The Thirteenth International Symposium
on Combinatorial Search (SoCS 2020)

A Simple and Fast Bi-Objective Search Algorithm (Extended Abstract)[*]

Carlos Hernández Ulloa,[†] William Yeoh,[‡] Jorge A. Baier,[*, §]
Han Zhang,[⋆] Luis Suazo,[†] Sven Koenig[⋆]
[†] Departamento de Ciencias de la Ingeniería, Universidad Andrés Bello, Chile
[‡] Department of Computer Science and Engineering, Washington University in St. Louis, USA
[*] Departamento de Ciencia de la Computación, Pontificia Universidad Católica de Chile, Chile
[§] Millennium Institute for Foundational Research on Data, Chile
[⋆] Department of Computer Science, University of Southern California, USA

Abstract

Many interesting search problems can be formulated as bi-objective search problems, that is, search problems where two kinds of costs have to be minimized. We describe our new Bi-Objective A* (BOA*) algorithm and show that it can run an order of magnitude (or more) faster than state-of-the-art bi-objective search algorithms. See our longer ICAPS paper (Hernández et al. 2020) for details.

Introduction

Researchers have extended A* to solve bi-objective shortest path problems where one wants to find the set of Pareto-optimal paths from the start state to the goal state. Two such state-of-the-art A* extensions are NAMOA* (Mandow and Pérez-de-la-Cruz 2010) and its improvement NAMOA*dr (Pulido, Mandow, and Pérez-de-la-Cruz 2015). Upon generating any node, these search algorithms have to check if the newly found path to some state s is dominated by a previously found path to s and, if so, discard the newly found path. They also need to check whether a previously found path to s is dominated by the newly found path to s and, if so, discard the previously found path. In this context, path p dominates path p' iff each kind of path cost of p is no larger than the corresponding kind of path cost of p' and at least one kind of path cost of p is smaller than the corresponding kind of path cost of p'. The set of Pareto-optimal paths is the set of paths that are not dominated by any path.

In this paper, we introduce our *Bi-Objective A** (BOA*) algorithm that prunes dominated paths more efficiently than NAMOA* and NAMOA*dr by exploiting that the heuristic function is consistent. It performs all dominance checks in constant time.

Our Bi-Objective A* (BOA*) Algorithm

A bi-objective *search graph* is a tuple $(S, E, \mathbf{c})$, where S is the finite set of *states*, $E \subseteq S \times S$ is the finite set of edges, and $\mathbf{c}$ is a *cost function* that associates a pair of non-negative real costs with each edge. $Succ(s) = \{t \in S \mid (s,t) \in E)$ denotes the successors of state s. A bi-objective *search problem instance* is a tuple $P = (S, E, \mathbf{c}, s_{start}, s_{goal})$, where $(S, E, \mathbf{c})$ is a search graph, $s_{start} \in S$ is the start state, and $s_{goal} \in S$ is the goal state. A *path* from s_1 to s_n is a sequence of states $s_1, s_2, \ldots, s_n$ such that $(s_i, s_{i+1}) \in E$ for all $i \in \{1, \ldots, n-1\}$. A path is a *solution* for instance P iff it is a path from s_{start} to s_{goal}. If $\mathbf{p} = (p_1, p_2)$ and $\mathbf{q} = (q_1, q_2)$, $\mathbf{p} \prec \mathbf{q}$ denotes that ($p_1 < q_1$ and $p_2 \leq q_2$) or ($p_1 = q_1$ and $p_2 < q_2$). In this case, we say that $\mathbf{p}$ *dominates* $\mathbf{q}$. $\mathbf{c}(\pi) = \sum_{i=1}^{n-1} \mathbf{c}(s_i, s_{i+1})$ is the cost of path $\pi = s_1, \ldots, s_n$. $\pi \prec \pi'$ for two paths π and π' denotes that $\mathbf{c}(\pi) \prec \mathbf{c}(\pi')$. In this case, we say that π *dominates* π'.

Given an instance P, a *Pareto-optimal solution* π for P is a solution for P such that $\pi' \not\prec \pi$ for all solutions π' for P, that is, a Pareto-optimal solution is not dominated by any solution. The Pareto-optimal solution set is the set of all Pareto-optimal solutions. We are interested in finding a cost-unique Pareto-optimal solution set, which is any maximal subset of the Pareto-optimal solution set such that any two solutions in the subset do not have the same cost.

