Proceedings of the Ninth International
Symposium on Combinatorial Search (SoCS 2016)

Searching for Real-Time Heuristic Search Algorithms

Vadim Bulitko
Department of Computing Science
University of Alberta
Edmonton, Alberta, T6G 2E8, Canada
bulitko@ualberta.ca

Abstract

Heuristic search is a core area of Artificial Intelligence with applications to planning, scheduling and game playing. Real-time heuristic search applies to search problems where plan execution needs to start before a complete solution can be computed. Since the inception of real-time heuristic search in the early 1990s a great number of algorithms have been proposed and evaluated. In this paper we break them down into building blocks and conduct a search in the space of such building blocks. Even simple tabulated and iterative searches find new real-time heuristic search algorithms outperforming manually crafted contemporary algorithms.

1 Introduction and Related Work

Starting with LRTA* (Korf 1990) real-time heuristic search agents interleave three processes: local planning, heuristic learning and move selection. In over the two decades since LRTA*, researchers have explored different heuristic learning rules and different move selection mechanisms. Information in addition to the heuristic has also been learned.

The number of techniques proposed by the researchers in the field of real-time heuristic search is overwhelming. More importantly, the interactions between these techniques are difficult to analyze theoretically or empirically. Our recent conference paper (Bulitko 2016) framed the problem of finding a high-performance combination of real-time heuristic search techniques as a search task itself. Unlike human researchers, such search has no prior expectations, intuitions or biases. It simply traverses a large space of real-time heuristic search algorithms. Yet, preliminary results in the standard testbed of pathfinding on video-game maps were promising. A simple search on a desktop computer conducted in under a day led to a new real-time heuristic search algorithm that outperformed manually crafted algorithms.

Here we explore the space of algorithms and discover that many randomly chosen algorithms appear to outperform manually designed published algorithms. Thus, it is no surprise that even a simple search produced a well-performing algorithm. We confirm the results using a validation set of problems as well as a slightly different space of algorithms.

2 Problem Formulation and Empirical Setup

We follow the problem formulation and the empirical setup of Bulitko (2016). An agent is to find a shortest path from a start state/vertex to a goal state/vertex on a weighted undirected safely explorable graph. For any state under consideration it uses and updates the heuristic – an estimate of the remaining travel cost. The suboptimality of the found path (capped at a pre-defined maximum) is the objective function – values closer to 1 (optimal) are preferred. We evaluate algorithms' suboptimality empirically by averaging it over subsets of the 493298 video-game pathfinding problems situated on 342 maps (Sturtevant 2012).

3 Space of Search Algorithms

A real-time heuristic search agent interleaves moving to the next state and updating its heuristic (learning). The two steps can be implemented with a number of rules, called *building blocks*. In this paper we re-use the blocks from our previous publication (Bulitko 2016). We modify one of the blocks (heuristic weighting) by adding a second weight to c and remove backtracking as it tends to trade first-trial solution cost for convergence. We denote any resulting algorithm by listing its building blocks as $w \cdot \text{lop}_b(w_c \cdot c + h)$+da+E where the last two parts (da and E) are optional. Based on exploratory experiments, we set the ranges for the control parameters as $w \in [1, 10], w_c \in [1, 10], \texttt{da} \in \{true, false\}, \texttt{expendable} \in \{true, false\}, \text{lop} \in \{\min, \text{avg}, \text{median}, \max\}, b \in [0, 1]$ which define a continuous six-dimensional space of real-time heuristic search algorithms.

To see how algorithms in the space are distributed in terms of their performance we generated 36400 random algorithms in the space and ran each on a random subset of 5000 problems. Maximum suboptimality cutoff was set to 1000. The average of the sample suboptimalities (50.6) is an estimate of the expected performance of an algorithm randomly sampled from the algorithm space. Remarkably, it is substantially better than sample suboptimality of RTA* (Korf 1990) (186.7) but worse than that of the more recent daLRTA*+E (Hernández and Baier 2012; Sharon, Sturtevant, and Felner 2013) (30.5). Note that 73.5% of algorithms in the database have better sample suboptimality than the expectation for a randomly drawn algorithm (due to the long tail of the distribution). The classic RTA* is in the bottom

5.5% and even the contemporary daLRTA*+E is outperformed by 55% of all database algorithms.

4 Search in the Space of Search Algorithms

In this section we use two simple search techniques in the space of real-time search algorithms. First we use an uninformed search we call *tabulated*. Second, we implement *iterative* search where previous iterations inform the current iteration. We used both techniques in our previous publication (Bulitko 2016). In this paper we extend that work by (i) applying both techniques to a new space of algorithms, (ii) validating the results by using a different set of problems.

