An Evaluation of Heuristic Functions for Bicriterion Shortest Path Problems *

E. Machuca, L. Mandow and J.L. Pérez de la Cruz

Dpto. Lenguajes y Ciencias de la Computación. Universidad de Málaga 29071 - Málaga (Spain).
{machuca, lawrence, perez}@lcc.uma.es

Abstract. The paper considers a set of heuristic search algorithms for the Bicriterion Shortest Path Problem. These are presented as particular cases of a more general algorithm. Their performance is evaluated over a random problem set using two general heuristic functions proposed by Tung & Chew (1992). The experimental results show that the heuristics reduce the space requirements of the algorithms considered. One contribution of this paper is to show that, contrary to intuition, no improvement in time performance is achieved in these cases by using heuristics. In fact, one of the heuristics appears to work against time efficiency. The paper provides also an explanation to the observed phenomena and points out possible lines of improvement.

1 Introduction

Heuristic search in Shortest Path Problems is a central field of study in Artificial Intelligence. The A* algorithm [1] [2][3] is an efficient solution for the case of admissible graph search guided by single-objective heuristic evaluation functions.

The Bicriterion Shortest Path Problem (BSP) is an extension of the Shortest Path Problem with practical applications in different domains, like routing in multimedia networks [4], route planning [5] satellite scheduling [6], or domain independent planning [7]. BSP problems require the evaluation of two different and frequently conflicting objectives for each alternative. These problems rarely have a single optimal solution. Most frequently, a set of *nondominated* (Pareto-optimal) solutions can be found, each one presenting a particular trade-off between the objectives under consideration. The number of nondominated solutions in BSP problems is known to grow exponentially with solution depth in the worst case [8]. Fortunately, several classes of interesting multiobjective problems do not exhibit this worst-case behavior [9].

The Artificial Intelligence and Operations Research communities have contributed several extensions of the A* algorithm to the multiobjective case. These include Tung-Chew [10], MOA* [11], and NAMOA* [12]. One of the fundamental differences between these algorithms is their path/node selection strategies. The selection strategy of NAMOA* was shown to have better formal properties than that of MOA*, and to improve in space requirements both formally and empirically [12] [13]. However, to the

* This work is partially funded by/Este trabajo está parcialmente financiado por: Consejería de Innovación, Ciencia y Empresa. Junta de Andalucía (España), P07-TIC-03018

author's knowledge, there are no results comparing the performance of the heuristic selection strategies of NAMOA* and Tung-Chew.

The work of Tung & Chew [10] described also two general heuristic functions for multiobjective shortest path problems. The real impact of these heuristic functions in algorithm performance has never been tested with empirical results. This paper considers a general bicriterion search procedure that encompasses both NAMOA* and Tung-Chew. The impact of Tung and Chew's heuristics in the time and space requirements of several particular instances of this procedure is evaluated over a random problem set. Several of the algorithms considered exhibit a similar reduction in space requirements when compared to uninformed search. However, contrary to intuition, no improvement in time performance is achieved. In fact, one of the heuristics considered seems to work against time efficiency. The paper provides an explanation to the observed phenomena and points out possible lines of improvement.

The structure of the paper is as follows. Section 2 introduces some common terminology useful to understand BSP problems and a description of the heuristics analyzed in the paper. It also describes a general multiobjective search procedure that helps to identify the four particular algorithms used in the experiments. Section 3 describes the experimental setup, and analyzes the space and time requirements of the algorithms considered. An adequate explanation of the observed phenomena is presented. Finally some conclusions and future work are outlined.

2 Algorithms

2.1 Bicriterion Shortest Path Problems

Let us consider two q-dimensional vectors $\boldsymbol{v}, \boldsymbol{v}' \in \mathbb{R}^q$. A partial order relation $\prec$ denominated *dominance* is defined as follows, $\boldsymbol{v} \prec \boldsymbol{v}'$ iff $\forall i (1 \leq i \leq q)\ v_i \leq v'_i$ and $\boldsymbol{v} \neq \boldsymbol{v}'$, where v_i denotes the i-th component of vector $\boldsymbol{v}$.

Given two q-dimensional vectors $\boldsymbol{v}$ and $\boldsymbol{v}'$ (where $q > 1$), it is not always possible to say that one is better than the other. For example in a bidimensional cost space vector $(2, 3)$ dominates $(2, 4)$, but no dominance relation exists between $(2, 3)$ and $(3, 2)$.

