

Anytime Algorithms for Biobjective Heuristic Search

Priyankar Ghosh, Partha Pratim Chakrabarti, and Pallab Dasgupta

Indian Institute of Technology Kharagpur, West Bengal, India 721302
{priyankar, ppchak, pallab}@cse.iitkgp.ernet.in

Abstract. Heuristic search algorithms (MOA*, NAMOA* etc) for multiobjective optimization problems, when run with admissible heuristics, typically spend a lot of time and require huge amount of memory for generating the Pareto optimal solution frontier due to the fact that these algorithms expand all nodes/paths whose cost is not dominated by any other optimal solution. In this paper, we present an anytime heuristic search framework for biobjective optimization problems named “Anytime Biobjective Search (ABS)” which quickly finds a set of nondominated solutions and keeps on improving the solution frontier with time. The proposed framework uses the upper and lower limit estimates on one of the objectives to split the search space into a given number of segments and independently runs a particular search algorithm (branch-and-bound, beam search etc.) within each of the segments. In this paper, we present how existing search strategies, branch-and-bound, beam, and beam-stack, can be used within the proposed framework. Experimental results reveal that our proposed framework achieves good anytime performance.

1 Introduction

A large set of optimization problems encountered in real-life applications involve two non commensurate objectives where one typically strives to reach a suitable trade-off between the conflicting objectives. We briefly mention some of the problems below. The operator scheduling problem [1] which is encountered during high level synthesis of digital designs involves two independent objectives – (a) *delay*, and (b) *total number of resources*. For example, with 5 resources the minimum delay schedule may need 8 time units, whereas increasing the number of resources to 7 may allow to have the minimum delay schedule with 6 time units. In finance, a common problem that involves two conflicting objectives are to choose a portfolio [2] where the non commensurate objectives are – (a) *the expected value of portfolio returns* (should be as high as possible), and (b) *the risk factor* (should be as low as possible). Another similar problem arises in connection with macro-economic policy making for central banks [3], where the desired solution tries to achieve a balance between two independent objectives, namely, *inflation* and *balance of trade deficit*. For the domain of traffic/transport information systems, similar problems often arise [4]. For example, in individual route planning for vehicles, *fuel cost* and *delay*, are two conflicting objectives. In the route guidance problem, *the system optimal routing* and *user constraint in the form of delay*, form two conflicting objectives [5].

Heuristic search [6] is often considered for determining optimal/near-optimal solutions for single as well as multiobjective optimization problems [7–9]. The best first

heuristic search algorithms, namely, A*, MOA*, and NAMOA* etc., when run with an admissible heuristic, typically spend a considerable amount of time for selecting the best node/path to be explored next. Often the techniques that quickly generate near optimal solutions are preferred. In the context of single objective optimization problems, several such techniques [10–12] are developed. Also combining multiple heuristics [13] and simultaneously searching with multiple setting [14] are studied for the single objective optimization problems. Another popular alternative is to use anytime algorithms which start with generating some sub-optimal solution and keep on generating better quality solution over time [15–19]. However, anytime algorithms are only studied for single objective optimization problems.

For multiobjective optimization problems, several techniques [20–25] have been developed for finding near optimal solutions. These methods typically aim to find a solution frontier which is within a factor of the optimal solution. Several approximation schemes (PTAS, FPTAS, etc.) for multiobjective optimization problems have been proposed [20, 22–24]. In [26], a branch-and-bound based method is presented which uses the notion of *separating hypersurface* for pruning. However, all of these methods work on explicit graphs except the method in [21]. Perny et al. [21] presents an MOA* based ϵ -admissible algorithm for multiobjective search. Although this method works on implicit representation, the proposed algorithm is not an anytime algorithm. There is a significant body of literature [27, 28] on applying evolutionary computing based methods for multiobjective optimization problems for finding near optimal solutions. However, these methods are not evaluated in anytime setting.

In this paper, we present an anytime heuristic search framework named “Anytime Biobjective Search (ABS)” that works on problems with biobjective cost criterion. To the best of our knowledge, this is the first attempt to develop anytime algorithms for optimization problems with more than one objectives. So far, anytime algorithms has been developed for single objective optimization problems only. Our proposed algorithm is particularly useful for biobjective optimization problems whose search space are implicit in nature and typically described using a set of state transformation rules. The proposed framework quickly finds an initial nondominated solution frontier and keeps on improving the solution frontier with time.

We use the idea of splitting the search space using the upper and lower limit estimates on one of the objectives into a given number of non-overlapping segments. Using multiple stacks to expedite the branch-and-bound search for single objective optimization problems had been proposed in [29]. Within each of the segments, ABS independently runs a specific search algorithm (branch-and-bound, beam etc.) and tries to balance the search effort over the segments. We present how existing search strategies, namely, branch-and-bound, beam, and beam-stack etc., can be used within the proposed framework. While using running ABS, the solution found in a particular sub-range is used for further pruning of the other sub-ranges. Experimental results reveal that our proposed framework achieves good anytime performances and sometimes outperforms NAMOA* in terms of finding the optimal solution frontier. In this paper, at first we present the necessary formalisms and definitions required for discussing the proposed method. The proposed algorithm, ABS, is presented in the following section. Then, we present the experimental results followed by the concluding remarks.

