Proceedings of the Fourteenth International Symposium on Combinatorial Search (SoCS 2021)

Iterative-deepening Bidirectional Heuristic Search with Restricted Memory*

Shahaf S. Shperberg[1], **Steven Danishevski**[1], **Ariel Felner**[1], **Nathan R. Sturtevant**[2]
[1] Ben-Gurion University, Be'er Sheva, Israel
[2] University of Alberta, Edmonton, Canada
shperbsh@post.bgu.ac.il, stiven@post.bgu.ac.il, felner@bgu.ac.il, nathanst@ualberta.ca

Abstract

This extended abstract presents a bidirectional heuristic search algorithm called IDBiHS that operates under restricted memory. Several variants of this algorithm are introduced for different types of memory restrictions, and are compared against existing algorithms with similar restrictions.

Restricted Memory Search Algorithms

Search algorithms can be classified into three main categories with regards to the amount of memory they consume.

1. **Unrestricted memory (UM)**. Such algorithms use memory proportional to the size of the search tree that they explored, which could grow polynomially or exponentially with the search depth d. UM algorithm often fail to solve hard problems due to memory exhaustion.
2. **Linear memory (LM)**. An algorithm may only store a single branch of the search tree (a path), and its memory consumption is $O(d)$. IDA* (Korf 1985) is a prominent example of an LM unidirectional heuristic search (UniHS) algorithm, as well as it's many enhancements. In addition there are other UniHS LM algorithms such as IBEX (Helmert et al. 2019), and an LM BiHS algorithm which is a variant of the SFBDS algorithm (Felner et al. 2010; Lippi, Ernandes, and Felner 2016).
3. **Fixed memory (FM)**. An algorithm is given a fixed amount of memory M (on top of the memory required for storing a single path) and must never exceed it. Cases 2 and 3 are denoted hereafter as **Restricted Memory (RM)**.

   Perimeter search (Dillenburg and Nelson 1994) is a class of FM BiHS algorithms in which a perimeter around $start$ or around $goal$ is constructed and a DFS is executed from the opposite direction until the perimeter is reached. Two notable variants of perimeter search are BIDA* (Manzini 1995), and BAI (Kaindl et al. 1995).

Iterative-deepening BiHS

We now introduce our new admissible algorithm, iterative-deepening bidirectional heuristic search, or IDBiHS. We first present an LM variant and then proceed to FM variants.

*A full version of this paper appears in ICAPS-2021

LM Variant of IDBiHS

IDBiHS maintains a threshold on the current lower-bound of solutions cost (initialized as $h(start, goal)$); this threshold is used as a bound to the f-value of nodes searched, thus it is called fT. In each iteration the task is to find a solution of cost fT. If such a solution is not found, fT is incremented, and a new iteration begins. IDBiHS uses fT to bound the g-values of searched nodes from each direction using a *split function* (similar to those used by the GBFHS algorithm (Barley et al. 2018)). Specifically, the split function determines the *meeting point* by setting the forward g-threshold gT_F.

Once gT_F is obtained, a forward DFS procedure (F_DFS) is called from $start$. When F_DFS encounters a node n_F, it has three cases:

1. **Expand.** If $f_F(n_F) \leq fT$ and $g_F(n_F) \leq gT_F$, then expand n_F and move to one of its children.
2. **Prune.** If $f_F(n_F) > fT$, prune n_F and backtrack.
3. **Suspend and Match.** If $f_F(n_F) \leq fT$ and $g_F(n_F) > gT_F$, suspend F_DFS and call the backward DFS (B_DFS) attempting to match n_F from the backward side. B_DFS is called for every forward frontier node n_F until a solution is found or all forward frontier nodes have been explored.

When calling B_DFS on candidate nodes n_F, the g-threshold for the backward direction (gT_B) needs to be defined. gT_B is defined specifically for each node n_F to be matched as follows: $gT_B(n_F) = fT - g_F(n_F) - \epsilon$, where ϵ is the least-cost edge

Upon reaching a candidate meeting node n_F in F_DFS, B_DFS performs a DFS iteration from $goal$. When B_DFS encounters a node n_B, it has three options:

1. **Expand.** If $f_B(n_B) \leq fT$ and $g_B(n_B) \leq gT_B$, then expand n_B and move to one of its children.
2. **Prune.** If $f_B(n_B) > fT$, prune n_B and backtrack.
3. **Match.** If $f_B(n_B) \leq fT$ and $g_B(n_B) > gT_B$, match n_F against n_B. If they represent the same state, a solution has been found. Otherwise, n_B can be immediately pruned.