Given a set of vectors X, we shall define $nd(X)$ as the set of non-dominated vectors in X, i. e., $nd(X) = \{\boldsymbol{x} \in X \mid \nexists \boldsymbol{y} \in X \quad \boldsymbol{y} \prec \boldsymbol{x}\}$

A total order relation $\prec_L$ denominated *lexicographic order* is defined as follows, $\boldsymbol{v} \prec_L \boldsymbol{v}'$ iff for some j, $v_j < v'_j$ and $\forall i < j$, $v_i = v'_i$. A useful property of the lexicographic order is that the lexicographic optimum in a set of vectors is also a non-dominated vector.

Let G be a locally finite labeled directed graph $G = (N, A, \boldsymbol{c})$ with $|N|$ nodes and $|A|$ arcs (n, n') labeled with positive vectors $\boldsymbol{c}(n, n') \in \mathbb{R}^q$. The cost of a path is defined as the sum of the costs of its arcs; obviously, the cost of a path is a q-dimensional vector. A multiobjective search problem in G is stated as follows:

Given a start node $s \in N$ and a set of goal nodes $\Gamma \subseteq N$, find the set of all *non-dominated* paths P in G, i. e., the set of all paths P such that

i) P goes from s to a node in Γ;

ii) the cost of P is non-dominated by the costs of any other path satisfying i).

The Bicriterion Shortest Path Problem (BSP) is a particular case of MSP in which $q = 2$, i. e., arc costs have two real components.

2.2 A general algorithm

The general algorithm ran in the experiments reported in this paper, is presented in table 1. This algorithm is a slight generalization of NAMOA* [12]. In fact, the only difference with NAMOA* lies in step 3. This difference will be explained in subsection 2.4.

The main features of the algorithm are as follows:

- The algorithm uses an acyclic search graph SG to record interesting partial solution paths. For each node n in SG two sets —$G_{cl}(n)$ and $G_{op}(n)$— denote the sets of non-dominated cost vectors of paths reaching n that have or have not been explored yet respectively (i.e. closed or open). Each cost vector in these sets labels one or more pointers emanating from n to its parents with that cost. Initially, s is the only node in SG.
- The algorithm keeps a list $OPEN$ of partial solution paths that can be further expanded. For each node n in SG and each nondominated cost vector $\boldsymbol{g} \in G_{op}(n)$ there will be a corresponding label $(n, \boldsymbol{g}, F(n, \boldsymbol{g}))$ in $OPEN$. Initially, the tuple $(s, \boldsymbol{g}_s, F(s, \boldsymbol{g}_s))$ is the only label in $OPEN$. Notice that contrary to what happens in single-objective search, many different nondominated paths may reach a given node. Therefore, it is the number of cost vectors stored in $G_{op}(n)$ and $G_{cl}(n)$ what determines memory requirements, while the number of nodes plays a minor role.
- At each iteration the algorithm will consider the extension of an open label $(n, \boldsymbol{g}, F)$ that stands for a partial solution path from s to n with cost $\boldsymbol{g}$.
- Two sets —$GOALN$ and $COSTS$— record all goal nodes reached and all non-dominated cost vectors to goal nodes respectively. Once a solution is known, its cost vector can be used to discard (filter) dominated open labels.
- Search terminates only when the $OPEN$ list is empty, i.e. when all open labels have been selected or filtered.
- Heuristic estimates will normally involve a set of vectors $H(n)$ for each node n. These vectors estimate the costs of nondominated paths from n to each goal node. Let $\boldsymbol{g}(P)$ be the function that returns the cost of path P, defined by the sum of the costs of all its component arcs. Therefore, for each path P_{sn} from s to n with cost $\boldsymbol{g}(P_{sn}) = \boldsymbol{g}_P$, there will be a set of heuristic evaluation vectors $F(P_{sn})$. This function is the multiobjective analogue of $f(n)$ in A*, $F(P_{sn}) = F(n, \boldsymbol{g}_P) = nd(\{\boldsymbol{f} \mid \boldsymbol{f} = \boldsymbol{g}_P + \boldsymbol{h} \ \wedge \boldsymbol{h} \in H(n)\})$. In this paper, we will consider the situation where $H(n) = \{\boldsymbol{h}(n)\}$, i.e., there is only one vector estimate $\boldsymbol{h}$. Therefore $F(P_{sn}) = \{\boldsymbol{f}\}$, $\boldsymbol{f} = \boldsymbol{g} + \boldsymbol{h}$.

2.3 Heuristic functions

The main goal of this paper is the comparative study of the performance of the general algorithm when instantiated with some heuristic functions. Three heuristic functions will be considered.

