BFHSP: A Breadth-First Heuristic Search Planner

Rong Zhou and Eric A. Hansen
Department of Computer Science and Engineering
Mississippi State University
Mississippi State, MS 39762
{rzhou,hansen}@cse.msstate.edu

Overview

Our Breadth-First Heuristic Search Planner (BFHSP) is a domain-independent STRIPS planner that finds sequential plans that are optimal with respect to the number of actions it takes to reach a goal. We developed BFHSP as part of our research on space-efficient graph search. It uses breadth-first search since we found that breadth-first search is more efficient than best-first search when divide-and-conquer solution reconstruction is used to reduce memory requirements. The specific search algorithm used by BFHSP is Breadth-First Iterative-Deepening A* (Zhou & Hansen 2004) with some enhancements. Like HSP2.0 (Bonet & Geffner 2001a), BFHSP can search in either progression or regression space. The admissible heuristic function used is the $h_{\max}$ heuristic (Bonet & Geffner 2001b) in progression search, and the *max-pair* heuristic (Haslum & Geffner 2000) in regression search.

Divide-and-Conquer Solution Reconstruction

Our research objective in developing BFHSP is to design heuristic search algorithms that can find optimal plans using limited memory, especially in complex graphs with many duplicate paths where IDA* is usually ineffective. BFHSP uses divide-and-conquer solution reconstruction to reduce its memory requirement. Divide-and-conquer solution reconstruction was first introduced to the heuristic search community by Korf (1999), based on a similar strategy used in dynamic programming algorithms for sequence comparison. The technique exploits the fact that it is not necessary to store all expanded nodes in a Closed list in order to prevent re-generation of already-expanded nodes. Instead, it suffices to store a subset of nodes that forms a *boundary* between the frontier and interior of the explicit search graph (Zhou & Hansen 2003).

Although nodes inside the boundary can be removed from memory without risking duplicate search effort, this means it is no longer possible to reconstruct a solution by the traditional traceback method. To allow divide-and-conquer solution reconstruction, each node stores information about a node along an optimal path to it that divides the problem in about half. Once the search problem is solved, information about this midpoint node is used to divide the search problem into two subproblems: the problem of finding an optimal path from the start node to the midpoint node, and the problem of finding an optimal path from the midpoint node to the goal node. Each of these subproblems is solved by the same search algorithm, in order to find a node in the middle of their optimal path. The process continues recursively until primitive subproblems are reached, and all nodes on the optimal solution path have been identified. Since the time it takes to solve all subproblems is very short compared to the time it takes to solve the original search problem, this technique saves a great deal of memory in exchange for limited time overhead for solution reconstruction.

There are several different ways to store information about the midpoint node. BFHSP adopts the method used by Sparse-Memory A* (Zhou & Hansen 2003). Each node stores a pointer to its predecessor or to an intermediate node along an optimal path, called a *relay node*, which is retained in memory. The advantage of this approach is that it takes less space and allows faster solution reconstruction.

Breadth-First Heuristic Search

A significant difference between BFHSP and HSP2.0 is that BFHSP uses a breadth-first instead of the traditional best-first strategy of node expansion. This difference is based on our discovery that when divide-and-conquer solution reconstruction is used, breadth-first search is more memory-efficient than best-first search (Zhou & Hansen 2004). The reason for this is that memory requirements depend on the number of nodes needed to maintain a boundary between the frontier and interior of the search, and not the total number of nodes expanded. Figure 1 conveys an intuition of how breadth-first search results in a smaller set of boundary nodes. It shows that best-first node expansion "stretches out" the boundary, whereas breadth-first search does not and uses an upper bound to limit the width of the boundary. Although breadth-first search expands more nodes than best-first search, the memory it saves by storing a smaller boundary results in more efficient search.

Note that BFHSP uses both an admissible heuristic function and an upper bound to limit exploration of the search space. No node is inserted into the Open list if its f-cost is greater than an upper bound on the cost of an optimal solution, since such nodes cannot be on an optimal path.

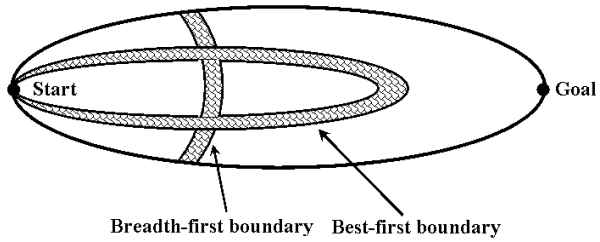

Figure 1: Comparison of best-first and breadth-first boundaries. The outer ellipse encloses all nodes with f-cost less than or equal to an (optimal) upper bound.

A breadth-first search graph divides into layers, one for each depth. To prevent duplicate search effort, BFHSP keeps (at least) three layers in memory: the currently-expanding layer, its immediate previous layer, and the next layer. In addition, it also stores a *relay layer* for the purpose of solution reconstruction. Other layers can be pruned to recover memory.

BFHSP provides two options regarding how previously-explored layers are removed from memory. The first option, called *aggressive pruning*, removes immediately any layer that is not one of the four layers mentioned previously. The second option, called *lazy pruning*, is the same as the first one, except that it removes layers only when memory is full. Because BFHSP with lazy pruning is the same as breadth-first branch-and-bound search until memory is full, the time overhead of solution reconstruction is avoided if there is enough memory. In IPC-4, BFHSP uses lazy pruning during solution reconstruction, because subproblems are often small enough in size that they can be solved by ordinary breadth-first branch-and-bound search.

