A* Variants for Optimal Multi-Agent Pathfinding

Meir Goldenberg
ISE Department
Ben-Gurion University, Israel
mgoldenbe@gmail.com

Ariel Felner, Roni Stern, Guni Sharon
ISE Department
Ben-Gurion University, Israel
felner@bgu.ac.il,
{roni.stern, gunisharon}@gmail.com

Jonathan Schaeffer
CS Department
University of Alberta, Canada
jonathan@cs.ulaberta.ca

Abstract

Several variants of A* have been recently proposed for finding optimal solutions for the multi-agent pathfinding (MAPF) problem. We describe the application of the new enhanced partial-expansion technique to MAPF, show how pattern databases can be applied on top of this technique and compare the different A* variants experimentally.

Introduction

Most recently, solving the multi-agent pathfinding (MAPF) problem optimally has gained much attention, resulting in several new algorithms (Sharon et al. 2011; 2012) and variants of A* (Standley 2010; Felner et al. 2012). In this paper, we focus on the new variants of A*.

A recently developed technique called *enhanced partial expansion A** (EPEA*) (Felner et al. 2012), uses domain-specific knowledge to avoid the generation of nodes whose f-value is greater than the cost of the optimal solution. The main contribution of our paper is the detailed presentation of a way to apply EPEA* to MAPF. Furthermore, our current method of applying EPEA* is generalized to allow using pattern databases (PDBs) on top of EPEA*. Effectively applying PDBs to MAPF is a challenging task. To the best of our knowledge, we report an application of PDBs to MAPF for the first time in the literature. The last contribution of ours is a new variant of A* for optimally solving MAPF, which is a hybrid of the *operator decomposition* (OD) technique of (Standley 2010) and the *partial expansion* technique of (Yoshizumi, Miura, and Ishida 2000) (the latter is referred to as *basic partial expansion* (BPE) hereafter).

Multi-agent pathfinding: formal definition

We focus on the following commonly used variant of MAPF (Standley 2010; Sharon et al. 2011; 2012). The *input* to MAPF is: **(1)** A graph $G(V, E)$ and **(2)** k agents, each labeled coupled with a start and a goal location. At a given time step, each agent can perform a `move` action to a neighboring location or can `wait` (stay idle) at its current location. Each vertex can be occupied by at most one agent at a given time. In addition, if a and b are neighboring vertices, two different agents cannot simultaneously traverse the connecting edge in opposite directions (from a to b and from b to a). Agents are allowed to *follow* each other, i.e., agent a_i could move from x to y at the same time as agent a_j moves from y to z.

The task is to find a minimum-cost sequence of $\{move, wait\}$ actions for each agent such that each agent will be located in its goal position. Both `move` and `wait` actions cost 1.0, except for the case when the `wait` action is applied at an agent's goal location and costs zero. If an agent waits m times at its goal location and then moves, the cost of that move is $m + 1$.

Application of enhanced partial expansion to MAPF

*Enhanced partial expansion A** (EPEA*) (Felner et al. 2012) uses *a priori* domain knowledge to avoid generating *surplus* nodes (i.e. nodes whose f-value is greater than the cost of the optimal solution) as follows. First, distinction is made between the regular f-value ($g + h$) of a node n, called its *static value* and denoted by $f(n)$ (small f), and the value currently stored for n in the open list, called the *stored value* of n and denoted by $F(n)$ (capital F). Initially $F(n) = f(n)$. When expanding a node n, EPEA* generates only the children n_c with $f(n_c) = F(n)$. This is achieved by formalizing the domain-specific knowledge as *operator selection function* (OSF). The difference $F(n) - f(n)$ is denoted by Δf. We refer the reader to (Felner et al. 2012) for further details. In this section we describe an OSF for MAPF that is generalized to allow for the usage of PDBs.

We define a *composite* agent (CA) as a group of agents. A special data structure, called the *composite agent operators structure* (CAOS) contains, for each possible state of each composite agent all legal (i.e. without collisions of agents within the CA) operators ordered by Δf. Since this data structure can be very large, we compute it on demand using the technique of (Felner and Adler 2005).

Figure 1 (left) shows an example with 3 composite agents with 5, 3 and 4 operators, respectively. Suppose that the node being expanded has the static value of $f = 2$ and the stored value of $F = 10$. The OSF will need to find all combinations of operators for composite agents with the sum of Δf's equal to $10 - 2 = 8$. In Figure 1 (left), the first such

CA_1	CA_2	CA_3
0	0	0
0	**1**	1
3	3	**4**
3		4
5		

		Unique Nodes Generated, $\times 10^3$				Run-Time, ms			
k	Ins	ODA*	BPEODA*	EPEA*	EPEA*+PDBs	ODA*	BPEODA*	EPEA*	EPEA*+PDBs
2-6	1	335.34	105.14	9.94	**9.31**	2,153	1,803	**278**	354
7-8	25	219.11	67.04	7.82	**4.41**	1,637	1,312	335	**232**
9-10	13	705.76	211.54	17.57	**10.01**	16,660	8,846	3,062	**1,089**

Figure 1: Left: Computing OSF for MAPF Right: comparison of A* variants

choice is shown in solid.