1. CREATE:
   - An acyclic search graph SG rooted in s.
   - List of alternatives, $OPEN$ = $\{(s, \boldsymbol{g}_s, F(s, \boldsymbol{g}_s))\}$.
   - Two empty sets, $GOALN$, $COSTS$.
2. CHECK TERMINATION. If $OPEN$ is empty, then backtrack in SG from the nodes in $GOALN$ and return the set of solution paths with costs in $COSTS$.
3. PATH SELECTION. Select an alternative $(n, \boldsymbol{g}_n, F)$ from OPEN. Delete $(n, \boldsymbol{g}_n, F)$ from $OPEN$, and move $\boldsymbol{g}_n$ from $G_{op}(n)$ to $G_{cl}(n)$.
4. SOLUTION RECORDING. If $n \in \Gamma$, then
   - Include n in $GOALN$ and $\boldsymbol{g}_n$ in $COSTS$.
   - Eliminate from $OPEN$ all alternatives (x, g_x, F_x) such that all vectors in F_x are dominated by $\boldsymbol{g}_n$ (FILTERING).
   - Go back to step 2
5. PATH EXPANSION: If $n \notin \Gamma$, then
   For all successors nodes m of n that do not produce cycles in SG do:
   (a) Calculate the cost of the new path found to m: $\boldsymbol{g}_m = \boldsymbol{g}_n + \boldsymbol{c}(n, m)$.
   (b) If m is a new node
      i. Calculate $F_m = F(m, \boldsymbol{g}_m)$ filtering estimates dominated by $COSTS$.
      ii. If F_m is not empty, put $(m, \boldsymbol{g}_m, F_m)$ in $OPEN$, and put $\boldsymbol{g}_m$ in $G_{op}(m)$ labelling a pointer to n.
      iii. Go to step 2.

   else (m is not a new node), in case
   - $\boldsymbol{g}_m \in G_{op}(m)$ or $\boldsymbol{g}_m \in G_{cl}(m)$: label with $\boldsymbol{g}_m$ a pointer to n, and go to step 2.
   - If $\boldsymbol{g}_m$ is non-dominated by any cost vectors in $G_{op}(m) \cup G_{cl}(m)$ (a path to m with new cost has been found), then :
      i. Eliminate from $G_{op}(m)$ and $G_{cl}(m)$ vectors dominated by $\boldsymbol{g}_m$
      ii. Calculate $F_m = F(m, \boldsymbol{g}_m)$ filtering estimates dominated by $COSTS$.
      iii. If F_m is not empty, put $(m, \boldsymbol{g}_m, F_m)$ in $OPEN$, and put $\boldsymbol{g}_m$ in $G_{op}(m)$ labelling a pointer to n.
      iv. Go to step 2.
   - Otherwise: go to step 2.

Table 1. A general multiobjective search algorithm.

When no heuristic information is available, the trivial heuristic function is given by $\boldsymbol{h}_0(n) = \mathbf{0}$, for all nodes n.

Tung and Chew [10] proposed two nontrivial heuristic functions that are well defined for every Bicriterion Path Problem. The first heuristic [1] will be called in this paper $\boldsymbol{h}_{12}(n) = (h_1(n), h_2(n))$ and is computed as follows: consider a graph G_i ($i = 1, 2$) with same nodes than the given graph G and arcs reversed. For each arc (n', n) in G_i a scalar cost is defined, given by the i-th component of the cost $\boldsymbol{c}(n, n')$ in G. Then apply Dijkstra's algorithm [14] to this graph to find the shortest path from a terminal node t to the usual starting node s, breaking the ties by a lexicographic order. In this way, we compute for each node n, a value $h_i(n)$, that is, the cost of the "best" path from t to n (i. e., from n to t), considering only the i-th component of the cost. It is obvious that

[1] Tung and Chew [10] call this heuristic q.

the vectorial heuristic $\boldsymbol{h}_{12}(n) = (h_1(n), h_2(n))$ is optimistic, i. e., $\boldsymbol{h}_{12}(n) \prec \boldsymbol{f}^*(n)$ for every node n.

The second heuristic [2] proposed by Tung and Chew will be called in this paper h_{mix} and is defined in a similar way, as the result from applying Dijkstra's algorithm to the graph resulting from reversing every arc, but the cost is now the sum of the components of the cost of the original graph, i. e., arc (n', n) is labeled with $c(n', n) = \sum_i c_i(n, n')$. Notice that h_{mix} is a scalar heuristic.