2 Definitions

Let $G = \langle V, E \rangle$ be a locally finite directed graph, where V is the set of nodes, $\Gamma \subseteq V$ is the set of goal nodes, and $E \subseteq V \times V$ is the set of directed edges. Every edge $e_{ij} = (v_i, v_j) \in E$ is labeled with a positive 2-dimensional vector, $c(e_{ij}) \in \mathbb{R}^+ \times \mathbb{R}^+$, where c_1 and c_2 are the two cost-components of vector, c . A path in G is a sequence of nodes, $p = \langle v_1, \dots, v_k \rangle$, such that $(\forall i, 1 \leq i < k), (v_i, v_{i+1}) \in E$. The path cost $g(p) = (\sum_{i=1}^{k-1} c_1(c(e_{i,i+1})), \sum_{i=1}^{k-1} c_2(c(e_{i,i+1})))$, is the sum of the cost vectors of its component edges. For a path p , we use $c_1(g(p))$ and $c_2(g(p))$ to denote $\sum_{i=1}^{k-1} c_1(c(e_{i,i+1}))$ and $\sum_{i=1}^{k-1} c_2(c(e_{i,i+1}))$, respectively. A solution is a path in G from start node v_s to a goal node $v_t \in \Gamma$. Since the edge costs are 2-dimensional vectors, a set of Pareto optimal solutions can be computed for G . Without loss of generality, we assume that both objectives are to be minimized.

Definition 1. [Dominance] We define two types of dominance relations, simple dominance, denoted using ' $\preceq$ ', and strict dominance, denoted using ' $\prec$ ' as follows.

1. $\forall g_i, g_j \in \mathbb{R}^2, [g_i \preceq g_j] \Leftrightarrow [(c_1(g_i) \leq c_1(g_j)) \wedge (c_2(g_i) \leq c_2(g_j))]$
2. $\forall g_i, g_j \in \mathbb{R}^2, [g_i \prec g_j] \Leftrightarrow [g_i \preceq g_j] \wedge (g_i \neq g_j)$ □

Definition 2. [Nondominated Set] For a set of vectors, $\mathcal{U}$, we define the set of non-dominated vectors in $\mathcal{U}$ as $nd(\mathcal{U}) = \{g_i \in \mathcal{U} \mid \nexists g_j \in \mathcal{U}, g_j \prec g_i\}$. □

We also use two lexicographical orderings (L_1 -order and L_2 -order) which induce a total order among the cost vectors.

Definition 3. [L-ordering] Let g_i and g_j be two 2-dimensional vectors. Then, based on L_1 -ordering, $g_i < g_j$, if either $c_1(g_i) < c_1(g_j)$ or $[c_1(g_i) = c_1(g_j)] \wedge [c_2(g_i) < c_2(g_j)]$. Based on L_2 -ordering, $g_i < g_j$, if either $c_2(g_i) < c_2(g_j)$ or $[c_2(g_i) = c_2(g_j)] \wedge [c_1(g_i) < c_1(g_j)]$. The other relational operators ($=$ and $>$) based on L_1 -ordering and L_2 -ordering can be defined in a likewise manner. □

For each node v_i in a graph G , $\mathcal{H}(v_i)$, denotes the set of nondominated heuristic cost vectors at node v_i such that every vector $h(v_i) \in \mathcal{H}(v_i)$ denotes the estimate of cost vector of one or more solution paths from v_i to goal nodes in Γ . Clearly for each goal node, $v_t \in \Gamma$, $\mathcal{H}(v_t) = \{(0, 0)\}$.

Definition 4. [Heuristic Evaluation Vector] For every path, p , in $\mathcal{G}$ starting from v_s to a node v_i , with cost $g(p)$, the set of heuristic evaluation vectors is denoted by $\mathcal{F}(p) = \mathcal{F}(v_i, g_i) = nd(\{f \mid [f = g(p) + h] \wedge [h \in \mathcal{H}(v_i)]\})$ □

Definition 5. [Admissible Heuristics] A heuristic function is said to be admissible if the set $\mathcal{H}(v_i)$ of heuristic vectors computed using that function for each node v_i satisfies that for every solution path, p , from v_i to a goal node having nondominated cost, $h^*(p)$, $\exists h(v_i) \in \mathcal{H}(v_i)$ such that $h(v_i) \preceq h^*(p)$. □

In the rest of the paper we use c_1^l and c_1^u to denote the lower bound and upper bound estimates of the c_1 cost-components of the solutions. Suppose, Σ represents the set of all solutions. Clearly, $(\forall p \in \Sigma), [c_1(g(p)) \geq c_1^l] \wedge [c_2(g(p)) \leq c_1^u]$. Such estimates can be computed upfront using some approximate method.

3 Proposed Algorithmic Framework

In this section we present a generic anytime heuristic search framework for biobjective optimization problems named, “Anytime Biobjective Search” (ABS). Our proposed framework uses the lower and upper bound estimates on one of the objectives and divides the entire value range of that objective into a given number of sub-ranges to add a breadth first component to the search effort. Without loss of generality we assume that the lower and upper bound estimate is provided for the first objective function, i.e., c_1^l and c_1^u are given as inputs. The entire range of values, $c_1^u - c_1^l$, is divided into a given number, k , of sub-ranges. The sub-ranges may or may not be equal to each other. Several popular anytime strategies, like branch-and-bound etc. can be used on top of our framework. We denote the i^{th} subrange as an interval $J_i = [r_i, r_i + 1)$ where $1 \leq i \leq k - 1$. Since $r_{k+1} = c_1^u$ the last subrange is $J_k = [r_k, r_k + 1]$. Each subrange is searched independently using a particular search method and the subrange to be searched next is selected using a given selection policy.