Effectively, we have to solve the following combinatorial enumeration problem: given k bins with balls each tagged with a number, enumerate all ways of choosing one ball from each bin, such that the total sum of the numbers on the balls is Δf. Since this problem is exponential in the number of bins (which corresponds to the number of composite agents) and is solved for every expansion, it is critical that this problem be solved efficiently.

Our solution is a simple recursive procedure with several enhancements (which we hope to described in a full version of the paper). For the above example with three bins, this procedure tries each of the choices for the first bin and performs a recursive call with the update sum for the remaining bins. For example, when the third choice (which is the first operator with individual $\Delta f = 3$) is tried for the first (i.e. left-most) bin, the remaining two bins have to contribute $8 - 3 = 5$ to the sum. Therefore, we can use a recursive call to our procedure for the remaining two bins and the required sum of 5.

PDBs for MAPF

PDBs for MAPF can be effective only if they are built for agents that participate in many conflicts. However, this information is not known *a priori*. For example, given an instance with 20 agents, a special method is needed to find an effective way to pair up the agents for pairwise databases. We overcome this problem by using the *independence detection* (ID) framework of (Standley 2010). Whenever ID joins two agents into a group, we use this information to build pairwise PDBs at later stages of ID.

Basic partial expansion (BPE) A* with operator decomposition

BPEA* (our term for the technique of (Yoshizumi, Miura, and Ishida 2000)) is generic and can be applied on top of any procedure for neighbor generation. In particular, BPEA* can be applied on top of *operator decomposition* due to (Standley 2010) as follows. Each intermediate note is a successor of some standard node. The most immediate such standard node is called the *standard predecessor* of the intermediate node. When a node n is expanded there are two cases: **(1)** if n is standard, then the regular BPE condition applies, otherwise **(2)** a child n_c is kept only if it's f-value is equal to the stored value of the standard predecessor of the node being expanded. We call this algorithm BPEODA*.

Experimental results

Figure 1 (right) shows the comparison of different algorithms on a four-connected 8x8 grid with no obstacles with various numbers of agents. Except for the most trivial instances A* and BPEA* were not able to solve the problem within the allocated resources (two minutes and two gigabytes of memory per instance). First, as reported by (Standley 2010), ODA* is faster than A*. Second, BPEA* is faster than A*, but it suffers from the overhead of generating the surplus nodes in order to prune them away. Third, BPEODA* significantly outperforms both A* and ODA* due to maintaining a much smaller open list resulting in cheaper open list operations. EPEA* is faster than all these variants. In addition, for hard instances, PDBs give a significant (up to three times on average) improvement on top of EPEA*. Since the PDBs are instance-dependent, their building times cannot be amortized over all instances. However, we see that building PDBs well pays off for the hard instances. For easier instances, EPEA*+PDBs was still the best algorithm in terms of nodes, but not in terms of time.

Conclusions

We presented a study of several variants of A* for optimally solving the multi-agent pathfinding (MAPF) problem. We hope to present several important and hitherto uncovered insights about the techniques of (Standley 2010) in the full version of the paper.

Acknowledgements

This research was supported by the Israeli Science Foundation (ISF) grant 305/09 to Ariel Felner and the Natural Sciences and Engineering Research Council of Canada grant to Jonathan Schaeffer.

References

Felner, A., and Adler, A. 2005. Solving the 24-puzzle with instance dependent pattern databases. In *SARA-05*, 248–260.

Felner, A.; Goldenberg, M.; Sharon, G.; Stutervant, N.; Stern, R.; Beja, T.; Schaeffer, J.; and Holte, R. 2012. Partial-expansion a* with selective node generation. In *to appear in AAAI*.

Sharon, G.; Stern, R.; Goldenberg, M.; and Felner, A. 2011. The increasing cost tree search for optimal multi-agent pathfinding. In *IJCAI*, 662–667.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2012. Conflict-based search for optimal multi-agent path finding. In *to appear in AAAI*.

Standley, T. 2010. Finding optimal solutions to cooperative pathfinding problems. In *AAAI*, 173–178.

Yoshizumi, T.; Miura, T.; and Ishida, T. 2000. A* with partial expansion for large branching factor problems. In *AAAI/IAAI*, 923–929.