For undirected graphs, storing only one previous layer is sufficient to prevent *all* duplicate search effort (Zhou & Hansen 2004). For directed graphs, the number of times a node can be re-generated by BFHSP is *at most* linear in the depth of the search. This contrasts sharply to the potentially exponential number of node re-generations for linear-space search algorithms that rely on depth-first search.

Breadth-First Iterative-Deepening A*

Although BFHSP uses an upper bound to limit its search space, it is possible to run the planner without a previously-computed upper bound. Instead, an iterative-deepening strategy can be used to avoid expanding nodes that have an f-cost greater than a hypothetical upper bound. *Breadth-First Iterative-Deepening A** (BFIDA*) first runs breadth-first heuristic search using the f-cost of the start node as an upper bound. If no solution is found, it increases the upper bound by one (or to the least f-cost of any unexpanded nodes) and repeats the search until a solution is found. In this respect, it is similar to Depth-First Iterative-Deepening A* (Korf 1985). The difference is that it never expands the same node twice during the same iteration. (This claim holds for undirected graphs, and for many – but not all – directed graphs.) The amount of memory used is the same as the amount of memory BFHSP would use given an optimal upper bound. However, BFIDA* may run more slowly than BFHSP with a previously-computed upper bound, because running multiple iterations of BFHSP takes extra time.[1]

To reduce the number of iterations, BFHSP uses an improved version of BFIDA*, called *BFIDA*_CR*, that is based on an idea used in IDA*_CR (Sarkar *et al.* 1991), where "CR" stands for controlled re-expansion. The idea is to create an algorithm in which the number of nodes expanded in successive iterations increases exponentially with the number of iterations. Among other things, BFIDA*_CR has an interesting advantage over IDA*_CR. That is, for planning problems with unit action cost, BFIDA*_CR can guarantee that the first solution found is optimal, because it uses breadth-first search; whereas IDA*_CR cannot, due to its use of depth-first search.

Unlike conventional iterative-deepening search, which increases its bound to the minimum f-cost of any unexpanded nodes after each iteration, BFIDA*_CR may use a slightly higher bound to reduce overall node expansions by reducing the number of iterations it takes to find a solution. The benefit of using BFIDA*_CR is most evident in problems with small branching factor but long solution depth, such as the newly-released *airport* domain in IPC-4.

Admissible Search Heuristics

BFHSP uses the admissible $h_{\max}$ heuristic (Bonet & Geffner 2001b) in progression search and the *max-pair* heuristic (Haslum & Geffner 2000) in regression search. In addition, we implemented the *max-triple* heuristic for regression search by considering triples (instead of pairs) of atoms. The max-triple heuristic is more accurate than the max-pair heuristic, and often results in four or five-fold reduction in node expansions. The max-triple heuristic is, however, more time-consuming to compute and takes more memory to store, because its time (and space) complexity is cubic in the number of atoms. As a result, it is not the default search heuristic in BFHSP. An interesting observation, however, is that using the max-triple heuristic lets BFHSP solve some STRIPS instances of the *philosophers* problem that cannot be solved by using the max-pair heuristic in regression search, because using the max-triple heuristic makes it possible to recognize high-order mutexes (Blum & Furst 1995) and to prune states that contain them.

Special Features

Breadth-first (heuristic) search, when applied to problems with unit action cost, has the advantage that when a node is first generated, an optimal path to it has been found. With some changes to the algorithm, this property can be exploited to reduce the internal memory requirement of BFHSP. In fact, we have developed an external-memory version of BFHSP that uses disk storage in order to bound its internal-memory requirement (Forthcoming). However, we did not use it in IPC-4, because given the constraints of the Competition (30 minutes of CPU time and 1 gigabytes of

[1] It is possible to improve the efficiency of BFHSP by reusing information stored from previous iterations of BFIDA*, but we did not explore this possibility in our current implementation.

RAM), it is unclear whether memory is the bottleneck instead of time. In our experience with IPC-4, there are more problems for which BFHSP ran out of time before it ran out of memory, than the other way around.

Conclusion

Our primary design goal for BFHSP is to reduce its memory requirement, which is an important issue for many optimal heuristic search-based planners. Unfortunately, the time and space constraints of this Competition do not make it possible to fully demonstrate the advantages of BFHSP. For example, we have run BFHSP for days without running out of memory and have used it to find optimal plans for STRIPS problems that are far beyond the reach of HSP2.0 or HSPr* (Haslum & Geffner 2000). We believe that in many real-world applications where optimality is important, memory is likely to be a bottleneck, and BFHSP will have an advantage over other optimal planners.

Acknowledgement

We thank Blai Bonet and Hector Geffner for making publicly available their code for HSP2.0, upon which BFHSP is built.

References

Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis. In *Proc. of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, 1636–42.

Bonet, B., and Geffner, H. 2001a. Heuristic search planner 2.0. *AI Magazine* 22(3):77–80.

Bonet, B., and Geffner, H. 2001b. Planning as heuristic search. *Artificial Intelligence* 129(1):5–33.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In *Proc. of the 5th International Conference on AI Planning and Scheduling*, 140–149.

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

Korf, R. 1999. Divide-and-conquer bidirectional search: First results. In *Proc. of the 16th International Joint Conference on Artifiicial Intelligence (IJCAI-99)*, 1184–1189.

Sarkar, U.; Chakrabarti, P.; Ghose, S.; and Sarkar, S. D. 1991. Reducing reexpansions in iterative-deepening search by controlling cutoff bounds. *Artificial Intelligence* 50:207–221.

Zhou, R., and Hansen, E. 2003. Sparse-memory graph search. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, 1259–1266.

Zhou, R., and Hansen, E. 2004. Breadth-first heuristic search. In *Proc. of the 14th International Conf. on Automated Planning and Scheduling*.