rough draft Finding Optimal Solutions to the Multi-Agent Pathfinding Problem Using Heuristic Search

Trevor Standley
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90095
tstand@cs.ucla.edu

Abstract

In the multi-agent pathfinding problem, multiple agents must cooperate to plan non-interfering paths that bring each agent from its current state to its goal state. We present the first practical and admissible algorithm for the multi-agent pathfinding problem. First, we introduce a general technique called operator decomposition, which can be used to reduce the branching factors of many combinatorial algorithms, including an algorithm for multi-agent pathfinding. Next, we show that partial expansion can be used to allow each agent to consider only a subset of possible moves at every step without sacrificing completeness. We then show how the independent subproblems common in instances of the multi-agent pathfinding problem can be exploited. Finally, we show empirically that each of the three techniques significantly improves the performance of the standard admissible algorithm for the multi-agent pathfinding problem.

Introduction

Pathfinding, the problem of planning a route to a destination that avoids obstacles, is a classic problem in the design of video game AI. When only a single path is needed, the problem is effectively solved using the classic A* algorithm. But when multiple paths must be executed simultaneously, care must be taken to avoid computing conflicting paths. While multi-agent pathfinding is becoming increasingly important in modern video games, it has many applications outside of entertainment such as robotics, aviation, and vehicle routing (Wang and Botea 2008; Svestka and Overmars 1996).

Unfortunately, it has been shown PSPACE-hard to compute a set of non-conflicting paths with the shortest total length for a given instance in a gridworld (Hopcroft, Schwartz, and Sharir 1984), and the standard admissible algorithm for this problem has a branching factor that is exponential in the number of agents (Silver 2005). It is for these reasons that modern research on this problem focuses on finding a good set of paths rather than an optimal set of paths. However, as we will see, the types of problems generally encountered in some multi-agent pathfinding domains are not intractable, and this paper shows that many nontrivial instances can be solved exactly in milliseconds.

This paper is divided into the following sections. First, we describe related work. Then, we describe one particular formulation of the multi-agent pathfinding problem. Next, we elaborate on the standard complete algorithm and our three improvements. Finally, we present our experimental results and conclusion.

Related Work

In (Silver 2005), the author describes the general algorithm employed by the majority of video games, Local Repair A* (LRA*). In LRA*, paths are computed individually for each agent, and conflicts are not discovered until execution. If the algorithm discovers that following a plan one step further results in a conflict, the algorithm re-plans the paths of the conflicting agents, disallowing only the moves that would immediately result in conflict. Not only are the plans that this algorithm computes vastly suboptimal, but the algorithm often results in cyclic dependences and deadlock, and therefore many agents don't ever make it to their goals in many real-world cases. The author's solution to this problem, Windowed Hierarchical Cooperative A* (WHCA*), focuses on reducing the number of future re-plans by creating a reservation table for some number of moves into the future. The algorithm chooses an ordering of agents, and plans a path for each agent using A*, taking care to avoid conflicts with previously computed paths by checking against the reservation table. While the paths computed by WHCA* are significantly shorter than those computed by LRA*, and nearly all agents eventually reach their destinations, the greedy nature of this algorithm results in paths that are still approximately 20% worse than the optimal paths for large numbers of agents on random maps. Furthermore, on random maps, about 2% of agents still fail to reach their destinations, thus requiring human intervention.

Other attempts such as those in (Wang and Botea 2008) and (Jansen and Sturtevant 2008) establish a direction for every position that an agent can occupy, and encourage or require any agent to move in that general direction at every step. These methods reduce the chance of computing conflicting paths by creating the analog of traffic laws for the agents to follow. While these methods are effective at reducing the number of incompatible paths, they highly limit the types of paths that agents can follow, and thus virtually guarantee that the paths found will be sub optimal. For example, there is no reason for a driver only concerned with quickly reaching his destination (like an ambulance) to drive on the

