Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence

Trading Space for Time in Grid-Based Path Finding

William Lee and Ramon Lawrence
http://db.ok.ubc.ca/path
University of British Columbia Okanagan
3333 University Way, Kelowna, BC, Canada, V1V 1Z3
wylliam.lee@gmail.com and ramon.lawrence@ubc.ca

Abstract

Grid-based path finding is required in many games to move agents. We present an algorithm called DBA* that uses a database of pre-computed paths to reduce the time to solve search problems. When evaluated using benchmark maps from Dragon Age™, DBA* requires less time for search and produces less suboptimal paths than the PRA* implementation used in Dragon Age™.

Introduction

As games have evolved, their size and complexity has increased. It is not uncommon for hundreds of agents to be path finding simultaneously. Consequently, game developers are often forced to make compromises and spend considerable time tuning and validating their implementations.

Variations of A* (Hart, Nilsson, and Raphael 1968) and PRA* (Sturtevant 2007) are commonly used in video games. However, they must plan a complete (but possibly abstract) path before the agent can move. Real-time algorithms such as kNN LRTA* (Bulitko, Björnsson, and Lawrence 2010) and HCDPS (Lawrence and Bulitko 2012) guarantee a constant bound on planning time, but often require a considerable amount of pre-computation time and space.

We propose a grid-based path finding algorithm called DBA* that combines the real-time constant bound on planning time with abstraction using sectors as used in PRA*. DBA* uses less space than previous real-time algorithms while providing better paths than PRA*. DBA* was evaluated on Dragon Age™ maps with average suboptimality less than 3%, and requiring on average less than 200 KB of memory and between 1 and 10 seconds for pre-computation.

Background

This work formulates grid-based path finding as a heuristic search problem where states are vacant grid cells and edges connect adjacent cells. Octile distance is used for the heuristic. Algorithms are evaluated based on path quality, memory used, and time. *Response time* is the maximum planning time per move. *Suboptimality* is $(\frac{pathCost}{optimalCost} - 1) * 100\%$. The *overall time* is the time to construct the full solution, and the *average move time* is the overall time divided by the number of moves made.

A* is a common algorithm for path finding as it has good performance characteristics. However, its overall time depends on the problem size and complexity, resulting in highly variable response time and memory usage. The PRA* variant implemented in Dragon Age (Sturtevant 2007) improves response time and memory use by abstracting the search space into sectors and computing a complete solution in the abstract space. The abstraction requires a small amount of memory, and it results in solutions that are suboptimal (between 5 to 15%).

Real-time algorithms guarantee a constant bound on planning time per action. kNN LRTA* (Bulitko, Björnsson, and Lawrence 2010) creates a database of compressed problems and online uses the closest database problem as a solution template. There is no guarantee of complete coverage, which may result in very suboptimal solutions. HCDPS (Lawrence and Bulitko 2012) performs abstraction by defining regions where all states are bidirectionally hill-climbable with the region representative. A compressed path database was then constructed defining paths between all region representatives. This allows all searches to be done using hill-climbing. HCDPS is faster than PRA* with improved path suboptimality, but its abstraction consumes more memory. LRTA* with subgoals (Hernández and Baier 2011) pre-computes a subgoal tree from each goal state where a subgoal is the next target state to exit a heuristic depression. Online, the subgoals help it escape a heuristic depression.

Approach Overview

DBA* performs offline pre-computation before online path finding. The offline stage abstracts the grid into sectors and regions and pre-computes a database of paths to navigate between adjacent regions. A sector is an n x n grid of cells (e.g. 16 x 16). A region is a set of cells in a sector that are mutually reachable without leaving the sector. Regions are produced by one or more breadth-first searches until all cells in a sector are assigned to a region. A sector may have multiple regions, and a region is always in only one sector. A region center or representative state is selected for each region. The construction of regions follows that in (Sturtevant 2007).

DBA* then proceeds to construct a database of optimal paths between adjacent regions using A*. Each path found

is stored in compressed format by storing a sequence of subgoals, each of which can be reached from the previous subgoal via hill-climbing. Hill-climbing compressible paths are described in (Bulitko, Björnsson, and Lawrence 2010).

To navigate between non-adjacent regions, an R x R matrix (where R is the number of regions) is constructed where cell (i, j) contains the next region to visit on a path from R_i to R_j, the path cost, and the path itself (if the two regions are adjacent). The matrix is initialized with the paths between adjacent regions, and dynamic programming is used to determine the costs and next region for all cells.

