Proceedings, The Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-16)

Per-Map Algorithm Selection in Real-Time Heuristic Search

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

Abstract

Real-time heuristic search is suitable for time-sensitive pathfinding and planning tasks when an AI-controlled non-playable character must interleave its planning and plan execution. Since its inception in the early 90s, numerous real-time heuristic search algorithms have been proposed. Many of the algorithms also have control parameters leaving a practitioner with a bewildering array of choices. Recent work treated the task of algorithm and parameter selection as a search problem in itself. Such automatically found algorithms outperformed previously known manually designed algorithms on the standard video-game pathfinding benchmarks. In this paper we follow up by selecting an algorithm and parameters automatically per map. Our sampling-based approach is efficient on the standard video-game pathfinding benchmarks. We also apply the approach to per-problem algorithm selection and while it is effective there as well, it is not practical. We offer suggestions on making it so.

1 Introduction and Related Work

Heuristic search is a core area of Artificial Intelligence. In it an autonomous agent finds a path on the search graph connecting the start and the goal states. The agent is guided by a heuristic function (i.e., an estimate of the remaining path cost) and lower-cost paths are preferred. The classic A* algorithm (Hart, Nilsson, and Raphael 1968) and its more recent versions (Sturtevant and Buro 2005) have been used extensively for video-game pathfinding (Sturtevant 2007) and planning (Orkin 2006).

Real-time heuristic search is a form of agent-centered search (Koenig 2001) in which the amount of planning per agent's move is upper bounded independently of the number of states in the search space (Korf 1990). The upper bound is usually too low for the agent to compute a complete solution to the problem, forcing the agent to interleave planning and plan execution. As a result, suboptimal/wrong actions can be taken by the agent. To facilitate ever finding the goal state, the agent learns about the search space by updating its heuristic. Thus most real heuristic search algorithms interleave three processes: local planning, heuristic learning and move selection.

In over the two decades since LRTA*, researchers have explored different methods for looking ahead during the planning stage of each cycle (Koenig and Sun 2009); different heuristic learning rules (Bulitko 2004; Hernández and Meseguer 2005; Bulitko and Lee 2006; Rayner et al. 2007; Koenig and Sun 2009; Rivera, Baier, and Hernández 2015) and different move selection mechanisms (Ishida 1992; Shue and Zamani 1993a; 1993b; Shue, Li, and Zamani 2001; Hernández and Baier 2012). Finally, information in addition to the heuristic has been learned during (Bulitko et al. 2007; Sturtevant, Bulitko, and Björnsson 2010; Sturtevant and Bulitko 2011; Sharon, Sturtevant, and Felner 2013) and before (Bulitko et al. 2008; Bulitko, Björnsson, and Lawrence 2010; Botea 2011; Lawrence and Bulitko 2013) the search.

The proliferation of real-time search techniques proposed by the researchers can be overwhelming to a practitioner, especially due to substantial interaction between the techniques. Indeed, each technique is usually evaluated in its own empirical testbed and, at best, compared to a handful of others that the particular researcher happened to have implemented in the testbed. Broad comparisons are rare (Bulitko and Lee 2006). The interaction of the techniques is also difficult to predict theoretically as seemingly minor changes to the algorithms can have significant effects on their behavior (Sturtevant and Bulitko 2014; 2016).

Recent work by Bulitko (2016a) and Bulitko (2016b) cast the problem of selecting the best combination of real-time heuristic search techniques as a search problem in itself. This approach removes some of the expectations and biases of human researchers by conducting a search in the space of real-time heuristic search algorithms. This also allows one to compare a broad set of algorithms in a single experimental testbed such as the MovingAI video-game pathfinding problem set which is gradually becoming the *de facto* standard in the field (Sturtevant 2012).

Searching for real-time search algorithms quickly found novel combinations of the search techniques that significantly outperformed previously known human-designed algorithms. However, it also hit a performance ceiling where different ways of exploring the algorithm space all found algorithms with similar performance. It was then suggested that further progress can be made by selecting algorithms for each problem individually (Bulitko 2016b). The contribution of this paper is a simple selection algorithm approach

which we first evaluate on a per-map basis and then move onto the per-problem case. It is found effective in both cases although practical only in the per-map case.

