Proceedings of the Twelfth International
Symposium on Combinatorial Search (SoCS 2019)

A Learning-Based Framework for Memory-Bounded Heuristic Search: First Results

Carlos Hernández Ulloa,[1] **Jorge Baier,**[2,6] **William Yeoh,**[3] **Vadim Bulitko,**[4] **Sven Koenig**[5]
[1] Departamento de Ciencias de la Ingeniería, Universidad Andrés Bello, Chile
[2] Departamento de Ciencia de la Computación, Pontificia Universidad Católica de Chile, Chile
[3] Department of Computer Science & Engineering, Washington University in St. Louis, USA
[4] Department of Computing Science, University of Alberta, Canada
[5] Department of Computer Science, University of Southern California, USA
[6] Instituto Milenio Fundamentos de los Datos, Chile

Introduction

Memory-bounded search algorithms are typically used when the search space is too large for regular best-first search algorithms like A* to store in memory. There exists a large class of memory-bounded best-first search algorithms including Depth-First Branch-and-Bound (DFBnB), Iterative Deepening A* (IDA*) (Korf 1985), Recursive Best-First Search (RBFS) (Korf 1993), and Simplified Memory-Bounded A* (SMA*) (Russell 1992). Each of these algorithms rely on a different strategy to ensure that they use only a bounded amount of memory: IDA* bounds the amount of memory used by repeatedly running depth-first searches, increasing the explored depth at each iteration. RBFS uses lower and upper bounds that are tightened over time as it explores the search space while keeping only $b \cdot d$ nodes in memory, where b is the branching factor and d is the depth of the tree. And, finally, SMA* keeps only a bounded number of nodes in memory by pruning the least promising nodes from the OPEN list when it runs out of memory.

In this abstract, we summarize an alternative approach to memory-bounded best-first search. It is motivated by real-time heuristic search algorithms (Korf 1990), many of which iterate the following steps until the goal is reached: up to k nodes are expanded, where k is a user-defined bound; the h values of expanded nodes are updated to make them more informed; the agents moves along a path along the search tree just expanded. We propose a general framework that iteratively (1) runs a memory-bounded best-first search algorithm that terminates when k nodes are generated. If no solution is found, (2) it updates the h-values of the generated nodes, and (3) purges the h values of some nodes from memory. As such, the *total* number of h-values ever stored by our approach is upper-bounded by a constant. Under certain (reasonable) conditions, our framework is complete and preserves the (sub)optimality guarantees of the given best-first search algorithm in tree-shaped search spaces. The main conceptual difference between our framework and the SMA* algorithm is that it can be combined with any best-first algorithm with very minor modifications.

We present experimental results where we plug into our framework memory-bounded variants of Weighted A* (Pohl 1970). On traveling salesman problems we show that our framework is often able to find better solutions than DFBnB and Weighted DFBnB (wDFBnB) and in a smaller amount of time, especially in problems with large search spaces.

Proposed Approach

Our framework can work with any best-first search algorithm that satisfies the following constraints. First, it maintains two sets of states: an $OPEN$ list (implemented as a priority queue) that contains the search frontier and a $CLOSED$ list that contains the set of nodes that have been expanded. Second, it uses a heuristic h to define the key in $OPEN$ and that it returns the goal node when it is expanded. Third, the $OPEN$ and $CLOSED$ lists remain in memory after the algorithm returns. A best-first search algorithm can be made memory bounded by keeping track of the size of its $OPEN$ and $CLOSED$ lists and stopping when expansion would lead to exceeding a given size limit.

Our main function repeatedly iterates through the following operations: (i) run *Bounded-BFS*, (ii) update h-values to make it more informed, and (iii) purge the h-values from previous invocations of the BFS search.

Bounded-BFS. We borrow the concept of heuristic updates from real-time heuristic search and make the h-values of states more informed between calls to *Bounded-BFS*. We assume the user provides a consistent heuristic function h_0, and that *Bounded-BFS* uses function h to compute the key. Function $h(s)$, when no predecessor of s has been generated and s is not the root, initializes the stored h-value of s, $h_T(s)$, to $h_0(s)$. Otherwise, it updates $h_T(s)$ with a variant of the pathmax rule (Mero 1984). $h_T(s)$ is then returned to be used by the Bounded-BFS algorithm.

Update. If a goal is not found by the algorithm when it is about to grow the $CLOSED$ and $OPEN$ lists beyond k then we update the h-values in such a way that consistency is preserved with respect to all states in $OPEN \cup CLOSED$. The update procedure essentially backs up the h-values from the states in $OPEN$ to states in $CLOSED$. The updated h-value of each state s is stored as $h_T(s)$.

Purge. After the h-values are updated, the h-values of states that were not recently expanded or generated (i.e., are not in

the *OPEN* or *CLOSED* lists) are purged from memory. In this way, in iteration $i+1$ of the main loop, the algorithm uses the (learned) h-values of states generated in iteration i.

Theoretical Analysis

Theorem 1 *If h_0 is a consistent then h_T is also consistent on $OPEN \cup CLOSED$ after each heuristic update.*

To prove completeness of our algorithm (i.e., that it finds a solution when one exists) we need the following three properties to hold. **(P1)** Bounded-BFS breaks ties in favor of nodes with a higher g-value. Such algorithms always expand subtrees of the search space whose nodes have the same f-value in a *depth-first* manner. **(P2)** The framework is given a bound that allows Bounded-BFS to expand the longest possible path (without state repetition) in the search space. This bound may sound exceedingly large but it is important to note that this is a worst-case theoretical bound. **(P3)** If state t has been generated from state s and $f(s) < f(t)$, then $h(s) < c(s,t) + h(t)$. This property restricts the way in which f is defined in terms of g and h and is satisfied by a wide range of algorithms including Weighted A*, Greedy Best-First Search and Focal Search (Pearl and Kim 1982).