Given the start and goal, the region for the start R_s and for the goal R_g is known if there is only one region in the sector, or calculated using a sector bounded BFS. The path matrix is used to build a path between R_s and R_g. This path may be stored in the database if the regions are adjacent, or it is the concatenation of paths by navigating through adjacent regions from R_s to R_g. The complete path consists of navigating from s to the region representative R_s, then following the subgoals to region representative R_g, then to g.

The response time is minimal as the agent can navigate from s to the start region representative R_s as soon as the BFS is completed. The complete path between regions is iterative with the agent following the subgoals in a path from one region to the next without constructing the entire path.

Experimental Results

Algorithms were evaluated on ten standard benchmark maps from Dragon Age: Origins™(available at http://movingai.com), which have an average number of open states of 96,739 and total cells of 574,132. For each map, 100 sample problems were run from the problem set. The algorithms compared were DBA*, PRA* as implemented in Dragon Age (Sturtevant 2007), HCDPS, and LRTA* with subgoals (Hernández and Baier 2011). HCDPS was run for neighborhood depth $L = \{1, 2, 3, 4\}$. PRA* was run for sector sizes of 16 x 16, 32 x 32, 64 x 64, and 128 x 128. DBA* was run under all combinations of parameters.

Figure 1 displays suboptimality versus move time, and Figure 2 displays suboptimality versus memory used. Each point represents an algorithm with a different configuration. For PRA*, there are 4 points corresponding to 16, 32, 64, and 128 sector sizes. HCDPS has 4 points corresponding to $L = \{1, 2, 3, 4\}$. For DBA*, there are separate series for each sector size (16, 32, 64, 128), and each series consists of 4 points corresponding to $L = \{1, 2, 3, 4\}$. DBA* variants are generally faster with better suboptimality than HCDPS and PRA*. Larger sector sizes for both DBA* and PRA* hurt suboptimality and increase time, but with a much larger effect for PRA* because the quality of the abstraction is poor for larger sector sizes and the problem space is not significantly reduced in size. Since DBA* uses its path database rather than solving using A* in the abstract space, its move time and suboptimality does not increase as much with larger sectors. LRTA* with subgoals (shown as sgLRTA*) is almost optimal with a relatively high move time. DBA* consumes less memory than sgLRTA* and HCDPS* but slightly more than PRA*.

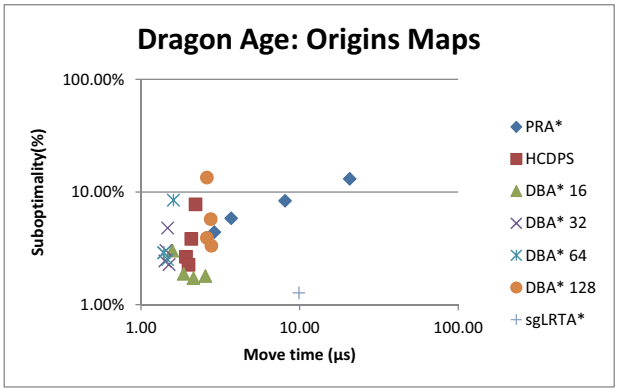

Figure 1: Suboptimality versus Move Time

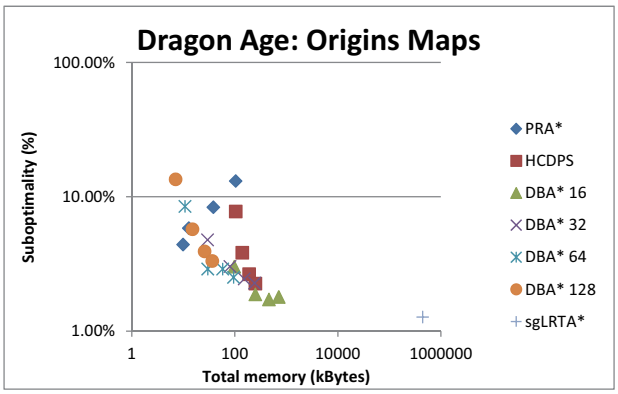

Figure 2: Suboptimality versus Total Memory Used

Conclusions and Future Work

DBA* has lower suboptimality than other abstraction-based approaches with a faster response time. It represents a quality balance and integration of the best features of previous algorithms. Future work includes defining techniques for efficiently updating pre-computed data to reflect grid changes.

References

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

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

Hernández, C., and Baier, J. A. 2011. Real-time heuristic search with depression avoidance. In *IJCAI*, 578–583.

Lawrence, R., and Bulitko, V. 2012. Database-driven real-time heuristic search in video-game pathfinding. *IEEE TCAIG* PP(99).

Sturtevant, N. 2007. Memory-efficient abstractions for pathfinding. In *AIIDE*, 31–36.