The rest of the paper is organized as follows. We formulate the real-time heuristic search problem in Section 2. We then present the parameterized search algorithm in Section 3. The MovingAI empirical testbed is described in Section 4. We discuss the space of algorithms in Section 5 and present and evaluate our per-map algorithm selection approach in Section 6.1. We then apply it to per-problem selection in Section 6.2. Current limitations and future work are found in Section 7 followed by conclusions.

2 Problem Formulation

We re-use the problem formulation of Bulitko (2016a) and Bulitko (2016b) and reproduce it below for the reader's convenience. A *search problem* is a tuple (S, E, c, s_0, s_g, h) where S is a finite set of *states* and $E \subset S \times S$ is a set of *edges* between them. S and E jointly define the search graph. The search graph is stationary, undirected and connected (and thus safely explorable). No state has an edge leading to itself. Each edge $(s_a, s_b) \in E$ is weighted by the *cost* $c(s_a, s_b) = c(s_b, s_a) > 0$. The agent begins in the start state s_0 and changes its current state by traversing edges (i.e., taking actions). The cumulative cost of all edges it traverses prior to reaching the goal state s_g is the *solution cost*. The *suboptimality* is the ratio of the solution cost produced by an agent to the cost of the shortest possible path, $h^*(s_0)$. Lower values are desired; 1 indicates optimality.

The agent has access to a heuristic h which is an estimate of the remaining cost to the goal. We do not assume the heuristic to be admissible or consistent. The agent is free to update it in any way as long as $h(s_g) = 0$. The heuristic at time t is denoted by h_t; the initial heuristic $h_0 = h$ is included in the problem description.

The objective is to find a real-time heuristic search algorithm with the lowest expected suboptimality. The expectation is empirically approximated by *sample suboptimality*: running an algorithm on a set of benchmark problems and averaging suboptimality of the solutions. Per-problem suboptimality is capped at $\alpha_{\max} \geq 1$ plus possibly an edge cost.

3 Parameterized Real-time Heuristic Search

We use the parameterized real-time heuristic search algorithm of Bulitko (2016b) which operates as follows (Algorithm 1). As long as the goal state is not reached (line 3) the agent parameterized by control parameters w, w_c, b, lop, da, expendable and denoted by $w \cdot \text{lop}_b(w_c \cdot c + h)$+da+E goes through these steps.*

If the depression avoidance (Hernández and Baier 2012) block is included ($\texttt{da} = true$) then line 5 temporarily sets the agent's neighborhood to only the states where the amount of learning $|h_t(s) - h_0(s)|$ is minimal.

Next, in line 6, the learning rule covers heuristic weighting (Rivera, Baier, and Hernández 2015; Bulitko 2016b), learning operator and lateral learning (Bulitko 2016a), using the control parameters w, w_c, lop and b. The learning operator lop can be the traditional $\min$ or the more recently used $\max, \text{avg}, \text{median}$. The lateral learning defines the agent's neighborhood N_b^f as the b fraction of the neighborhood $N(s_t)$ with the lowest f values:

$$N_b^f(s) = \left(s^1, \ldots, s^{\lfloor b|N(s_t)| \rfloor}\right) \quad (1)$$

where $\left(s^1, \ldots, s^{\lfloor b|N(s_t)| \rfloor}, \ldots, s^{|N(s_t)|}\right)$ is the immediate neighborhood sorted in the ascending order by their $f = c + h$. For instance, s^1 has the lowest $f(s^1) = c(s_t, s^1) + h(s^1)$ value in the set $\{f(s) \mid s \in N(s_t)\}$ whereas $s^{|N(s_t)|}$ has the highest f value in that set. Clearly, for $b = 1$ we get the full neighborhood: $N_1^f(s_t) = N(s_t)$. For $b = 0$ we define $N_0^f(s_t)$ as the single neighbor with the lowest f: $\left\{s^1\right\}$.