Algorithm 1. Anytime Biobjective Search(ABS)

input : The start node v_s , c_1^l , c_1^u , the number of ranges k , and strategy σ
output : Pareto optimal solution frontier

- 1 Construct the root node of the explicit graph $\mathcal{G}$ with v_s ;
- 2 Invoke strategy specific initialization function, $Init\text{-}\sigma$;
- 3 **while** $\exists i$, such that $1 \leq i \leq k$ **and** $\delta_i^\sigma \neq \emptyset$ **do**
- 4 Use a given policy to determine the value of i such that $(1 \leq i \leq k)$ and $\delta_i^\sigma \neq \emptyset$;
- 5 Start/Resume the search σ on δ_i^σ using $SearchRange\text{-}\sigma$;
- 6 **end**
- 7 Report the solutions in $\mathcal{S}$;

In this paper, we investigate three strategies, namely, (a) Branch-and-bound (denoted as *BB*) (b) Beam search (denoted as *BM*), and (c) Beam-stack search (denoted as *BmS*) on our proposed framework. For each sub-range, J_i ($1 \leq i \leq k$), ABS maintains a separate data-structure δ_i^σ . The data-structures depend on the underlying strategy $\sigma \in \{BB, BM, BmS\}$. For example, δ_i^{BB} can be implemented as a queue or stack depending on whether *breadth first* or *depth first* variation of branch-and-bound is used. The major steps of the proposed framework are shown in Algorithm 1.

ABS expands the given implicit graph, G , repeatedly over iterations using state transformation rules to construct and grow the *explicit graph*, $\mathcal{G}$. A *solution frontier*, $\mathcal{S}$, containing the set of nondominated solutions is maintained throughout the course of the algorithm. ABS uses path selection and path expansion as the basic operations. At any point of time ABS maintains the set of *potential solution paths* (the paths that may be extended to reach a solution) corresponding to i^{th} sub-range in δ_i^σ . ABS selects a particular sub-range, say the j^{th} one, using the given selection policy and invokes the strategy specific search routine, $SearchRange\text{-}\sigma$ on δ_j^σ . For our experimentation, we use round-robin selection policy. However, ABS can work with any other selection policies.

We maintain the paths to every node v_d of the explicit graph $\mathcal{G}$ using a list, $\mathcal{P}(v_d)$. Every element of $\mathcal{P}(v_d)$ is a triplet, $\langle v_q, j, \mathbf{g}(p) \rangle$, representing a distinct path, p , to node, v_d , where triplet $\langle v_q, j, \mathbf{g}(p) \rangle$ denotes that path p is an extension of the j^{th} path to the

parent v_q of v_d and have cost $\mathbf{g}(p)$. Every *potential solution path* is also represented using a triplet, $\langle v_d, \mathbf{g}_d, ptr \rangle$, which denotes that the corresponding potential solution path ends at node v_d and have cost $\mathbf{g}_d$. ‘ ptr ’ stores the pointer to the entry representing the corresponding path in $\mathcal{P}(v_d)$.

The individual strategy specific steps are carried out in two different subroutines, namely *Initialize*(σ) and *SearchRange*- $\sigma(\delta_i^\sigma)$. However, all the search strategies use the following routine, *UpdateSolutions*, to maintain the current solution frontier $\mathcal{S}$. Whenever a new solution path is found, *UpdateSolutions* checks whether the new solution, p , is dominated by any of the existing solutions in $\mathcal{S}$ and if the p is a nondominated one, then all the existing solutions in $\mathcal{S}$ that are dominated by p is removed from $\mathcal{S}$. Next, we explain the working of beam search, branch-and-bound, and beam-stack search within ABS framework.

Function *UpdateSolutions*($\mathcal{S}, \langle v_q, \mathbf{g}_q, ptr_q \rangle$)

/* Suppose, for goal v_q , p_q is the corresponding solution path */

- 1 **foreach** $p_i \in \mathcal{S}$ **do** **if** $\mathbf{g}(p_i) \prec \mathbf{g}_q$ **then return**;
- 2 **foreach** $p_j \in \mathcal{S}$ **do**
- 3 **if** $\mathbf{g}_q \prec \mathbf{g}(p_j)$ **then** Remove p_j from $\mathcal{S}$;
- 4 **end**
- 5 Add p_q to $\mathcal{S}$;

[Branch-and-bound Search] The traditional *branch-and-bound* strategy performs an ordered traversal of a given search space where the pruning is performed based on the cost of current best solution. The traversal is *depth-first* when the underlying list of nodes is implemented as a *stack* and *breadth-first* when that list is implemented as *queue*. In the multiobjective setting, the pruning is performed when the cost of a potential solution is dominated by one of the solutions found so far. ABS framework is used to implement a *multiple-list* variation of the traditional branch-and-bound search, called *multiple-list branch-and-bound (MLBB)* in the context of biobjective costs in the following way. The entire search space is split into a given number of non-overlapping segments based on the value of the c_1 cost-component. Within each of the segments branch-and-bound search is run independently. The entire range of values of c_1 cost-component, $(c_1^u - c_1^l)$, is divided into a given number, k , of equal sub-ranges and for each sub-range a separate list is maintained. δ_i^{BB} denotes the list for i^{th} sub-range.