Tabulated Search. We first tabulated the parameter space with 7 equally spaces tabulation points: $w \in \{1, 2.5, 4, 5.5, 7, 8.5, 10\}$, $w_c \in \{1, 2.5, 4, 5.5, 7, 8.5, 10\}$, $\texttt{da} \in \{true, false\}$, $\texttt{expendable} \in \{true, false\}$, $\text{lop} \in \{\min, \text{avg}, \text{median}, \max\}$, $b \in \{0, 0.17, 0.33, 0.5, 0.67, 0.83, 1\}$. This resulted in 5488 algorithms which we ran on a single random set of 500 problems. The algorithm with the lowest sample suboptimality of 16.04 was $1 \cdot \text{median}_1(7 \cdot c + h) + E$, found in 9.4 hours. We then ran it on the entire benchmark set of 493298 MovingAI problems, against the classic LRTA*, RTA* (Korf 1990) and its weighted variant wLRTA* (Rivera, Baier, and Hernández 2015). We also considered existing algorithms that explicitly aim to escape heuristic depressions quicker than LRTA*: aLRTA*, daLRTA* (Hernández and Baier 2012), its weighted version wdaLRTA* (Rivera, Baier, and Hernández 2015) and its combination with removing expendable states, daLRTA*+E (Sharon, Sturtevant, and Felner 2013). We did not include f-LRTA* or f-LRTA*+E in our evaluation since both versions were found inferior to daLRTA* by Sharon, Sturtevant, and Felner (2013). All algorithm parameters have been tuned by running different parameter values on problems from the benchmark set (Bulitko 2016). The lookahead was set to 1 in all algorithms. The mean solution suboptimalities were as follows: LRTA* 459.8, RTA* 358.9, daLRTA* 44.1, wLRTA* 39.7, wdaLRTA* 30.6, daLRTA*+E 31.8 and $\text{median}(7 \cdot c + h)$+E 19.3.

The results show that the search algorithm found by the tabulated search process substantially outperforms parameter-tuned manually designed algorithms, both classic and contemporary. We repeated tabulated search two more times with different 500 problem samples and while different algorithms were found, they had similar suboptimality when evaluated on the entire problem set and thus soundly outperformed the manually designed competitors. But perhaps tabulated search overfits its output to the specific problems used in tabulated search? We created an equally sized validation set of problems distributed in the same proportions over the same 342 maps. Each problem was generated randomly (as opposed to being downloaded from the MovingAI website). On the validation set the algorithms ranked in the same order.

Iterative Search. The tabulated search covers the space of algorithms in a uniform, grid-like fashion. An alternative is to conduct beam-like iterative search where each iteration's results define the algorithms used for the next iteration. As before (Bulitko 2016), we begin with an initial set of randomly chosen L algorithms. At each iteration step there are L algorithms whose performance is evaluated by running on a random sample of n problems picked from the benchmark set. The algorithms are then sorted by their sample suboptimality. The bottom $L/2$ algorithms are discarded (die) whereas the top $L/2$ not only make it to the next iteration (survive) but are also used to create the other $L/2$ algorithms for the next iteration (reproduction). Specifically, $L/2$ times we randomly draw a pair of the parents and randomly combine (crossover) their algorithmic descriptions: the vectors $(w, w_c, \texttt{da}, \texttt{expendable}, \text{lop}, b)$ (genes). The resulting vector is then randomly perturbed (mutated). The process runs for M steps. The output is the algorithm with the lowest sample suboptimality.

Such iterative search explores the space of algorithms non-uniformly, focusing on the more promising areas. Furthermore, since later iterations evaluate better algorithms, the overall time is reduced since better algorithms solve problems faster. On the other hand, early iterations may divert the search to a less-promising area of the space. We confirmed our earlier results (Bulitko 2016) and were able to find algorithms with suboptimality similar to that of the tabulated search. For instance, iterative search with $L = 50$ algorithms in the population and $M = 50$ steps (sample size 500, suboptimality cutoff 1000) took only 2.3 hours. The agent with the lowest sample suboptimality was $4.774 \cdot \min_{0.321}(4.342 \cdot c + h)$+E. When evaluated on all 493298 benchmark problems in $\mathcal{S}_{493298}$ the agent had suboptimality of 20.08 ± 0.0653. This is very similar to the performance of the algorithms found by the tabulated search and substantially better than manually designed algorithms.

Another run of the iterative search with the same parameters found a different algorithm ($9.813 \cdot \max_{0.266}(9.250 \cdot c + h)$+da+E) but with similar suboptimality: 20.54 ± 0.0753. Evaluating the found algorithms on the validation set of problems gave very similar suboptimality values, just like it did for the algorithms found by the tabulated search.

5 Conclusions & Future Work

We confirmed earlier results that a simple search in the space of real-time heuristic search algorithms can produce high-performance algorithms. Future work will incorporate larger local search spaces and select algorithms per problem.

References

Bulitko, V. 2016. Evolving real-time heuristic search algorithms. In *ALIFEXV*, (in press).

Hernández, C., and Baier, J. A. 2012. Avoiding and escaping depressions in real-time heuristic search. *JAIR* 43:523–570.

Korf, R. 1990. Real-time heuristic search. *AI* 42(2–3):189–211.

Rivera, N.; Baier, J. A.; and Hernández, C. 2015. Incorporating weights into real-time heuristic search. *AI* 225:1–23.

Sharon, G.; Sturtevant, N. R.; and Felner, A. 2013. Online detection of dead states in real-time agent-centered search. In *SoCS*, 167–174.

Sturtevant, N. R. 2012. Benchmarks for grid-based pathfinding. *IEEE TCIAIG* 4(2):144 – 148.