If the expendable state detection (Sharon, Sturtevant, and Felner 2013) block is included ($\texttt{expendable} = true$) then in line 8 the current state is removed from the graph if there was learning in it and it is indeed locally expendable (i.e., a state whose immediate neighbors can be all reached from each other within the immediate neighborhood; denoted by the predicate $\mathcal{E}$). Then the agent moves to the next state in line 9. †

Algorithm 1: Parameterized Real-time Heuristic Search

input : search problem (S, E, c, s_0, s_g, h), control parameters w, w_c, b, lop, da, expendable
output: solution $(s_0, s_1, \ldots, s_g)$

1 $t \leftarrow 0$
2 $h_t \leftarrow h$
3 **while** $s_t \neq s_g$ **do**
4 **if** da **then**
5 $N(s_t) \leftarrow N_{\text{min learning}}(s_t)$
6 $h_{t+1}(s_t) \leftarrow \max\left\{h_t(s_t),\ w \cdot \underset{s \in N_b^f(s_t)}{\text{lop}} (w_c \cdot c(s_t, s) + h_t(s))\right\}$
7 **if** expendable & $h_{t+1}(s_t) > h_t(s_t)$ & $\mathcal{E}(s_t)$ **then**
8 remove s_t from the search graph
9 $s_{t+1} \leftarrow \arg\min_{s \in N(s_t)} (c(s_t, s) + h_t(s))$
10 $t \leftarrow t + 1$
11 $T \leftarrow t$

4 Empirical Testbed

We use the same empirical testbed as Bulitko (2016a) and Bulitko (2016b). The search problems came from the Moving AI benchmark set (Sturtevant 2012). The search graph is an 8-connected two-dimensional grid. The cardinal moves

*The search is terminated as soon as the agent's path becomes longer than $\alpha_{\max} \cdot h^*(s_0)$ which is then assumed to be the cost of the agent's solution.

†If at any time the neighborhood becomes empty and the agent has no moves to pick from then the agent quits without producing a solution. Such unsolved problems contribute $\alpha_{\max} \cdot h^*(s_0)$ plus the cost of the least expensive move ($\min_{s_a, s_b \in S} c(s_a, s_b)$) to the agent's statistics on the solution cost.

had the cost of 1, diagonal moves cost $\sqrt{2}$. The initial heuristic was the octile distance: the cost of the shortest path in the absence of obstacles. We treated the water terrain type as an obstacle and excluded all problems which thereby became unsolvable (e.g., the start state is in an obstacle cell). This resulted in 493298 problems situated on 342 maps. The maps were from the video games *StarCraft*, *WarCraft III*, *Baldur's Gate II* (maps scaled up to 512×512) and *Dragon Age: Origins* (Figure 1).

5 Space of Algorithms

We use the same control parameter ranges as Bulitko (2016b): $w \in [1, 10], w_c \in [1, 10], \texttt{da} \in \{true, false\}, \texttt{expendable} \in \{true, false\}, \text{lop} \in \{\min, \text{avg}, \text{median}, \max\}, b \in [0, 1]$ which define a six-dimensional space of real-time heuristic search algorithms. This formulation combines algorithms and algorithm parameters in a single space of (parameterized) algorithms.

To explore the algorithm space we first scaled up the result of Bulitko (2016b) by running 55700 algorithms on 5000 random problems each.[‡] Each batch of the 5000 problems was selected at random without replacement from the benchmark set of 493298 problems and used for 100 algorithms whose six control parameters were picked uniformly randomly from the ranges listed above. Then another batch of 5000 problems was used for the next 100 algorithms and so on, 557 times. Sample suboptimality of each algorithm is shown in a histogram in Figure 2 with $\alpha_{\max} = 10^3$.