In each round of the *multiple-list branch-and-bound*, the lists are selected one by one starting from the first list. If the selected list is non-empty, function *SearchRange-BB* is invoked on that list. In *SearchRange-BB*, the list is popped/dequeued (depending on whether the list is implemented as stack or queue respectively) to select the next entry (a potential solution path in the explicit graph $\mathcal{G}$) to be expanded. If a solution is found, function *UpdateSolutions* is used to update the current solution frontier and the function *SearchRange-BB* returns to the main driver function (Algorithm 1) to allow the search to proceed in other segments. Otherwise, *SearchRange-BB* expands the selected entry to generate only those successors whose costs are not dominated by the cost of the solutions in the current solution frontier and which are not a candidate for pruning. Each of the generated successors is inserted into the appropriate list (δ_i^{BB}) according to the range to which its c_1 cost-component of the path cost vector $\mathbf{g}$ belongs.

We have shown the *SearchRange-BB* function for the *depth-first* version of *multiple list branch-and-bound*. For implementing the *breadth-first* version of *multiple-list branch-and-bound* each list δ_i^{BB} has to be implemented using queue.

It may be noted that whenever a new nondominated solution is found, the stacks are pruned using the cost vector of that new solution. Also while generating the successors, it may happen that the cost vector of the newly found path might dominate the cost vector of some of the paths generated earlier. In this case those paths which are dominated are pruned along with their corresponding stack entries.

Function Initialize-BB (k)

- 1 $\forall i, 1 \leq i \leq k$, Initialize δ_i^{BB} ;
 - 2 Insert $\langle v_s, \mathbf{0}, null \rangle$ to δ_1^{BB} ;
-

Function SearchRange-BB (δ_i^{BB})

- 1 $p_\delta \leftarrow \text{Pop}(\delta_i^{BB})$; /* Suppose $p_\delta = \langle v_q, \mathbf{g}_q, ptr_q \rangle$ */
 - 2 **if** $v_q \in \Gamma$ **then**
 - 3 UpdateSolutions($\mathcal{S}, \langle v_q, \mathbf{g}_q, ptr_q \rangle$);
 - 4 $\forall j, i \leq j \leq k$, prune δ_j^{BB} , using $\mathbf{g}_q$;
 - 5 **return**;
 - 6 **end**
 - 7 Expand p_δ and generate the set of valid successors δ_{succ}^{BB} which represent potential solution paths;
 - 8 **foreach** element p'_δ in δ_{succ}^{BB} **do**
 - 9 Suppose $p'_\delta = \langle v_d, \mathbf{g}_d, ptr_d \rangle$;
 - 10 $idx \leftarrow \lceil (c_1^u - c_1(\mathbf{g}_d)) / k \rceil$;
 - 11 Push ($\delta_{idx}^{BB}, p'_\delta$);
 - 12 **end**
 - 13 **goto** Line-1;
-

[Beam Search] Typically *beam-search* works with a given beam-width, b . In the ABS framework in the initialization phase (function *Initialize-BM*), NAMOA* is run until the size of δ_{open}^{BM} reaches $k \times b$ where k and b are two parameters of the algorithm. Here, k represents the number of beams to be used and b is the size of an individual beam.

Function Initialize-BM (k, b)

- 1 Construct an empty list δ_{open}^{BM} ;
 - 2 Insert $\langle v_s, \mathbf{0}, null \rangle$ to δ_{open}^{BM} ;
 - 3 Continue NAMOA* search using δ_{open}^{BM} as *OPEN* list till $|\delta_{open}^{BM}| \leq k \times b$;
 - 4 Order the elements of δ_{open}^{BM} according to the c_1 cost component;
 - 5 Split δ_{open}^{BM} into k consecutive sublists where each sublist has no more than b entries;
 - 6 **for** $j \leftarrow 1$ **to** k **do**
 - 7 $\delta_j^{BM} \leftarrow t^{th}$ sublist of δ_{open}^{BM} ;
 - 8 $r_j \leftarrow$ the c_1 cost component of the first element of δ_j^{BM} ;
 - 9 **end**
 - 10 $r_{k+1} \leftarrow c_1^u$;
-

The contents of δ_{open}^{BM} is sorted in L_1 -order and split into k sublists. We maintain k beams parallelly and function *SearchRange-BM* starts beam search on a given beam δ_i^{BM} . It returns whenever a new solution is found. It may be noted that the k ranges obtained after splitting the contents of δ_{open}^{BM} may not be equal and primarily depends on the search space. Also this method is not complete due to the fact that beam search itself is not complete.