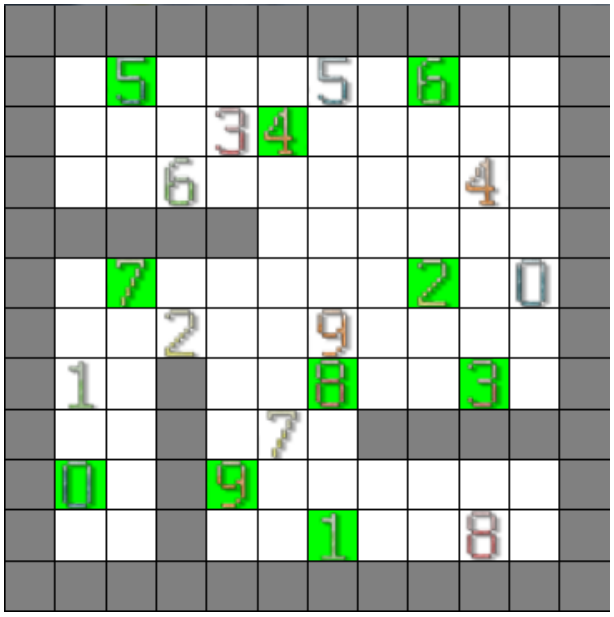

Figure 1: A state. Numbered cells represent agents. Green cells represent destinations.

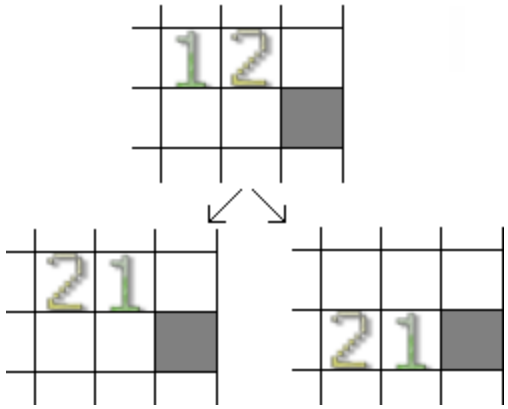

Figure 2: Disallowed state transitions. It is illegal for any two agents to switch locations (left) or to cross through each other (right)

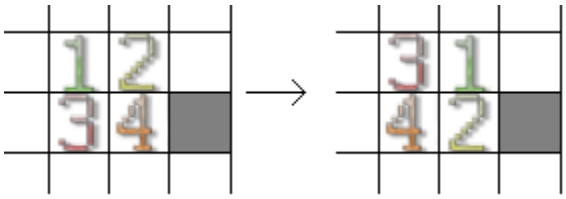

Figure 3: In this formulation of the multi-agent pathfinding problem moves such as this rotation, that result in each agent occupying a previously occupied cell, can be legal.

right side of the road if he knows that nobody is going to use the left side. And as paths are still computed one agent at a time without regard to the consequences a particular choice of path can have on the paths available to future agents, this strategy also suffers from the problem that human intervention is often necessary to ensure that each agent reaches its destination.

Problem Formulation

There are many distinct types of multi-agent pathfinding all can be solved using a variant of our algorithm. One example is planning the motions of robotic arms in a factory to each accomplish a separate goal without moving into each other. Another example is planning the schedules of trains in a railroad network without scheduling a pair of trains on a collision course. For the sake of clarity and simplicity, however, the testbed for the algorithm as presented will be an eight-connected gridworld like the one in Figure 1.

In the eight-connected gridworld variant, each agent occupies a single cell of the gridworld, and has a unique destination. Agents can move simultaneously to one of their eight adjacent cells or choose to stay at their current cell during a single timestep. The cost of a plan is the total number of timesteps each agent spends away from its goal. However, it is possible for the moves of two agents to conflict, and such conflicts must be made illegal.

Obviously, it should not be legal for any agents to occupy the same cell at the end of any timestep. In addition, each of the transitions depicted in Figure 2 should not be allowed to occur in a single step because it would require agents to pass through each other. The transition depicted in Figure 3, however, is legal[1] and leads to motions that look more coordinated. In fact, we know of no other practical algorithm that can exploit such moves.

The Standard Admissible Algorithm

The standard admissible algorithm for this problem is A* search with the following problem representation: a state is a tuple of grid locations, one per agent, and there are potentially 9^n legal operators for each state, one of $(N, NE, E, SE, S, SW, W, NW, 0)$ for each agent.