The average of the sample suboptimalities (50.6) is an estimate of the expected performance of an algorithm randomly sampled from the algorithm space. As before (Bulitko 2016b), it is better than RTA* (Korf 1990) (186.7) but worse than daLRTA*+E (Hernández and Baier 2012; Sharon, Sturtevant, and Felner 2013) (30.5). Also, as before, 73.4% of algorithms in the experiment had better sample suboptimality than the expectation for a randomly drawn algorithm (due to the long tail of the distribution). RTA* is once again in the bottom 5.6%; daLRTA*+E is outperformed by 54.9% of all sampled algorithms.

To put the numbers in perspective, searching through the algorithm space, Bulitko (2016b) found several algorithms with the suboptimality of around 20, such as $\text{median}(7 \cdot c + h) + E$, mean suboptimality of 19.3. This is about 1.5 times better than the best tried human-designed algorithms which struggled to reach 30 such as daLRTA*+E, mean suboptimality of 31.8 or 7-daLRTA* with the mean suboptimality of 30.6. These suboptimality values are for $\alpha_{\max} = 10^5$, averaged over the entire MovingAI benchmark set which is why they differ slightly from the sample suboptimality on the 5000 problems, $\alpha_{\max} = 10^3$ reported above.

6 Adaptive Algorithm Selection

The various searches Bulitko (2016b) conducted through the algorithm space failed to find a single algorithm with the sample suboptimality substantially below 20, so he suggested selecting algorithms on a per-problem basis but did not propose a specific way of doing so.

To evaluate the potential of adaptive algorithm selection we searched through the set of the 55700 algorithms run on 5000 problems each described above. The best of the 55700 algorithms, $2.654 \cdot \text{median}_{0.153}(2.486 \cdot c + h)$+da+E, achieved sample suboptimality of 17.84 on its 5000 problems.[§] However, selecting the best algorithm from the set on a per-problem basis yielded the mean suboptimality of 8.41, about a two-fold improvement.

Given the potential, we will now present a sampling-based method for adaptive algorithm selection, first on a per-map basis and then on a per-problem basis.

6.1 Per-map Algorithm Selection

Suppose a game AI programmer received a set of new game maps from the level-design department. Which algorithms would be best on these maps? A straightforward but computationally expensive way of answering the question would be to run a large number of randomly selected algorithms on the new maps and measure their sample suboptimality. Below we propose to trade some of the computational cost for suboptimality of the selected algorithms.

The method is trivially simple: the game developer should simply run as many algorithms on the new maps as he/she has the time for and then select the algorithm with the best sample suboptimality. To evaluate the efficiency of this approach empirically, we randomly drew 1500 algorithms from the algorithm space. We then ran each of them on 4500 problems randomly chosen (without replacement) from the 493298 problems in our MovingAI benchmark set, $\alpha_{\max} = 10^3$. Aggregating per-problem suboptimality on the 4500 problems into per-map suboptimality on the 329 maps, we obtained a suboptimality matrix of 1500 rows (one per algorithm) by 329 columns (one per map). Note that the problem-per-map distribution in the MovingAI set is non-uniform: some maps have more problems on them than others. As a result, algorithm suboptimality averaged over maps may differ from that averaged over problems. For instance, suppose an algorithm had suboptimality of $10, 20, 30$ on problems a, b, c. Suppose a, b belong to map 1 and c belongs to map 2. Aggregating the values we record suboptimality of $(10 + 20)/2 = 15$ for map 1 and 30 for map 2. Thus, the non-weighted per-map average $(15 + 30)/2 = 22.5$ differs from the per-problem average of $(10 + 20 + 30)/3 = 20$. To compensate for that we weighted map averages by problem-to-map counts: $\frac{2}{3} \cdot 15 + \frac{1}{3} \cdot 30 = 20$.

We then ran 100 trials as follows. On each trial we randomly split the 329 maps/columns into 219 (old/training maps) and 110 (new/test maps). We then evaluated three ways of selecting the algorithm for the new maps.

Prior. Suppose the AI programmer does not examine the new maps at all but instead uses the best algorithm he/she previously had for the old maps. To simulate this, we se-

[‡]The choice of the specific parameters is due to a combination of observed performance and running time constraints.