Algorithm	Selection rule	Additional selection rule	Filtering criteria
NAMOA-LEX-H0	best $\boldsymbol{g}$	lex order	$\boldsymbol{g}$ dominated
NAMOA-LEX	best $\boldsymbol{g} + \boldsymbol{h}_{12}$	lex order	$\boldsymbol{g} + \boldsymbol{h}_{12}$ dominated
TC-BS	best $\sum_i g_i$		$\boldsymbol{g} + \boldsymbol{h}_{12}$ dominated
TC-HS	best $h_{mix} + \sum_i g_i$		$\boldsymbol{g} + \boldsymbol{h}_{12}$ dominated

Table 2. Instantiations of the general algorithm.

2.4 Instantiation of the general algorithm

Heuristic functions $\boldsymbol{h}(n)$ do not appear explicitly in the pseudocode in table 1. However, there are four points where $\boldsymbol{h}(n)$ is implicitly invoked:

- Step 3, PATH SELECTION. Usually the node selected from *OPEN* is non-dominated according to certain heuristic function.
- Step 4, SOLUTION RECORDING (filtering). All alternatives $(x, \boldsymbol{g}_x, F_x)$, such that all vectors in F_x are dominated by the cost $\boldsymbol{g}_n$ of a newly found solution, are eliminated from *OPEN*. Since we are assuming that there is just an element $\boldsymbol{f}_x \in F_x$, $\boldsymbol{f}_x = \boldsymbol{g}_x + \boldsymbol{h}_x$, the heuristic function also plays a role at this step.
- Step 5(b).i and 5(b).iii.ii. These are two new occurrences of filtering when performing the step PATH EXPANSION. Filtering depends on the values of F_x and hence on the values of the heuristic function [3].

Algorithm NAMOA* [12] is just the algorithm of table 1 when the same heuristic function is applied both for path selection (step 3) and for filtering. Since h_{mix} is a scalar function, it could not be sensibly used for filtering. Therefore, considering the heuristic functions defined in subsection 2.3, there will be two instances of NAMOA: blind NAMOA* (with $\boldsymbol{h}_0$ for selecting and filtering) and informed NAMOA* (with $\boldsymbol{h}_{12}$ for selecting and filtering). In both cases, the selection procedure of step 3 could be stated as follows:

PATH SELECTION (version A). Select an alternative $(n, \boldsymbol{g}_n, \{\boldsymbol{f}\})$ from OPEN such that $\nexists (n', \boldsymbol{g}_{n'}, \{\boldsymbol{f}'\}) \in OPEN \quad | \quad \boldsymbol{f}' \prec \boldsymbol{f}$.

[2] Tung and Chew [10] call this heuristic with the somewhat confusing name h^*.

[3] Notice that the original *algorithm* proposed by Tung and Chew [10] does not perform all these filtering operations, or performs them in a different way

In other words, a nondominated $\boldsymbol{f}$ is selected from OPEN. This guarantees admissibility [12]. But this rule is not enough to determine the selected path; usually there are many nondominated $\boldsymbol{f}$ in OPEN. An additional rule is needed. The original description of NAMOA* [12] suggests LEX, the lexicographical order, as a suitable rule. Therefore, we consider two instantiations of the general algorithm with version A of path selection: NAMOA-LEX-H0 (blind NAMOA-LEX) and NAMOA-LEX (informed NAMOA-LEX).

If path selection is not dictated by the same heuristic function used for filtering, other possibilities appear. Tung and Chew [10] proposed the following selection rule that uses the h_{mix} heuristic (preserving $\boldsymbol{h}_{12}$ for filtering):

PATH SELECTION (version B). Select an alternative $(n, \boldsymbol{g}_n, \{\boldsymbol{f}\})$ from OPEN such that $\forall(n', \boldsymbol{g}_{n'}, \{\boldsymbol{f}'\}) \in$ OPEN, it holds that $h_{mix} + \sum_i g_i \leq h'_{mix} + \sum_i g'_i$.

The corresponding instance of the algorithm will be called TC-HS (for Tung & Chew with Heuristic Selection).

In order to evaluate the impact of the h_{mix} heuristic, we shall consider an additional blind selection rule:

PATH SELECTION (version C). Select an alternative $(n, \boldsymbol{g}_n, \{\boldsymbol{f}\})$ from OPEN such that $\forall(n', \boldsymbol{g}_{n'}, \{\boldsymbol{f}'\}) \in$ OPEN, it holds that $\sum_i g_i \leq \sum_i g'_i$.