Function SearchRange-BM (δ_i^{BM})

- 1 Expand the deepest layer d of δ_i^{BM} and generate the set of valid successors δ_{succ}^{BM} where each successor represents a potential solution path;
 - 2 **if** a solution $\langle v_q, \mathbf{g}_q, ptr_q \rangle$ is generated in the last step **then**
 - 3 UpdateSolutions($\mathcal{S}, \langle v_q, \mathbf{g}_q, ptr_q \rangle$);
 - 4 **return**;
 - 5 **end**
 - 6 **if** $\delta_{succ}^{BM} = \emptyset$ **then**
 - 7 Remove the contents of the all layers of δ_i^{BM} ;
 - 8 **return**;
 - 9 **end**
 - 10 Sort the elements of δ_{succ}^{BM} in the non-decreasing order the c_1 cost component;
 - 11 Add the first b elements of δ_{succ}^{BM} to the $(d+1)^{th}$ layer of δ_i^{BM} ;
 - 12 **goto** Line-1;
-

[Beam-stack Search] In order to make the beam search complete, *beam-stack search* was proposed by Zhou et al. in [10]. The beam-stack search can be applied within our proposed ABS framework in the following way. The initialization phase is identical to *beam search* and in this phase a heuristic breadth first search is run till the size of δ_{open}^{BmS} reaches $k \times b$ where k and b are the same two parameters of the algorithm that are used in beam search. Then the contents of δ_{open}^{BM} is sorted in L_1 -order and split into k sublists. In the ABS framework k beam-stacks are maintained in parallel where each beam-stack, δ_i^{BmS} ($1 \leq i \leq k$), consists of multiple layers of maximum width b . A $\langle \text{level-range, slice-index} \rangle$ pair is maintained for each layer. For the layer with depth d of i^{th} beam-stack the *level-range* is denoted as $[\tau_{i,d}^{min}, \tau_{i,d}^{max}]$ which denotes the range of the values of c_1 cost-components of successors in the next layer. The successors of a layer are sorted in L_1 -order for ordered traversal of the search space. Since the width of each layer is limited by the beam-width b , the successors of a layer are explored in an ordered fashion by keeping a slice of b successors only. When the search backtracks to the current layer the next slice of successors (containing at most b elements) is explored. Since the beam-stack search strategy is complete, it is guaranteed to generate optimal solution frontier given sufficient time. However experimental results show that such time for finding the optimal solution frontier may be very large.

4 Experimental Results

Since NAMOA* [9] uses the same underlying path selection and expansion policy and the overall performance of NAMOA* is better than that of MOA* [30], we compare the performance of ABS framework with NAMOA*. Also the same problem domain

name	t_1	t_2	t_3	t_4	t_5
dim.	61×61	71×71	81×81	91×91	101×101
$\#S$	61	84	92	98	106
$T(\theta_{100})$	21.94	63.87	113.26	178.85	219.41

name	t_6	t_7	t_8	t_9	t_A
dim.	111×111	121×121	131×131	141×141	151×151
$\#S$	118	134	146	154	167
$T(\theta_{100})$	362.40	743.82	1869.44	2673.13	4289.72

Fig. 1. Average running time (in seconds) for NAMOA*

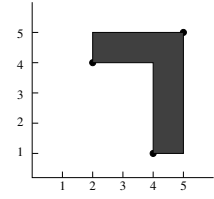


Fig. 2. Hypervolume

which was used in [30, 31], namely randomly constructed $N \times N$ square grids is used for comparison. We have conducted the experiments using a machine with Intel Core i3-2100 cpu, 4GB RAM, running Ubuntu Linux 10.10.

We have constructed 25 grid instances for each N . The average time required for computing the optimal solution frontier for grids with different dimensions using NAMOA* are reported in Table 1. The cost of each edge in a $N \times N$ grid is a two dimensional vector, c , where both the c_1 and c_2 components of c are chosen randomly between 1 and 5. The node at position $(0, 0)$ is the source and the node at position (N, N) is the destination. The Manhattan distance heuristic function was used in this problem. We used the hypervolume metric [27] to determine the quality of the solution frontier. In Figure 2 the area of shaded region is the hyper volume of the Pareto frontier $\{(2, 4), (4, 1)\}$ with respect to the point $(5, 5)$. The hypervolume is computed with respect to the point (c_1^u, c_2^u) , where c_1^u and c_2^u are the upper bound estimates of the c_1 and c_2 cost-component respectively. The optimality of a solution frontier is defined as :

$$\theta = \frac{\text{hypervolume of the solution frontier}}{\text{hypervolume of the optimal solution frontier}} \times 100$$

The time required in seconds for reaching $x\%$ optimality is denoted using $T(\theta_x)$. Also we use θ_i to denote the optimality measure of the first solution returned by an anytime algorithm. We have deliberately used very large time-out value (5000 seconds) while running NAMOA* in order to generate the actual Pareto optimal frontier so that the anytime performance of the different search strategies under ABS framework can be compared with respect to the exact Pareto optimal frontier.