The $Open$ list of BOA* contains *nodes*. Each node x has a state $s(x)$, a g-value $\mathbf{g}(x)$, an f-value $\mathbf{f}(x)$, and a parent $parent(x)$ and corresponds to a path to $s(x)$ of cost $\mathbf{g}(x)$.

Algorithm 1 shows the pseudocode of BOA*. It takes as input a bi-objective search problem and a consistent heuristic function and computes a cost-unique Pareto-optimal solution set by maintaining a $g_2^{\min}$-value for every state. In each iteration, it extracts a node x with g-value $\mathbf{g}(x) = (g_1, g_2)$ from the $Open$ list with the lexicographically smallest f-value $\mathbf{f}(x) = (f_1, f_2)$ of all nodes in the $Open$ list (Line 9). It does not expand the node if its g_2-value is at least $g_2^{\min}(s(x))$ or its f_2-value is at least $g_2^{\min}(s_{goal})$ (Line 10). Otherwise, it updates $g_2^{\min}(s(x))$ (Line 11) and expands the node. If $s(x)$ is the goal state, then BOA* has found an undominated solution and adds node x to the solution set $sols$ (Lines 12-14). Otherwise, it calculates the child nodes of node x (Lines 15-21). It does not add a child node y to the $Open$ list if its g_2-value is at least $g_2^{\min}(s(y))$ or its f_2-value is at least $g_2^{\min}(s_{goal})$ (Line 20). Otherwise, it generates the child node by adding it to the $Open$ list (Line 21). It terminates when the $Open$ list is empty and returns the solution

[*]The research was partially supported by the National Science Foundation under grant numbers 1409987, 1724392, 1812619, 1817189, 1837779, 1838364, and 1935712 as well as by the Department of Computer Science at PUC Chile under a FondDCC grant.

Algorithm 1: Bi-Objective A* (BOA*)

```
   Input  : A search problem $(S, E, \mathbf{c}, s_{start}, s_{goal})$ and a consistent
            heuristic function $\mathbf{h}$
   Output: A cost-unique Pareto-optimal solution set
1  $sols \leftarrow \emptyset$
2  for each $s \in S$ do $g_2^{\min}(s) \leftarrow \infty$
3  $x \leftarrow$ new node with $s(x) = s_{start}$
4  $\mathbf{g}(x) \leftarrow (0, 0)$
5  $parent(x) \leftarrow$ null
6  $\mathbf{f}(x) \leftarrow (h_1(s_{start}), h_2(s_{start}))$
7  Initialize $Open$ and add $x$ to it
8  while $Open \neq \emptyset$ do
9      Remove a node $x$ from $Open$ with the lexicographically smallest
         $f$-value of all nodes in $Open$
10     if $g_2(x) \geq g_2^{\min}(s(x)) \lor f_2(x) \geq g_2^{\min}(s_{goal})$ then continue
11     $g_2^{\min}(s(x)) \leftarrow g_2(x)$
12     if $s(x) = s_{goal}$ then
13         Add $x$ to $sols$
14         continue
15     for each $t \in \texttt{Succ}(s(x))$ do
16         $y \leftarrow$ new node with $s(y) = t$
17         $\mathbf{g}(y) \leftarrow \mathbf{g}(x) + \mathbf{c}(s(x), t)$
18         $parent(y) \leftarrow x$
19         $\mathbf{f}(y) \leftarrow \mathbf{g}(y) + \mathbf{h}(t)$
20         if $g_2(y) \geq g_2^{\min}(t) \lor f_2(y) \geq g_2^{\min}(s_{goal})$ then continue
21         Add $y$ to $Open$
22 return $sols$
```

New York City (NY)				
264,346 states, 730,100 edges, $\lvert sols \rvert$ = 199 on average				
	Solved	Average	Max	Min
sBOA*	50/50	9.75	148.65	0.10
NAMOA*dr	50/50	0.65	4.99	0.11
BOA*	50/50	**0.32**	**1.95**	0.11
BBDijkstra	50/50	1.94	23.43	0.26
BDijkstra	50/50	2.55	21.16	0.17