It is easily proven that minimizing $\sum_i g_i$ yields a nondominated $\boldsymbol{f}$,(just consider that $\boldsymbol{h}_0(n)$ is used for selection). Since this rule is a "blind" version of that in Tung and Chew [10], the corresponding instance of the algorithm will be called TC-BS (for Tung & Chew with Blind Selection).

Table 2 sums up the features of the four algorithms studied in this paper.

3 Empirical evaluation

Bidimensional square grids without obstacles were used to test the algorithms. Biobjective cost vectors (c_1, c_2) were generated for each arc. The values c_1, c_2 were independently and randomly generated from a uniform distribution of integer values in the range $[1, 10]$. Ten different sets of grids with sizes from 10×10 to 100×100 were generated. The results presented for each size are averaged over the ten different test sets. Search was conducted in all of the problems from one corner of the grid to the opposite. Therefore, solution depth varies from 20 to 200 in steps of 20. The results discussed in the following describe the number of iterations, space, and time requirements of each algorithm as a function of grid size.

The tests were run on a Windows XP 32-bit platform, with an Intel Core2 Quad Q9550 at 2.8Ghz, and 4Gb of RAM. The algorithms were implemented to share as much code as possible, using the LispWorks Professional 5.01 programming environment. Nevertheless, important differences are needed. Both informed and blind NAMOA-LEX share all their code and only differ in the heuristic function provided. The G_{op} sets were ordered lexicographically. Both TC-HS and TC-BS share also all of their code. However, the G_{op} sets were ordered according to their respective linear evaluation functions. In all implementations, only the current best cost estimate of each node was kept in $OPEN$ at each iteration, as suggested in [12]. The $OPEN$ list was implemented as a binary heap.

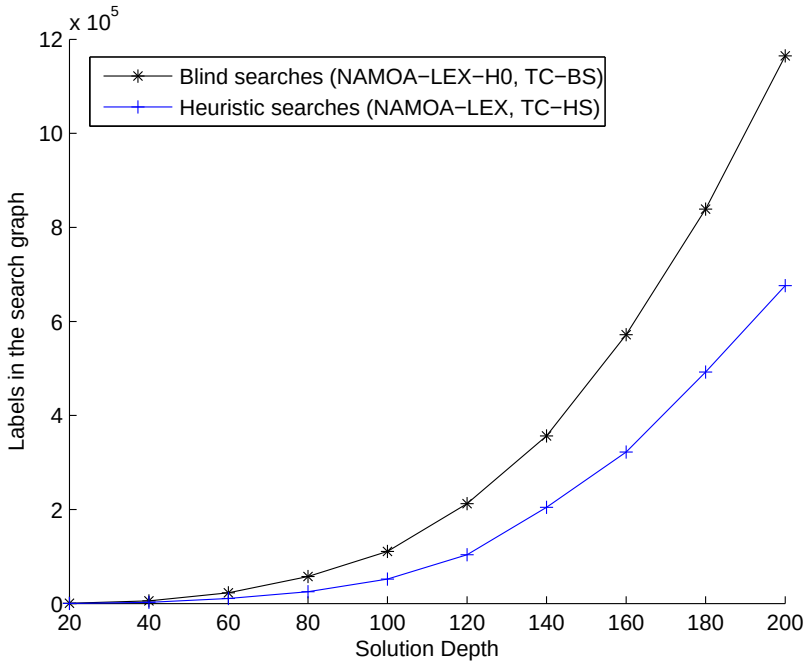

Fig. 1. Iterations/Space requirements at different solution depths, averaged over ten problem sets.

3.1 Number of iterations and space requirements

Most analysis of heuristic search algorithms concentrate on the number of iterations as a measure of their complexity. Figure 1 shows the number of iterations of the algorithms as a function of grid size. Both heuristic searches (NAOMA-LEX and TC-HS) obtained undistinguishable results. NAMOA-LEX and TC-HS exhibit an important reduction in the number of iterations (i.e labels considered) when compared with blind NAMOA-LEX-H0 and TC-BS. Again, blind searches (NAMOA-LEX-H0 and TC-BS) obtained indistinguishable results. The reduction in the number of iterations using heuristics is on average 45%. The number of labels stored in the search graph was found in all cases to equal the number of iterations. This is not surprising, since the label expanded by the algorithms at each iteration is permanently stored in the G_{cl} sets. Therefore, space requirements grow fast with solution depth. The number of nodes in each grid ($n \times n$) is comparatively much smaller than the total number of labels.

