *Artif. Intell.* 23: 13–27
- Mond B, Rosinger E E 1985 Interactive weight assessment in multiple attribute decision making. *Eur. J. Oper. Res.* 22: 19–25
- Nilsson N J 1980 *Principles of artificial intelligence*. (Palo Alto, CA: Tioga)
- Pearl J 1984 *Heuristics: Intelligent search strategies for computer problem solving* (New York: Addison Wesley)
- Rao V N, Kumar V, Korf R E 1991 Depth-first vs best-first search. In *Proc of AAAI-91*, pp. 434–440
- Russel S 1992 Efficient memory-bounded search methods. In *Proc of ECAI-92*
- Sarkar U K, Chakrabarti P P, Ghose S, DeSarkar S C 1991 Reducing reexpansions in iterative deepening search by controlling cutoff bounds. *Artif. Intell.* 50: 207–221
- Sarkar U K, Chakrabarti P P, Ghose S, DeSarkar S C 1992 Effective use of memory in iterative deepening search. *Inf. Process. Lett.* 42: 47–52
- Sen A K, Bagchi A 1989 Fast recursive formulations for best-first-search that allow controlled use of memory. In *Proc. IJCAI-89*, pp. 297–302
- Stewart B S 1988 *Heuristic search with a general order relation*. Ph D thesis, Univ. of Virginia, Charlottesville, VA
- Stewart B S, White C C 1991 Multiobjective A^* . *J. Assoc. Comput. Mach.* 38: 775–814
- Vansnick J C 1986 On the problem of weights in multiple criteria decision making. *Eur. J. Oper. Res.* 24: 288–294

Appendix A. Algorithm MOA**

The algorithm outline (for trees) is given below. The set of heuristic vectors at node n is denoted by $H(n)$ and the set of cost vectors of node n is denoted by $F(n)$. If any dimension of a cost vector in $F(n)$ is infinity, it would signify that the cost vector is useless, since it does not underestimate the cost of any finite solution. If all the cost vectors of a node are infinity, then the node can be discarded altogether.

Algorithm MOA**

1. [INITIALIZE]

OPEN $\leftarrow s$; SOLUTION_GOALS $\leftarrow \phi$; SOLUTION_COSTS $\leftarrow \phi$;

2. [TERMINATE]

If OPEN is empty then

- 2.1 Output the solutions from SOLUTION_GOALS.
- 2.2 Terminate.

3. [SELECT]

- 3.1 Remove the node n in OPEN with the minimum representative cost vector in K-order. Let this cost vector be $f(n)$. Resolve ties in favor of goal nodes, else arbitrarily.

4. [DOMINANCE CHECK]

If the representative cost vector of node n is dominated by the cost vector of a solution in SOLUTION_COSTS then:

- 4.1 Remove $f(n)$ from $F(n)$.

- 4.2 Select the next non-dominated cost vector of n (in K-order).
 - 4.2.1 If there is no non-dominated cost vector in $F(n)$ then
 - 4.2.1.1 Discard node n .
 - 4.2.1.2 Go to [Step 2].
 - 4.2.2 Otherwise declare the new vector as the representative cost of n
 - 4.2.2.1 Return node n to OPEN.
 - 4.2.2.2 Go to [Step 3].

5. [IDENTIFY SOLUTIONS]

If n is a goal node then

- 5.1 Put n in SOLUTION_GOALS and its cost vector in SOLUTION_COSTS.
- 5.2 Go to [Step 2].

6. [EXPAND]

Expand n and examine its successors.

- 6.1 Generate the successors of n .
- 6.2 If n has no successors, Go to [Step 2].
- 6.3 Otherwise, for each successor n_j of n , do the following:
 - 6.3.1 Evaluate the non-dominated set of heuristics $H(n_j)$ using *pathmax*.
 - 6.3.2 Compute the set $F(n_j)$ by adding the cost vectors in $H(n_j)$ with $G(n_j)$.
 - 6.3.3 Determine the minimum cost vector in $F(n_j)$ using K-ordering.
 Declare that cost as the representative cost vector of n_j .
 - 6.3.4 Enter n_j in OPEN.

7. [ITERATE]

Go to [step 2]

A.1 Admissibility of MOA^{**}

This section deals with the admissibility and optimality issues of MOA^{**} . The properties of MOA^{**} are very much similar to those of the algorithm A^* in the conventional search domain. Similar properties were proved for the algorithm MOA^* by Stewart and White (Stewart & White 1991). The objective of this section is to establish the correctness of MOA^{**} and highlight some of the other properties of the algorithm. The following assumptions define the domain in which MOA^{**} is admissible.

Assumption A.1. The heuristic function is admissible.

Assumption A.2. The number of children of a node are finite.

Assumption A.3. All non-dominated solutions are of finite length and have finite cost vectors, and the cost vectors of infinite paths are dominated by solutions.

The following two theorems establish the admissibility of MOA^{**} .