Colorado (COL)				
435,666 states, 1,042,400 edges, $\lvert sols \rvert$ = 427 on average				
	Solved	Average	Max	Min
sBOA*	50/50	38.88	1,141.78	0.17
NAMOA*dr	50/50	2.16	57.40	0.17
BOA*	50/50	**0.79**	**15.26**	0.17
BBDijkstra	50/50	4.79	83.07	0.41
BDijkstra	50/50	7.78	135.24	0.29

Florida (FL)				
1,070,376 states, 2,712,798 edges, $\lvert sols \rvert$ = 739 on average				
	Solved	Average	Max	Min
sBOA*	46/50	349.64	1,238.25	**0.43**
NAMOA*dr	50/50	19.66	329.79	**0.43**
BOA*	50/50	**4.59**	**60.54**	**0.43**
BBDijkstra	50/50	91.36	1,772.48	1.11
BDijkstra	50/50	158.33	2,722.69	0.77

Table 1: Runtime (in seconds) on 50 instances of the specified road map. When an algorithm times out after 3,600 seconds, we use 3,600 seconds in the calculation of the average.

set (Line 22).

Experimental Results

We compare BOA*, NAMOA*dr, BOA* with standard linear-time dominance checking (sBOA*), Bi-Objective Dijkstra (BDijkstra), and Bidirectional Bi-Objective Dijkstra (BBDijkstra) (Sedeño-Noda and Colebrook 2019). We use the C implementations provided by the authors for BBDijkstra and BDijkstra. We implement BOA*, sBOA*, and NAMOA*dr from scratch in C using a standard binary heap for the $Open$ list. We run all experiments on a 2.20GHz Intel(R) Xeon(R) CPU Linux machine with 128GB of RAM. We use road maps from the 9th DIMACS Implementation Challenge: Shortest Path.[1] The cost components represent travel distances (c_1) and times (c_2). The h-values are the exact travel distances and times to the goal state, computed with Dijkstra's algorithm. It takes 75 milliseconds to compute the h-values for the largest road map. The reported runtimes include this computation. All algorithms obtain the same number of solutions for all instances used in the experiments, implying that no two Pareto-optimal solutions have the same cost.

We compare the runtimes of the five algorithms on 50 instances each of 3 USA road maps used by Machuca and Mandow (2012). Table 1 shows the name of the road map, the number of states and edges of the map, and the average number of Pareto-optimal solutions. For each algorithm, it shows the number of instances solved within a runtime limit of 3,600 seconds as well as the average, maximum, and minimum runtimes (in seconds). We observe that NAMOA*dr can be an order-of-magnitude faster than sBOA*. BOA* can be several times faster than NAMOA*dr, especially on instances with large numbers of Pareto-optimal solutions. For example, BOA* is 4.3 times faster than NAMOA*dr on FL (with 739 Pareto-optimal solutions on average), while BOA* is only 2.03 times faster than NAMOA*dr on NY (with 199 Pareto-optimal solutions on average). BOA* can also be an order-of-magnitude faster than BBDijkstra and BDijkstra.

[1] http://users.diag.uniroma1.it/challenge9/download.shtml

Conclusions

We have presented Bi-Objective A* (BOA*), a simple and fast best-first bi-objective search algorithm. BOA* improves the efficiency of the dominance checks substantially, which is key to improving the efficiency of the search. The dominance checks of BOA* require only constant time, while the ones of existing bi- and multi-objective search algorithms require linear time. Consequently, BOA* can run faster than state-of-the-art algorithms.

References

Hernández, C.; Yeoh, W.; Baier, J. A.; Zhang, H.; Suazo, L.; and Koenig, S. 2020. A simple and fast bi-objective search algorithm. In *ICAPS*.

Machuca, E., and Mandow, L. 2012. Multiobjective heuristic search in road maps. *Expert Systems with Applications* 39(7):6435–6445.

Mandow, L., and Pérez-de-la-Cruz, J. 2010. Multiobjective A* search with consistent heuristics. *Journal of the ACM* 57(5):27:1–27:25.

Pulido, F.-J.; Mandow, L.; and Pérez-de-la-Cruz, J.-L. 2015. Dimensionality reduction in multiobjective shortest path search. *Computers & Operations Research* 64:60–70.

Sedeño-Noda, A., and Colebrook, M. 2019. A biobjective Dijkstra algorithm. *European Journal of Operational Research* 276(1):106–118.