For evaluating the performance of *multiple list branch-and-bound* using ABS framework, the number of lists, i.e., the value of k is chosen in such a way that the width of the interval represented by an individual subrange does not exceed a certain value, denoted using m , i.e., $r_{i+1} - r_i \leq m$ where $1 \leq i \leq k$. Table 1 and Table 2 report the *average* running time required for reaching different levels of optimality for the *depth-first* and *breadth-first* versions of *multiple list branch and bound* using ABS framework respectively. We have used a time-limit of 600 seconds in order to compare the anytime performance of different search strategies under ABS framework. We have experimented with different values of m (10, 15, 20, 25) and we report the result for $m = 10$ which is the best among the lot. We choose this value based on the results Between the depth-first and breadth-first version of MLBB on ABS, the depth-first version is the clear winner. Also it is interesting to observe that the performance of depth-first MLBB on ABS often beats the performance of NAMOA* for certain valuations of k .

The *average* running time required for *beam* search on ABS framework is reported in Table 3 for square grids with different dimensions. For experimentation we have

Table 1. Running time (in seconds) of depth-first MLBB on ABS framework (** denotes timeout after 600 seconds) for different dimensions of grids

	θ_i	$T(\theta_i)$	$T(\theta_{80})$	$T(\theta_{85})$	$T(\theta_{90})$	$T(\theta_{95})$	$T(\theta_{100})$
t_1	76.127%	2.258	4.142	5.375	6.063	7.035	8.677
t_2	75.363%	5.836	9.441	14.384	15.525	16.443	18.142
t_3	76.537%	11.233	18.664	25.457	28.316	31.566	34.480
t_4	74.913%	18.786	31.861	42.193	46.484	53.737	58.871
t_5	76.773%	31.237	52.972	72.567	85.654	96.649	105.565
t_6	75.241%	50.174	83.832	106.852	139.355	158.316	176.789
t_7	74.871%	79.286	121.243	188.393	216.746	244.948	266.635
t_8	77.227%	116.741	187.549	274.408	310.888	338.776	363.426
t_9	75.732%	173.255	303.122	395.732	465.256	512.963	553.989
t_A	76.479%	242.459	442.198	485.941	498.296	*	*

Table 2. Running time (in seconds) of breadth-first MLBB on ABS framework (** denotes timeout after 600 seconds) for different dimensions of grids

	θ_i	$T(\theta_i)$	$T(\theta_{80})$	$T(\theta_{85})$	$T(\theta_{90})$	$T(\theta_{95})$	$T(\theta_{100})$
t_1	68.610%	5.634	33.762	52.722	61.048	66.842	72.893
t_2	50.844%	8.040	129.840	184.408	220.631	237.398	258.468
t_3	65.827%	19.602	240.028	290.514	434.276	465.715	480.692
t_4	66.840%	28.326	151.222	330.203	551.338	*	*
t_5	60.499%	48.182	382.505	476.866	*	*	*
t_6	63.124%	75.083	315.121	545.938	*	*	*
t_7	65.058%	126.612	322.617	*	*	*	*
t_8	61.583%	182.997	345.139	*	*	*	*
t_9	67.104%	249.657	344.915	*	*	*	*
t_A	62.441%	348.272	355.810	*	*	*	*

Table 3. Running time (in seconds) of ABS with beam search strategy (- denotes that the target value of optimality not reached) for different dimensions of grids

	θ_i	$T(\theta_i)$	$T(\theta_{80})$	$T(\theta_{85})$	$T(\theta_{90})$	$T(\theta_{95})$	$T(\theta_{100})$
t_1	73.545%	0.873	0.655	1.379	3.342	4.723	-
t_2	71.020%	1.072	1.080	3.648	8.572	-	-
t_3	74.363%	1.488	2.472	6.326	12.048	-	-
t_4	72.829%	2.034	4.587	10.707	11.322	-	-
t_5	72.886%	2.537	5.923	12.289	29.034	-	-
t_6	69.072%	3.872	7.241	25.776	-	-	-
t_7	73.666%	3.824	14.655	35.451	-	-	-
t_8	71.922%	5.303	15.818	44.723	-	-	-
t_9	70.554%	6.028	26.375	85.089	-	-	-
t_A	71.802%	7.241	35.893	109.074	-	-	-

Table 4. Running time (in seconds) of ABS with beam-stack search strategy (** denotes timeout after 600 seconds) for different dimensions of grids

	θ_i	$T(\theta_i)$	$T(\theta_{80})$	$T(\theta_{85})$	$T(\theta_{90})$	$T(\theta_{95})$	$T(\theta_{100})$
t_1	73.545%	0.429	0.015	2.924	4.533	8.431	272.342
t_2	71.020%	1.465	1.932	5.990	10.092	86.574	287.048
t_3	74.363%	1.273	3.939	7.407	15.319	244.771	482.726
t_4	72.829%	2.031	4.832	12.383	16.877	425.720	*
t_5	72.886%	2.847	5.313	13.827	31.503	*	*
t_6	69.072%	3.587	7.260	27.963	*	*	*
t_7	73.666%	4.098	15.029	36.197	*	*	*
t_8	71.922%	5.829	17.886	45.615	*	*	*
t_9	70.554%	6.357	30.812	88.198	*	*	*
t_A	71.802%	7.290	33.489	112.175	*	*	*

chosen the value of k to be equal to N for an $N \times N$ grid. The value of *beam width*, b , is taken as 20. Table 4 reports the *average* running time required for *beam-stack* search on ABS framework for square grids with various dimensions. Here also we have chosen the value of k to be equal to the N for an $N \times N$ grid while running the experiments.