After B_DFS finishes without matching n_F, B_DFS returns *false* and F_DFS resumes by backtracking from n_F. Note that in every iteration F_DFS is called once, while B_DFS is called many times, once for each forward frontier node.

Domain	H	Expanded				Time(sec)			
		IDA*	SFBDS JIL(1)	IDBiHS 0.5	IDBiHS BW	IDA*	SFBDS JIL(1)	IDBiHS 0.5	IDBiHS BW
P[12]	G	95 (76%)	169 (79%)	85 (76%)	**83** (77%)	0.00	0.00	0.00	0.00
	G-1	9,665 (72%)	8,622 (74%)	3,417 (70%)	**2,857** (72%)	0.01	0.01	0.00	0.00
	G-2	383,488 (68%)	205,469 (71%)	75,205 (70%)	**72,939** (72%)	0.25	0.15	0.06	**0.05**
STP	MD	242,460,834 (50%)	139,225,772 (46%)	174,414,968 (40%)	**137,331,587** (48%)	47.85	35.90	36.44	**29.29**
Grid	Octile	**47,060** (75%)	131,488 (78%)	210,353 (78%)	122,304 (79%)	**0.00**	0.01	0.01	0.01

Table 1: Average node expansions and runtime of linear-memory algorithms

Once all forward paths have been explored by F_DFS and no solution has been found fT is incremented and a new iteration begins.

When computing the f-value of nodes in B_DFS a front-to-front heuristic towards n_F can be used (is available). In addition, if the heuristic is known to be consistent then another improvement is possible. Kaindl and Kainz (1997) defined $\mathit{Diff}_F(n_F) = g_F(n_F) - h_B(n_F)$, and $\mathit{Diff}_B(n_B) = g_B(n_B) - h_F(n_B)$, corresponding to the *error* of $h_B(n_F)$ and $h_F(n_B)$ respectively. When trying to connect a node n_B generated by B_DFS to a given forward frontier node n_F, it holds that $f_B(n_B) + \mathit{Diff}_F(n_F) \leq d(n_B, start)$ and that $\mathit{Diff}_B(n_B) \leq d(n_B, goal)$. Therefore, the maximum of $f_B(n_B) + \mathit{Diff}_F(n_F)$ and $f_F(n_F) + \mathit{Diff}_B(n_B)$ can be used instead of $f_B(n_B)$ to improve the pruning.

FM Variants of IDBiHS

We now introduce two FM variants of IDBiHS.

A*+IDBiHS is inspired by A*+IDA* (Bu and Korf 2019). Given a memory budget M, A*+IDA* first runs A* until either a solution is found, or the memory used by A* exceeds M. Then, it continues by running IDA* starting from the OPEN nodes, denoted hereafter as N_F. Similarly, A*+IDBiHS executes A* from $start$ until it runs out of memory. Then, IDBiHS is executed sequentially from the nodes in N_F as follows. fT is initialized to be the minimal f-value in N_F. Next, F_DFS is executed on all nodes $n \in N_F$ for which $f_F(n) = fT$. For each execution of F_DFS starting from a node $n \in N_F$. Then, once F_DFS fails to find an optimal solution from n, $h(n)$ is incremented.

IDBiHS-Trans uses a transposition table to store nodes. While A*+IDA* stores nodes near $start$ to minimize the number of duplicate nodes on lower depths, IDBiHS-Trans stores frontier nodes in order to match against multiple nodes at once, and thus calling B_DFS fewer times (from the other frontier). In particular, if the available memory is sufficient to store K frontier nodes, then the number of B_DFS calls will be reduced by a factor of K.