[§]The low value is due to the specific sample set of 5000 problems and $\alpha_{\max} = 10^3$. Evaluating the algorithm on all MovingAI problems with $\alpha_{\max} = 10^5$ gives the mean suboptimality of 19.8.

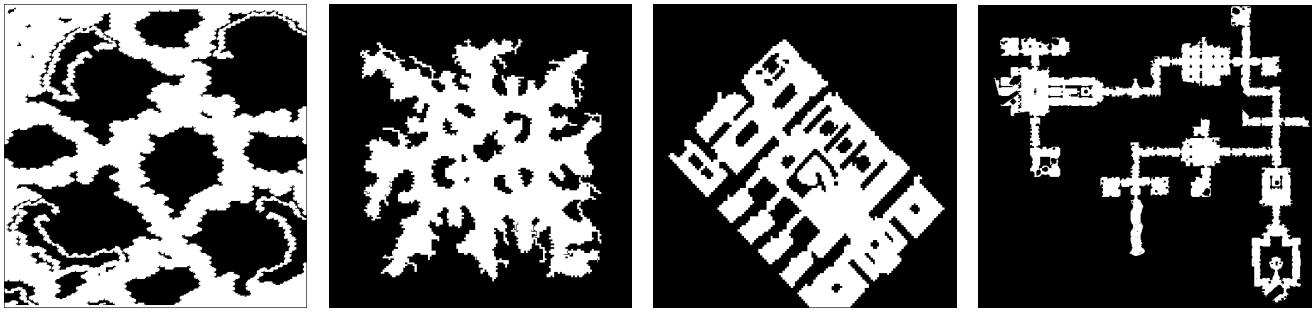

Figure 1: Sample maps from *StarCraft*, *WarCraft III*, *Baldur's Gate II* and *Dragon Age: Origins*.

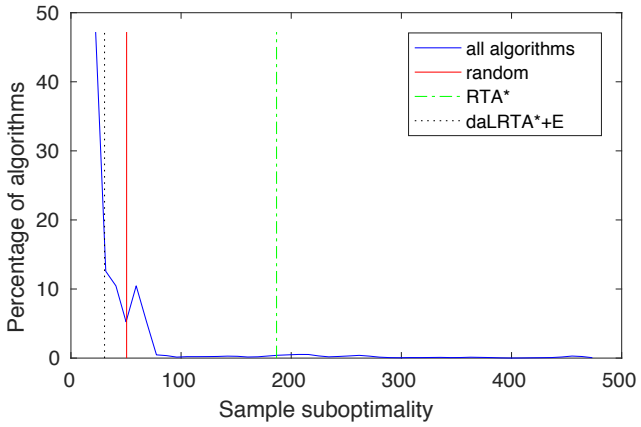

Figure 2: The space of algorithms.

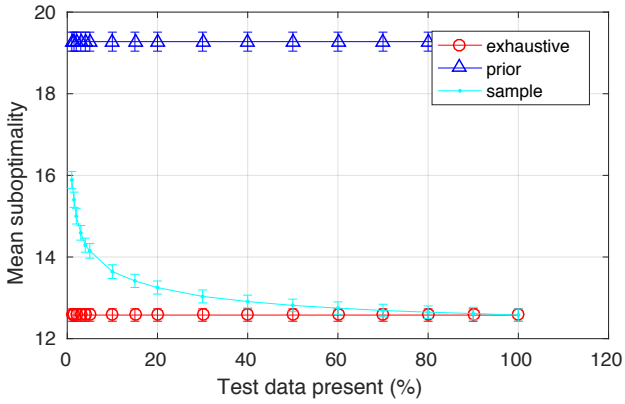

Figure 3: Algorithm selection per map.

Table 1: Algorithm selection per map: test suboptimality.