Table 5. Running time (in seconds) of depth-first MLBB on ABS framework for different dimensions of knapsacks

#items	θ_i	$T(\theta_i)$	$T(\theta_{75})$	$T(\theta_{80})$	$T(\theta_{85})$	$T(\theta_{90})$	$T(\theta_{95})$	$T(\theta_{100})$
50	81.640%	0.001	0.001	0.001	0.002	0.010	0.046	0.816
52	81.253%	0.001	0.001	0.001	0.004	0.015	0.085	1.221
54	82.170%	0.001	0.001	0.001	0.003	0.015	0.093	1.519
56	81.091%	0.001	0.001	0.002	0.005	0.020	0.095	1.732
58	82.994%	0.001	0.001	0.001	0.003	0.015	0.120	2.638
60	81.918%	0.001	0.001	0.001	0.004	0.017	0.099	2.303

Table 6. Running time (in seconds) of breadth-first MLBB on ABS framework for different dimensions of knapsacks

#items	θ_i	$T(\theta_i)$	$T(\theta_{75})$	$T(\theta_{80})$	$T(\theta_{85})$	$T(\theta_{90})$	$T(\theta_{95})$	$T(\theta_{100})$
50	74.069%	0.009	0.075	0.162	0.484	1.481	4.408	5.651
52	70.215%	0.014	0.105	0.238	0.695	2.720	8.574	11.603
54	68.271%	0.019	0.152	0.234	0.725	2.509	8.815	15.382
56	67.938%	0.021	0.202	0.424	1.326	4.275	9.141	16.024
58	64.251%	0.035	0.181	0.231	0.811	4.446	14.359	23.550
60	63.994%	0.038	0.293	0.519	1.686	6.171	12.677	20.279

Table 7. Running time (in seconds) of ABS with beam search strategy (‘-’ denotes that the target value of optimality not reached) for different dimensions of knapsacks

#items	θ_i	$T(\theta_i)$	$T(\theta_{75})$	$T(\theta_{80})$	$T(\theta_{85})$	$T(\theta_{90})$	$T(\theta_{95})$	$T(\theta_{100})$
50	92.057%	0.074	0.074	0.074	0.075	0.086	0.109	-
52	90.588%	0.069	0.069	0.069	0.071	0.089	0.129	-
54	90.939%	0.073	0.073	0.073	0.076	0.086	0.148	-
56	90.300%	0.070	0.070	0.070	0.073	0.103	-	-
58	90.090%	0.074	0.074	0.074	0.074	0.109	-	-
60	89.879%	0.073	0.073	0.073	0.074	0.106	-	-

Table 8. Running time (in seconds) of ABS with beam-stack search strategy for different dimensions of knapsacks

	θ_i	$T(\theta_i)$	$T(\theta_{75})$	$T(\theta_{80})$	$T(\theta_{85})$	$T(\theta_{90})$	$T(\theta_{95})$	$T(\theta_{100})$
50	92.057%	0.075	0.075	0.075	0.075	0.086	0.328	1.429
52	90.588%	0.070	0.070	0.070	0.075	0.296	1.193	3.591
54	90.939%	0.070	0.070	0.070	0.071	0.175	0.981	3.893
56	90.300%	0.070	0.070	0.071	0.072	0.123	1.489	4.663
58	90.090%	0.075	0.075	0.075	0.075	0.147	1.852	7.537
60	89.879%	0.072	0.072	0.072	0.072	0.124	1.149	4.625

The value of *beam-stack width*, b , is also taken as 20. On ABS framework, the initial anytime performance of beam-stack search is identical to the performance of beam search. However after a reaching certain level of optimality, beam-stack takes significant time to find the optimal solution. For most of the cases the algorithm encountered time-out after 600 seconds without finding the optimal solution. The overall performance of the depth-first version of MLBB on ABS is better than other strategies.

Apart from the two dimensional grids, we have also experimented with biobjective knapsack problems. We experimented with different number of items (50, 52, $\dots$, 60) and randomly constructed 50 instances of knapsacks for each item-count. The capacity of the knapsack is maintained as the half of the total weight of the items. The weights and the profits are chosen as random number between 50 and 100. We report the result for $m = 10$ for ABS with multiple list branch and bound. We use 100 as beam-width and the number of beams used is 10. Table 5 - 8 reports the anytime performance of different strategies on ABS framework. In this test domain the performance of beam-stack search is much better compared to the performance in the grid domain and comparable with the performance of the depth-first version of MLBB on ABS. However the overall performance of the depth-first version of MLBB on ABS is better than other strategies.

5 Conclusion

In this paper, we presented a general anytime framework, ABS, for finding the Pareto optimal solution frontier problems with a biobjective cost criterion. The class of optimization problems considered in this paper is of practical significance and has possible application in problems like *operator scheduling*, *route planning*, *portfolio selection* etc., where two non commensurate objectives are involved and one typically strives to reach a suitable trade-off between the conflicting objectives.