Empirical Evaluation

We performed experiments on three domains: the Pancake Puzzle, 15 puzzle (STP), and 8-Grid-based pathfinding. First, we compare the following LM algorithms: IDA*, SFBDS with JIL(1) as a jumping policy, and IDBiHS. IDBiHS was evaluated using two different split policies. The first policy, denoted as IDBiHS-0.5 splits each fT in the middle. The second policy aims to *balance the workload* (BW) between the two frontiers. IDBiHS-BW counts the number of nodes expanded in the previous iteration in the forward search, and those expanded in the backward search, and increments the g-threshold of the direction that expanded fewer nodes.

Table 1 presents the averages number of node expansions before finding an optimal solution and the runtime for the LM algorithms. We also report (in parenthesis) what percentage of the overall expansions was performed in the last C-layer. IDBiHS-BW is the best algorithm across all exponential domains. It outperforms IDA*, both in node expansions and time, by up to a factor of 5.3 and SFBDS by up to a factor of 4. The improvement is more significant for weaker heuristics. The runtime of IDBiHS-0.5 is slightly higher than that of IDBiHS-BW, but is still competitive, inferior only to SFBDS in STP. However, IDBiHS is outperformed by IDA* in the polynomial domain (Grid). Finally, the percentage of nodes expanded in the last C-layer is proportional to the heuristic strength. While for the UM algorithms most of the node expansions are usually performed before the last C-layer, for the LM algorithms most of the node expansions are performed in the last layer. Nonetheless, there is no significant different between the different LM algorithms in terms of the relative effort invested in the last C-layer.

We have also compared the baseline FM algorithms, Max-BAI, BIDA* and A*+IDA*, to our new algorithms, A*+IDBiHSand IDBiHS-Trans. In order to consider meaningful amounts of memory, we used a memory budget proportional to the number of states (denoted by S) stored by the best among the UM algorithms MM, A*, and reverse-A*. Specifically, we used $C \cdot S$, where $C \in \{50\%, 10\%, 1\%\}$. In most cases, IDBiHS-Trans requires the fewest expansions. However, it has a large runtime overhead per node. Nonetheless, IDBiHS-Trans achieved the fastest runtime in STP. Max-BAI performed well, however, its performance deteriorated the most when less memory was available. By contrast, A*+IDBiHS uses less overhead per node, and therefore is often the fastest algorithm.

All fixed memory algorithms experience a trade-off. Having more memory often results in fewer node expansions. However, more memory results in a larger overhead per node due to the cost of maintaining the required data-structures. Nonetheless, in most cases, it is more beneficial to use available memory (FM) than to not use memory at all (LM).

References

Barley, M. W.; Riddle, P. J.; López, C. L.; Dobson, S.; and Pohl, I. 2018. GBFHS: A Generalized Breadth-First Heuristic Search Algorithm. In *SoCS*, 28–36. AAAI Press.

Bu, Z.; and Korf, R. E. 2019. A*+IDA*: A Simple Hybrid Search Algorithm. In *IJCAI*, 1206–1212. ijcai.org.

Dillenburg, J. F.; and Nelson, P. C. 1994. Perimeter Search. *Artificial Intelligence* 65(1): 165–178.

Felner, A.; Moldenhauer, C.; Sturtevant, N. R.; and Schaeffer, J. 2010. Single-Frontier Bidirectional Search. In *AAAI*, 59–64. AAAI Press.

Helmert, M.; Lattimore, T.; Lelis, L. H. S.; Orseau, L.; and Sturtevant, N. R. 2019. Iterative Budgeted Exponential Search. In *IJCAI*, 1249–1257. ijcai.org.

Kaindl, H.; and Kainz, G. 1997. Bidirectional Heuristic Search Reconsidered. *J. Artif. Intell. Res.* 7: 283–317.

Kaindl, H.; Kainz, G.; Leeb, A.; and Smetana, H. 1995. How to Use Limited Memory in Heuristic Search. In *IJCAI*, 236–242. Morgan Kaufmann.

Korf, R. E. 1985. Depth-First Iterative-Deepening: An Optimal Admissible Tree Search. *Artif. Intell.* 27(1): 97–109.

Lippi, M.; Ernandes, M.; and Felner, A. 2016. Optimally solving permutation sorting problems with efficient partial expansion bidirectional heuristic search. *AI Commun.* 29(4): 513–536.

Manzini, G. 1995. BIDA*: An Improved Perimeter Search Algorithm. *Artif. Intell.* 75(2): 347–360.