This is an important result for the heuristics under consideration, since most applications of bicriterion search described in the literature report difficulties with space requirements.

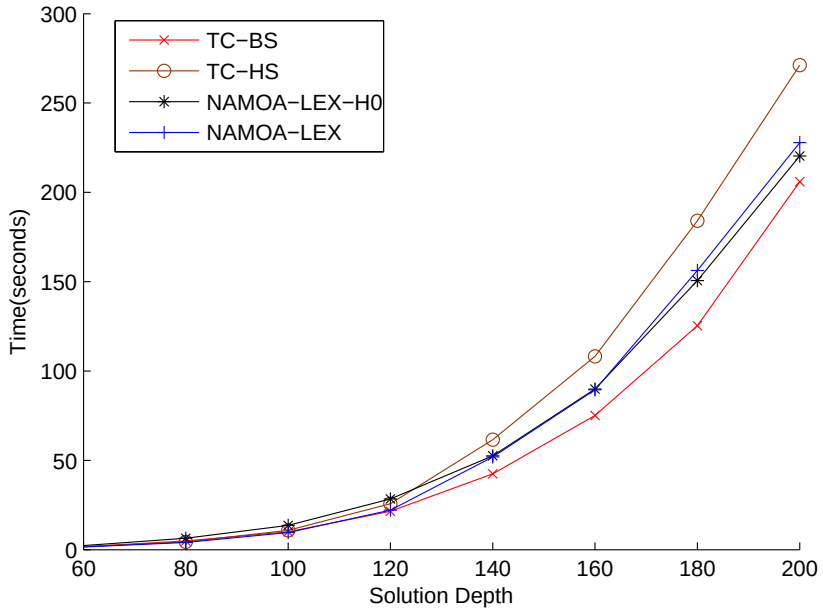

Fig. 2. Time requirements at different solution depths, averaged over ten problem sets.

3.2 Time requirements

A first conclusion from the experiments is that the time needed to calculate the heuristic values is not significant compared to total execution time (less than 5% on average, and less than 1% in the largest problems). This is due to the fact that the single objective search runs, used to precompute the heuristic values, have much smaller time requirements than bicriterion search.

Time requirements of all algorithms are shown in figure 2. Contrary to intuition, the experiments do not show the decrease in time that could be expected from the reduction in the number of iterations of the NAMOA-LEX and TC-HS algorithms. NAMOA-LEX has requirements similar to those of uninformed NAMOA-LEX-H0 in spite of the fact that the number of iterations is much smaller. Surprisingly, TC-BS takes significantly less time than TC-HS. In fact, TC-BS was the fastest algorithm and TC-HS the slowest. It is obvious from these results that search time cannot be easily extrapolated from the number of iterations. This is due to the fact that in bicriterion search, label expansion is by no means an atomic constant-time operation.

Bicriterion search algorithms share with scalar ones a variability in time per iteration due to the need to sort open alternatives. In bicriterion search the number of open alternatives is low at the beginning and end of the search, and high some time in between. Additionally, each new label selected for expansion generates a number of successor labels that need to be checked for dominance with the labels in the $G_{op}(n)$ and $G_{cl}(n)$

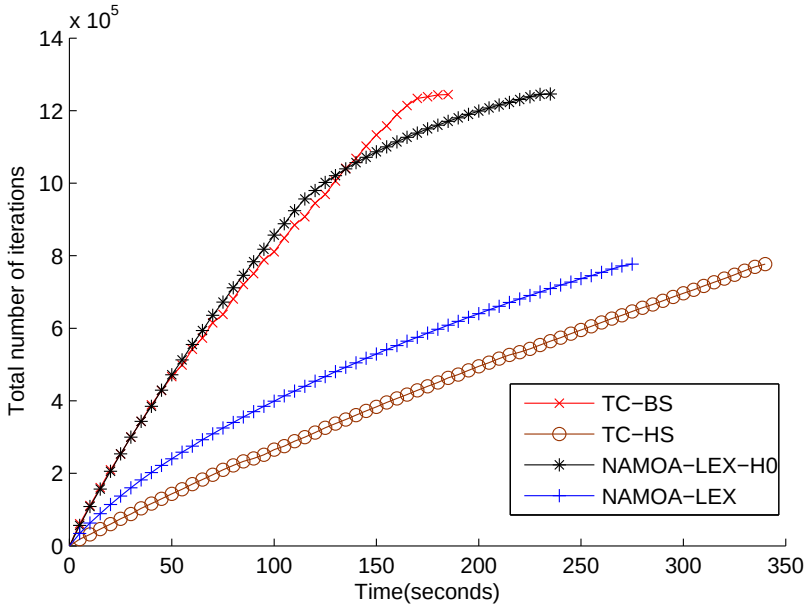

Fig. 3. Number of iterations vs. time for a particular search problem.

sets of the destination node n, as well as the set of nondominated labels found to the goal nodes (i.e. those kept in the $COSTS$ set). Particularly, the size of $COSTS$ can grow rapidly with solution depth. Therefore, the costly dominance checks used in the filtering steps Step 5(b).i and 5(b).iii.ii of the algorithm in table 1 seem to dominate time requirements.