References

1. Gajski, D., Dutt, N., Wu, A., Lin, S.: High-level synthesis: introduction to chip and system design. Kluwer Academic Publishers, Norwell (1992)
2. Elton, E.J., Gruber, M.J., Blake, C.R.: Modern portfolio theory, 1950 to date. SSRN eLibrary (1997)
3. Mishkin, F.S.: The Economics of Money, Banking, and Financial Markets, 7th edn. Addison-Wesley (2004)
4. Müller-Hannemann, M., Weihe, K.: On the cardinality of the Pareto set in bicriteria shortest path problems. *Annals OR* 147(1), 269–286 (2006)
5. Jahn, O., Möhring, R.H., Schulz, A.S., Stier-Moses, N.E.: System-optimal routing of traffic flows with user constraints in networks with congestion. *Operations Research* 53(4), 600–616 (2005)
6. Nilsson, N.J.: Principle of artificial intelligence. Tioga Publishing Co. (1980)
7. Stewart, B.S., White III, C.C.: Multiobjective A*. *J. ACM* 38, 775–814 (1991)
8. Dasgupta, P., Chakrabarti, P.P., DeSarkar, S.C.: Multiobjective heuristic search in AND/OR graphs. *Journal of Algorithms* 20(2), 282–311 (1996)
9. Mandow, L., Pérez-de-la-Cruz, J.L.: Multiobjective A* search with consistent heuristics. *J. ACM* 57(5), 27:1–27:25 (2010)
10. Zhou, R., Hansen, E.A.: Beam-stack search: Integrating backtracking with beam search. In: ICAPS, pp. 90–98 (2005)
11. Furcy, D., Koenig, S.: Limited discrepancy beam search. In: IJCAI, pp. 125–131 (2005)
12. Likhachev, M., Gordon, G.J., Thrun, S.: ARA*: Anytime A* with provable bounds on sub-optimality. In: NIPS (2003)
13. Röger, G., Helmert, M.: The more, the merrier: Combining heuristic estimators for satisficing planning. In: ICAPS, pp. 246–249 (2010)

14. Valenzano, R.A., Sturtevant, N.R., Schaeffer, J., Buro, K., Kishimoto, A.: Simultaneously searching with multiple settings: An alternative to parameter tuning for suboptimal single-agent search algorithms. In: ICAPS, pp. 177–184 (2010)
15. Vadlamudi, S.G., Aine, S., Chakrabarti, P.P.: MAWA* - a memory-bounded anytime heuristic-search algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B* 41(3), 725–735 (2011)
16. Thayer, J.T., Ruml, W.: Anytime heuristic search: Frameworks and algorithms. In: *The Third Annual Symposium on Combinatorial Search* (2010)
17. Likhachev, M., Ferguson, D., Gordon, G.J., Stentz, A., Thrun, S.: Anytime search in dynamic graphs. *Artif. Intell.* 172(14), 1613–1643 (2008)
18. Hansen, E.A., Zhou, R.: Anytime heuristic search. *J. Artif. Intell. Res. (JAIR)* 28, 267–297 (2007)
19. Aine, S., Chakrabarti, P.P., Kumar, R.: AWA* - a window constrained anytime heuristic search algorithm. In: *IJCAI*, pp. 2250–2255 (2007)
20. Tsaggouris, G., Zaroliagis, C.D.: Multiobjective optimization: Improved FPTAS for shortest paths and non-linear objectives with applications. *Theory Comput. Syst.* 45(1), 162–186 (2009)
21. Perny, P., Spanjaard, O.: Near admissible algorithms for multiobjective search. In: *ECAI*, pp. 490–494 (2008)
22. Angel, E., Bampis, E., Kononov, A.: On the approximate tradeoff for bicriteria batching and parallel machine scheduling problems. *Theor. Comput. Sci.* 306(1-3), 319–338 (2003)
23. Erlebach, T., Kellerer, H., Pferschy, U.: Approximating multiobjective knapsack problems. *Manage. Sci.* 48(12), 1603–1612 (2002)
24. Papadimitriou, C.H., Yannakakis, M.: On the approximability of trade-offs and optimal access of web sources. In: *FOCS*, pp. 86–92 (2000)
25. Barichard, V., Hao, J.K.: A Population and Interval Constraint Propagation Algorithm. In: Fonseca, C.M., Fleming, P.J., Zitzler, E., Deb, K., Thiele, L. (eds.) *EMO 2003. LNCS*, vol. 2632, pp. 88–101. Springer, Heidelberg (2003)
26. Sourd, F., Spanjaard, O.: A multiobjective branch-and-bound framework: Application to the biobjective spanning tree problem. *INFORMS Journal on Computing* 20(3), 472–484 (2008)
27. Branke, J., Deb, K., Miettinen, K., Slowinski, R.: *Multiobjective Optimization*. LNCS, vol. 5252. Springer, Heidelberg (2008)
28. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evolutionary Computation* 6(2), 182–197 (2002)
29. Sarkar, U.K., Chakrabarti, P.P., Ghose, S., Sarkar, S.C.D.: Multiple stack branch and bound. *Inf. Process. Lett.* 37(1), 43–48 (1991)
30. Machuca, E., Mandow, L., Pérez-de-la-Cruz, J.L., Ruiz-Sepúlveda, A.: A comparison of heuristic best-first algorithms for bicriterion shortest path problems. *European Journal of Operational Research* 217(1), 44–53 (2012)
31. Mandow, L., Pérez-de-la-Cruz, J.L.: A multiobjective frontier search algorithm. In: *IJCAI*, pp. 2340–2345 (2007)