Test data present	Exhaustive	Prior	Sample
5.0%	12.6±0.15	19.3±0.23	14.1±0.18
4.0%	12.6±0.15	19.3±0.23	14.3±0.17
3.0%	12.6±0.15	19.3±0.23	14.6±0.18
2.0%	12.6±0.15	19.3±0.23	15.0±0.19
1.5%	12.6±0.15	19.3±0.23	15.4±0.19
1.0%	12.6±0.15	19.3±0.23	15.9±0.21

lected a single algorithm with the lowest suboptimality on the old maps and ran it on the new maps.

Sample. Here the AI programmer does run *some* of the algorithms on *some* of the new maps. Then for each new map, he chooses the best among the algorithms run on that map. We simulated this by randomly removing $\{0, 10, \ldots, 80, 85, 90, 95, 96, 97, 98, 98.5, 99\}$ percent of suboptimality values from the new maps[¶]. We then selected the lowest suboptimality algorithm for each new map, using the remaining suboptimality values.

Exhaustive. Here the programmer invests the time and runs all of its algorithms on all new maps. Then he/she selects the best algorithm for each new map. We simulated this by selecting the lowest suboptimality algorithm for each new map, using all new-map suboptimality values (i.e., putting back the previously removed values).

In all three cases the performance measure is mean suboptimality of per-map selected algorithms on the test maps, averaged over the 100 trials. The results are plotted in Figure 3. The error bars show the standard error of the mean. As per the last row of Table 1, the sample method runs only 1% of the available algorithms and yet yields suboptimality only 26% worse than the exhaustive search.

[¶]We made sure that each map did have suboptimality value for at least one of the 1500 algorithms.

6.2 Per-problem Algorithm Selection

We then applied the same three approaches to algorithm selection on a per-problem basis. To do so we used the same suboptimality matrix for 1500 algorithms and 4500 problems but did not aggregate suboptimality values over maps. The results are found in Figure 4 and Table 2.

Several observations are in order. First, the exhaustive method has a better suboptimality now (8 versus 12.6) because selecting an algorithm for each problem is more flexible than doing so for each map. Second, while the sample method still evaluates only 1% of the algorithms for only 28% worse suboptimality, it is not a practical method for video games because pathfinding problems in video games are usually specified during the game, as the player directs their units. Thus, the AI does not have the luxury of running several real-time search algorithms before actually solving it in real time. A practical method for per-problem algorithm

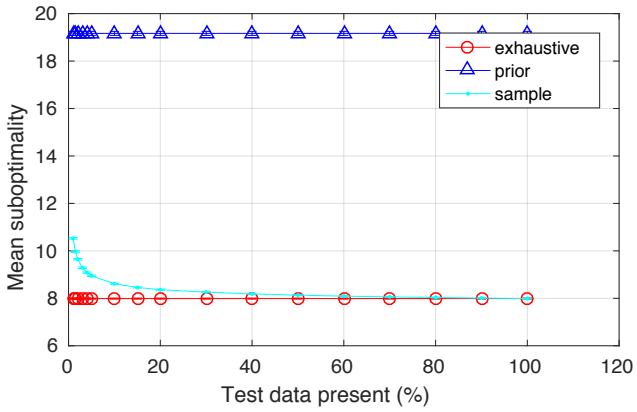

Figure 4: Algorithm selection per problem.

Table 2: Algorithm selection per problem: test suboptimality.

Test data present	Exhaustive	Prior	Sample
5.0%	8.0±0.02	19.2±0.09	8.9±0.03
4.0%	8.0±0.02	19.2±0.09	9.1±0.03
3.0%	8.0±0.02	19.2±0.09	9.3±0.03
2.0%	8.0±0.02	19.2±0.09	9.6±0.03
1.5%	8.0±0.02	19.2±0.09	10.0±0.03
1.0%	8.0±0.02	19.2±0.09	10.5±0.04

selection will use attributes of the problem to select a suitable algorithm, without running any algorithms prior to running the selected one. We discuss ways of doing so below.