Theorem A.1. *If there exists a goal node γ in the search space, then MOA^{**} is guaranteed to terminate.*

Proof. We first show that paths are expanded only up to a finite depth. Let us consider a node n at infinite depth. By assumption A.3, each of its cost vectors is dominated by the cost vector of every solution path. Since the cost vector of the path $P(s, \gamma)$ is finite, the cost vector of γ and some cost vector of every ancestor of γ dominate every cost vector of node n . Until γ is selected either γ or one of its ancestors must be present in OPEN. Since MOA** selects only those nodes that are non-dominated in OPEN and non-dominated by SOLUTION_COSTS, it follows that the node n at infinite depth will never be expanded.

The number of successors of a node are finite (by assumption A.2), which also implies that the number of paths up to a finite depth are finite. Since a finite set of paths are expanded up to a finite depth, the termination of MOA** is guaranteed. $\square$

Theorem A.2. *MOA** is admissible, that is, it terminates with all non-dominated solutions in the entire search space.*

Proof. When the heuristics are admissible, every ancestor node of a non-dominated solution node will contain a cost vector that either dominates or is equal to the cost vector of the solution node. It follows that every ancestor of a non-dominated goal node is also non-dominated. Therefore, until the solution node is found OPEN will contain a non-dominated node and MOA** cannot terminate. Since termination is guaranteed by theorem A.1 the result follows. $\square$

In this context it must be noted that the admissibility of MOA** is subject to the admissibility of the heuristic function, that is, each ancestor node of a goal contains at least one cost vector which dominates the cost vector of the solution. If there exists a non-dominated solution γ in the search space such that some node n in the path from s to γ does not have a cost vector which dominates the cost of γ , then MOA** is not guaranteed to find the solution γ . This happens when all the cost vectors of node n are dominated and node n is discarded by MOA**. In order to guarantee that all non-dominated solutions are found under such situations, an algorithm will have to expand dominated nodes as well. This indicates that brute-force search strategies may be the only strategies that are guaranteed to find all non-dominated solutions. However, it is easy to see that under such circumstances MOA** is guaranteed to find all those goals whose ancestors have admissible heuristics.

Appendix B. The outline of algorithm MOR*

The following is the outline of the algorithm MOR*. The algorithm uses a parameter called MAX which indicates the number of nodes that are allowed to be retained in the memory. The algorithm uses a variable called *node_count* to count the number of nodes currently retained in the memory.

Algorithm MOR*

1. [INITIALIZE]

```

OPEN  $\leftarrow$   $s$  ; CLOSED  $\leftarrow$   $\phi$  ;
SOLUTION_GOALS  $\leftarrow$   $\phi$  ; SOLUTION_COSTS  $\leftarrow$   $\phi$  ;
node_count  $\leftarrow$  0

```

Compute the set of cost vectors of s .

Assign the representative vector of s to $F(s)$ and each $F_i(s)$.

2. [TERMINATE]

If OPEN is empty then Terminate.

3. [SELECT 1]

3.1 Remove the node n in OPEN with minimum $F(n)$ in K-order.

3.2 Resolve ties in favor of the goal nodes, else in favor of nodes at a greater depth.

3.3 Set $GL \leftarrow F(n)$

4. [DOMINANCE CHECK]

If $F(n)$ is dominated by the cost vector of a solution in SOLUTION_COSTS then

4.1 Compute the next representative cost vector of n .

4.1.1 If there is no such non-dominated vector then

4.1.1.1 Discard node n .

4.1.1.2 Go to [Step 2].

4.1.2 Otherwise return n to OPEN and Go to [Step 3].

5. [IDENTIFY SOLUTIONS]

If n is a goal node then

5.1 Put its cost vector in SOLUTION_COSTS.

5.2 Output the solution.

5.3 Set $F(n) \leftarrow \infty$

5.4 Go to [Step 2]

6. [EXPAND]

6.1 Expand n generating all its successors.

6.2 For each successor n_j of n , do the following:

6.2.1 Compute the representative cost vector $F(n_j)$ of n .

6.2.2 Assign the vector $F(n_j)$ to each $F_i(n_j)$

6.2.3 Set $F_j(n) \leftarrow F(n_j)$

6.2.4 Enter n_j in OPEN

6.2.5 Increment node_count

6.3 Put n in CLOSED

7. [SELECT 2]

7.1 Select the successor m of n with minimum $F(m)$ in K-order.

7.2 If $F(m) \leq GL$ then select m for expansion and call it n .

7.3 Go to [Step 4].

8. [COST_REVISION]

8.1 Put the node m in a set Z

8.2 Repeat the following steps until Z is empty.

8.2.1 Remove the node p from Z. Let q be its parent and p be the j th successor of q .

8.2.2 Set $F_j(q) \leftarrow F(p)$

8.2.3 Let $F_{min}(q)$ denote the minimum among all $F_i(q)$ in K-order.

8.2.4 If $F(q) < F_{min}(q)$ then

8.2.4.1 Set $F(q) \leftarrow F_{min}(q)$

8.2.4.2 Put q in Z

9. [CONTINUE]

9.1 If $\text{node_count} \leq \text{MAX}$ then Go to [Step 2]

10. [PRUNING]

10.1 Select the leaf node r in OPEN with largest $F(r)$ in K-order.

10.2 If $F_1(r) = GL$ then Go to [Step 3].

10.3 Remove r from OPEN and decrement node_count .

10.4 Let t be the parent of r . If t is in CLOSED then

10.4.1 Remove t from CLOSED and put it in OPEN

10.5 Go to [Step 10.1]