Our analysis reveals that time per iteration is highly influenced by the number of solutions already found at each iteration (i.e. the size of the $COSTS$ set at that time). Figures 3 and 4 illustrate the behavior of the algorithms for one particular problem of size 100×100. Figure 3 shows that both TC-BS and NAMOA-LEX-H0 performed the same number of iterations in this problem. However, TC-BS performed a roughly constant number of iterations per second and abruptly slowed down at the end, while NAMOA-LEX-H0 slowed down more gradually and finally took more time to finish. Notice that TC-HS and NAMOA-LEX performed less iterations than the previous algorithms, but required more time per iteration from the beginning, specially TC-HS. The result is that these algorithms were slower while solving the same problem instance. Similar behaviour was found in all cases analyzed.

The explanation of this behavior can be found in figure 4. All algorithms found the same number of solutions. However, TC-BS found the solutions in the final seconds of search, coincidentally with the abrupt descent of iterations per second observed in figure 3. In a similar way, NAMOA-LEX-H0 found solutions more gradually but also in the final search stage, resulting in the second fastest alternative. NAMOA-LEX found

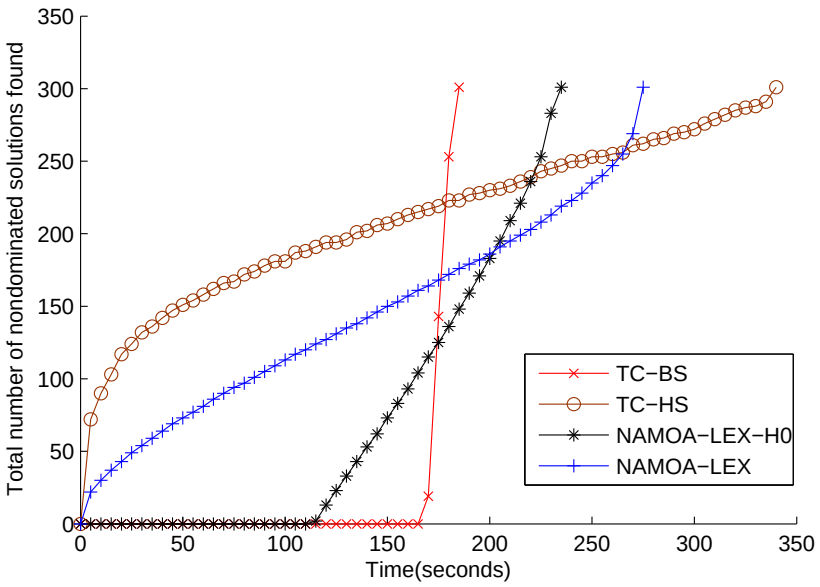

Fig. 4. Number of nondominated solutions found vs. time for a particular search problem.

solutions even faster and was therefore slower. Finally, TC-HS found solutions very quickly and was the slowest algorithm.

Figure 5 shows the time taken by each algorithm to find the *first* solution as a function of solution depth averaged for all problem sets. Again, TC-BS is the algorithm that starts to find solutions later, followed by NAMOA-LEX-H0. Both NAMOA-LEX and TC-HS find solutions very early and appear undistinguishable at this scale.

4 Conclusions and future work

The paper presents a general bicriterion search procedure that encompasses NAMOA* and Tung-Chew. This formal contribution highlights the differences between these algorithms and allows a clear comparison of different alternatives.

The effect of the heuristic functions proposed by Tung and Chew was evaluated empirically over four instances of the general search procedure (TC-HS, TC-BS, NAMOA-LEX, and NAMOA-LEX-H0). Several important conclusions can be drawn. In the first place, the experiments confirmed that the time needed to calculate the heuristics is not significant compared to total execution time. The heuristics were also found to improve the number of iterations, reducing the space requirements of the algorithms. The reduction of space requirements was similar in TC-HS (which used a combination of two heuristic functions) and NAMOA-LEX (which used only one of the heuristic functions).