7 Current Limitations and Future Work

While the method presented above is effective for per-map algorithm selection, it may be improved by using suboptimality of the algorithms run on a new map to predict suboptimality of all other algorithms. Such predictions can then be used to select an algorithm from among all algorithms in the set, not just the ones run as we did in the paper. We tried non-negative matrix factorization recently used for predicting player's enjoyment of the story (Yu and Riedl 2014). Specifically, we factored suboptimality values on the train maps into a set of basis maps and used those to predict the missing suboptimality values on the test maps. The results indicates that the predictions were insufficiently accurate as the algorithms selected using them did not outperformed those of the "prior" method. We then applied the same approach in the per-problem setting and while it outperformed the "prior" it was worse than "sample". Future work will attempt to improve the prediction accuracy.

Selecting algorithm parameters per-problem has been explored in the field of real-time heuristic search, including per-problem lookahead (Bulitko et al. 2008) and subgoal (Lawrence and Bulitko 2013) selection. The key is the mapping from problem attributes to appropriate algorithm parameters. In the past such mapping has been implemented via k-nearest-neighbor (Bulitko, Björnsson, and Lawrence 2010) and artificial neural networks (Bulitko et al. 2008) as well as simple databases (Lawrence and Bulitko 2013).

We attempted to select the best algorithm for a problem by finding problems with similar start and goal states on the same map and then combining the algorithms found to perform well on them. The results were worse than the those of the "prior" method likely due to the poor octile-distance-based similarity metric (e.g., close-by start states separated by a wall would be considered similar). Future work will investigate better similarity metrics.

We also represented a problem as a bitmap image of the video-game map with the start and goal states shown on it in color. We then trained deep convolutional networks AlexNet and GoogleNet as included in MatConvNet framework (Vedaldi and Lenc 2015) to predict the six parameters of a suitable algorithm for it. The results were slightly worse than those of the "prior" method. We are currently investigating using convolutional deep networks to predict algorithm suboptimality on a given problem, represented by a bitmap image. To avoid supplying the six algorithm parameters to the network, we train a single network per a cluster of similar algorithms. After training, a new problem is fed to each of the trained networks which predict the performance of their cluster. A representative algorithm for the cluster with the best predicted performance is then selected for the problem. This is similar to predicting algorithm performance using problem features and non-deep-learning predictors (Huntley and Bulitko 2013) and can be used both at problem and map levels.

Finally, future work will consider a broader space of real-time heuristic search algorithms adding building blocks such as deeper lookahead (Koenig and Sun 2009), time-bounded search (Björnsson, Bulitko, and Sturtevant 2009; Hernández, Baier, and Asín 2016) as well as non-pathfinding search domains.

8 Conclusions

In this paper we followed the recent work on searching the space of real-time heuristic search algorithms and took on the challenge proposed therein: to automatically select an algorithm on a per-problem basis. We presented a sampling-based method of doing so and demonstrated its benefits for selecting an algorithm per map. While the method is also effective in selecting an algorithm per problem, it is not practical. We then suggested possible practical methods as directions for future work.

Acknowledgments

Alexander Sampley performed the preliminary Deep Learning experiments. The funding was provided by NSERC. We also appreciate suggestions from the anonymous reviewers.

References

Björnsson, Y.; Bulitko, V.; and Sturtevant, N. 2009. TBA*: Time-bounded A*. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 431–436.

Botea, A. 2011. Ultra-fast optimal pathfinding without runtime search. In *Proceedings of the Conference on Artificial Intelligence and Interactive Digital Entertainment*, 122–127.

Bulitko, V., and Lee, G. 2006. Learning in real time search: A unifying framework. *Journal of Artificial Intelligence Research* 25:119–157.

Bulitko, V.; Sturtevant, N.; Lu, J.; and Yau, T. 2007. Graph abstraction in real-time heuristic search. *Journal of Artificial Intelligence Research* 30:51–100.

Bulitko, V.; Luštrek, M.; Schaeffer, J.; Björnsson, Y.; and Sigmundarson, S. 2008. Dynamic control in real-time heuristic search. *Journal of Artificial Intelligence Research* 32:419–452.