Theorem 2 *If our algorithm satisfies P1, P2, and P3, then it is complete over tree-shaped graphs.*

Theorem 3 *If the memory-unbounded version of Bounded-BFS run with a consistent heuristic h returns a solution upper-bounded by C then a solution returned by our framework with Bounded-BFS and h is upper-bounded by C.*

Table 1: Experimental results.

		Time (ms)				Solution Cost			
Cities	Alg.	w=1.0	1.1	1.2	1.3	w=1.0	1.1	1.2	1.3
22	wA*(464)	303	0.4	0.2	0.3	183.8	184.3	184.1	184.1
	wA*(926)	286	0.1	0.3	0.4	183.8	184.3	184.1	184.1
	wA*(1850)	275	0.2	0.3	0.6	183.8	184.3	187.0	201.5
	wDFBnB	24	2	0.1	0.1	183.8	184.3	196.6	201.8
32	wA*(994)	-	7,705	694	329	-	465.8	473.3	471.1
	wA*(1986)	-	3,090	438	424	-	465.8	465.8	471.1
	wA*(3970)	-	2,697	747	768	-	465.8	465.8	471.1
	wDFBnB	9,147	356	13	0.5	465.8	467.4	472.7	492.0
51	wA*(2552)	-	97	584	2	-	426.9	444.9	448.1
	wA*(5102)	-	250	41	4	-	426.9	444.7	448.1
	wA*(10202)	-	392	67	5	-	426.9	444.7	448.1
	wDFBnB	-	1,712	578	90	-	434.8	451.2	469.9
55	wA*(2972)	-	-	-	83,594	-	-	-	579.7
	wA*(5942)	-	-	-	32,251	-	-	-	592.4
	wA*(11882)	-	-	-	31,509	-	-	-	588.6
	wDFBnB	-	-	-	4,087	-	-	-	574.1
76	wA*(5702)	-	-	13	4	-	-	560.7	561.7
	wA*(11402)	-	-	24	6	-	-	560.7	561.7
	wA*(22802)	-	-	38	8	-	-	560.7	561.7
	wDFBnB	-	-	21,302	1,493	-	-	587.9	612.6
101	wA*(10102)	-	-	4,473	45	-	-	659.3	669.0
	wA*(20202)	-	-	3,463	44	-	-	663.8	671.0
	wA*(40402)	-	-	445	37	-	-	664.7	666.3
	wDFBnB	-	-	-	31	-	-	-	723.0
151	wA*(22652)	-	-	-	-	-	-	-	-
	wA*(45302)	-	-	-	13,972	-	-	-	743.9
	wA*(90602)	-	-	-	25,629	-	-	-	743.9
	wDFBnB	-	-	-	-	-	-	-	-

Empirical Evaluation

To evaluate the potential of our framework, we ran it using Weighted A* with $w \in \{1.0, 1.1, 1.2, 1.3\}$ on 7 routing problems from the literature[1] (`P-n22-k2.vrp`, `A-n32-k5.vrp`, `eil51.tsp`, `A-n55-k9.vrp`, `pr76.tsp`, `eil101.tsp` and `M-n151-k12.vrp`) with 22, 32, 51, 55, 76, 101, and 151 cities respectively. Each routing problem was solved as a symmetric TSP (sTSP). We used three memory-bound values (k) linked to the size of a given problem. We defined the minimum bound (mb) for an sTSP of n cities, with $n > 3$, as $mb = (n(n-1))/2 + 2$, which is the minimum number of nodes that must be generated to obtain a solution of depth n in the search tree of an sTSP. The bound values were then $k \in \{2mb, 4mb, 8mb\}$. We compare the results with wDFBnB (Hernández and Baier 2014) for the same values of w. The initial heuristic, h_0, is the in-out estimator (Pohl 1973).

We used a time limit of 100 seconds. Table 1 lists the number of cities of the problem, the algorithms (bound in parentheses), the runtime in milliseconds and the resulting solution cost. Hyphens mean that the algorithm timed out before finding a solution.

With Bounded Weighted A*, our framework is slower than wDFBnB on smaller problems (22, 32 and 55 cities) but superior for larger problems. For example, with 151 cities only our algorithm makes the time cutoff. With 101 cities, our algorithm finished for $w = 1.2$ while wDFBnB does not finish in time. In terms of suboptimality, our algorithm finds shorter solutions for most problems and w values.

Second, on some problems and some w values, the runtime decreases when the memory bound k increases. However, for other problems the opposite happens. This suggests that, to minimize the runtime, k has to be selected in a problem-dependent manner.

References

Hernández, C., and Baier, J. A. 2014. Toward a search strategy for anytime search in linear space using depth-first branch and bound. In *SoCS*.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence* 27(1):97–109.

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

Korf, R. E. 1993. Linear-space best-first search. *Artificial Intelligence* 62(1):41–78.

Mero, L. 1984. A heuristic search algorithm with modifiable estimate. *Artificial Intelligence* 23:13–27.

Pearl, J., and Kim, J. H. 1982. Studies in semi-admissible heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 4:392–399.

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. *Artificial Intelligence* 1(3-4):193–204.

Pohl, I. 1973. The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving. In *IJCAI*, 12–17.

Russell, S. 1992. Efficient memory-bounded search methods. *ECAI*.

[1] http://neo.lcc.uma.es/vrp/vrp-instances/, http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/