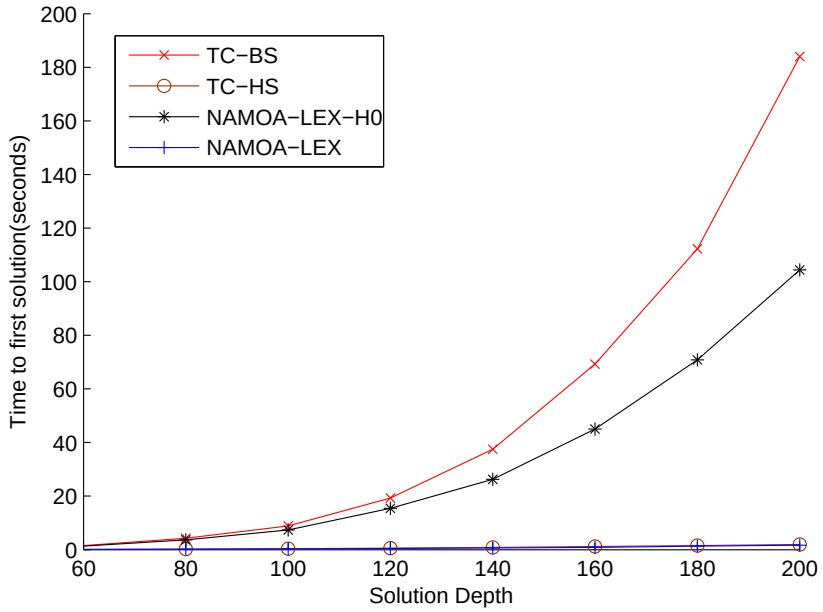

Fig. 5. Time requirements to reach the first solution averaged over all problem sets.

However, contrary to intuition, no time improvement could be observed in the algorithms with informed selection over the uninformed ones. The speed of the algorithms was found to be related to the discovery of nondominated solutions. Those algorithms that found solutions later in the search performed consistently faster. In this sense, the use of a specialized selection heuristic in TC-HS was found to work against the time efficiency of the algorithm.

The use of heuristics can effectively reduce the number of considered alternatives in bicriterion search algorithms. Most practical implementations of current algorithms perform linear or lexicographic orderings for label selection. The results presented in this paper suggest that the investigation of alternative orderings that combine heuristic search and delayed expansion of solutions could lead to more efficient algorithms.

References

1. Hart, P., Nilsson, N., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Systems Science and Cybernetics SSC-4 **2** (1968) 100–107
2. Pearl, J.: Heuristics. Addison-Wesley, Reading, Massachusetts (1984)
3. Dechter, R., Pearl, J.: Generalized best-first search strategies and the optimality of A*. Journal of the ACM **32**(3) (July 1985) 505–536
4. Clímaco, J.C.N., Craveirinha, J.M.F., Pascoal, M.M.B.: A bicriterion approach for routing problems in multimedia networks. Networks **41**(4) (2003) 206–220

5. Alechina, N., Logan, B.: State space search with prioritised soft constraints. Applied Intelligence **14**(3) (2001) 263–278
6. Gabrel, V., Vanderpooten, D.: Enumeration and interactive selection of efficient paths in a multiple criteria graph for scheduling an earth observing satellite. European Journal of Operational Research **139** (2002) 533–542
7. Refanidis, I., Vlahavas, I.: Multiobjective heuristic state-space search. Artificial Intelligence **145** (2003) 1–32
8. Hansen, P.: Bicriterion path problems. In: Lecture Notes in Economics and Mathematical Systems 177, Springer (1979) 109–127
9. Müller-Hannemann, M., Weihe, K.: On the cardinality of the pareto set in bicriteria shortest path problems. Annals OR **147**(1) (2006) 269–286
10. Tung, C.T., Chew, K.L.: A multicriteria pareto-optimal path algorithm. European Journal of Operational Research **62** (1992) 203–209
11. Stewart, B.S., White, C.C.: Multiobjective A*. JACM **38**(4) (1991) 775–814
12. Mandow, L., Pérez de la Cruz, J.L.: A new approach to multiobjective A* search. In: Proc. of the XIX Int. Joint Conf. on Artificial Intelligence (IJCAI'05). (2005) 218–223
13. Mandow, L., Pérez de la Cruz, J.L.: Comparison of heuristics in multiobjective A* search. In: CAEPIA'05 -LNAI 4177, Springer (2006) 180–189
14. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1** (1959) 269–271