Bulitko, V.; Björnsson, Y.; and Lawrence, R. 2010. Case-based subgoaling in real-time heuristic search for video game pathfinding. *Journal of Artificial Intelligence Research* 39:269–300.

Bulitko, V. 2004. Learning for adaptive real-time search. *CoRR* cs.AI/0407016.

Bulitko, V. 2016a. Evolving real-time heuristic search algorithms. In *Proceedings of the Fifteenth International Conference on the Synthesis and Simulation of Living Systems*, in press.

Bulitko, V. 2016b. Searching for real-time search algorithms. In *Proceedings of the Symposium on Combinatorial Search*, (in press).

Hart, P.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2):100–107.

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

Hernández, C., and Meseguer, P. 2005. LRTA*(k). In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1238–1243.

Hernández, C.; Baier, J. A.; and Asín, R. 2016. Time-bounded best-first search for reversible and non-reversible search graphs. Technical report, semanticscholar.org.

Huntley, D. A., and Bulitko, V. 2013. Search-space characterization for real-time heuristic search. *CoRR* abs/1308.3309.

Ishida, T. 1992. Moving target search with intelligence. In *Proceedings of the National Conference on Artificial Intelligence*, 525–532.

Koenig, S., and Sun, X. 2009. Comparing real-time and incremental heuristic search for real-time situated agents. *Journal of Autonomous Agents and Multi-Agent Systems* 18(3):313–341.

Koenig, S. 2001. Agent-centered search. *Artificial Intelligence Magazine* 22(4):109–132.

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

Lawrence, R., and Bulitko, V. 2013. Database-driven real-time heuristic search in video-game pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games* 5(3):227–241.

Orkin, J. 2006. Three states and a plan: the AI of FEAR. In *Game Developers Conference*, 4.

Rayner, D. C.; Davison, K.; Bulitko, V.; Anderson, K.; and Lu, J. 2007. Real-time heuristic search with a priority queue. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2372–2377.

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

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

Shue, L.-Y., and Zamani, R. 1993a. An admissible heuristic search algorithm. In *Proceedings of the International Symposium on Methodologies for Intelligent Systems*, volume 689 of *LNAI*, 69–75.

Shue, L.-Y., and Zamani, R. 1993b. A heuristic search algorithm with learning capability. In *ACME Transactions*, 233–236.

Shue, L.-Y.; Li, S.-T.; and Zamani, R. 2001. An intelligent heuristic algorithm for project scheduling problems. In *Annual Meeting of the Decision Sciences Institute*.

Sturtevant, N. R., and Bulitko, V. 2011. Learning where you are going and from whence you came: H- and G-cost learning in real-time heuristic search. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 365–370.

Sturtevant, N. R., and Bulitko, V. 2014. Reaching the goal in real-time heuristic search: Scrubbing behavior is unavoidable. In *Proceedings of the Symposium on Combinatorial Search*, 166–174.

Sturtevant, N., and Bulitko, V. 2016. Scrubbing during learning in real-time heuristic search. *Journal of Artificial Intelligence Research* (in press).

Sturtevant, N., and Buro, M. 2005. Partial pathfinding using map abstraction and refinement. In *Procedings of the National Conference on Artificial Intelligence*, 1392–1397.

Sturtevant, N. R.; Bulitko, V.; and Björnsson, Y. 2010. On learning in agent-centered search. In *Proceedings of the Conference on Autonomous Agents and Multiagent Systems*, 333–340.

Sturtevant, N. R. 2007. Memory-efficient abstractions for pathfinding. In *Proceedings of the Conference on Artificial Intelligence and Interactive Digital Entertainment*, 31–36.

Sturtevant, N. R. 2012. Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games* 4(2):144 – 148.

Vedaldi, A., and Lenc, K. 2015. MatConvNet – convolutional neural networks for MATLAB. In *Proceeding of the ACM International Conference on Multimedia*, 689–692.

Yu, H., and Riedl, M. O. 2014. Personalized interactive narratives via sequential recommendation of plot points. *Computational Intelligence and AI in Games, IEEE Transactions on* 6(2):174–187.