The algorithm (Algorithm 1) starts with a node containing the initial state on the open list, and at every iteration removes the most promising node on the open list, puts it onto the closed list, and generates each of its possibly 9^n legal successors along with each calculated heuristic function[2]. The algorithm then places each successor not yet explored onto the open list. This process repeats until a node that contains a goal state is removed from the open list.

While this method can be used to solve some trivial problem instances, each iteration can add approximately 59k

[1] It is not known whether the problem is PSPACE complete when transitions of the kind depicted in Figure 3 are allowed, but the legality of such transitions is irrelevant to the complexity of the algorithms.

[2] The section on partial expansion goes into detail about the heuristic.

Algorithm 1 The Standard Algorithm (A*)

```
1: open ← {(start, h(start))}
2: closed ← {}
3: while open ≠ {} do
4:    x ← deletemin(open)
5:    closed ←add x
6:    if x is goal then
7:       reconstruct the path and return
8:    end if
9:    for each legal successor S do
10:      if S ∉ closed then
11:         g(s) ← g(x) + numNotStopped(S)
12:         f(S) ← h(S) + g(S)
13:         remove any copy S' from open
14:         open ←add min((S, f(S)), (S', f(S')))
15:      end if
16:   end for
17: end while
```

nodes to the open list[3] when run on an instance with only 5 agents. The algorithm must add every possible successor of a node onto the open list, and it is common for a successor to be obviously bad to a human observer. This suggests that the standard algorithm can be improved.

Operator Decomposition

It would be ideal if the algorithm could consider the possible moves of each agent individually (one per node). If this could be accomplished, the branching factor of the algorithm would go from 9^n to just 9. But because each agent must be considered in turn, this reduction in branching factor would come at the cost of a linear increase in depth. So if the algorithm did no pruning this would be a fair trade (i.e $O((9^n)^d) = O(9^{nd})$), however this modification could give the algorithm the ability to never consider the vast majority of a standard node's 9^n children.

Unfortunately, the naive method for doing this is inadmissible[4]. Consider for example the situation depicted in Figure 4. If the algorithm considers agent 1 first, then it will conclude that it has only two legal moves $(N, 0)$. However, the shortest path to the goal requires both agents 1 and 2 to make the move E during the first timestep. One might think to try every possible ordering of agents, but this would not only lead to the algorithm considering $9n!$ children for a standard node, but also prohibit the algorithm from making rotational moves such as the ones in Figure 3. Note that the standard algorithm would consider the correct child during the course of its execution wile considering only 9^n children.

The problem with the naive way of choosing actions one agent at a time is that it is being too strict. It is restricting the types of moves that a pair of agents can make midway between a true timestep. If instead the algorithm waited until the moves of each agent had been considered and then checked to make sure that the resulting nodes represented the legal successors of a standard node, then the algorithm would not suffer from this problem.

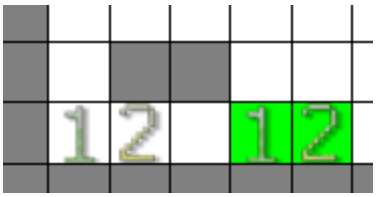

Figure 4: The failure of naive operator decomposition.

In the example of Figure 4, the algorithm would consider the three moves $(N, E, 0)$ of agent 1 on the start state first, and a node resulting from taking the move E would be placed onto the open list (note that at this point agent 1 and agent 2 are overlapping). Then, when that node was chosen for expansion, agent 2 would consider the move E. The algorithm could then check to see that the move pair $[1 : E, 2 : E]$ was legal (because *both* agents have made moves) before placing the resulting node onto the open list. The child resulting from agent 2 making the move 0 would also be checked, and then discarded (because the move pair $[1 : E, 2 : 0]$ results in two agents overlapping at the end of a timestep.)

The above process is capable of generating every legal successor of a standard node for any chosen order of agents, and will typically generate the correct node before the majority of others[5]. The process is thus admissible for all choices of agent order, and constitutes a substantial improvement on the standard algorithm.

It is actually possible to do better. The current process waits until a node resulting from every agent making a move is generated before checking that the moves made to get to that node are consistent. However it is possible to prune bad children early. If the move made by an agent is incompatible with a move made by a previously considered agent, no move made by a subsequent agent can fix the problem. Therefore, the algorithm should require that the moves made by an agent be consistent with the moves made by every previously considered agent, but not require that a move made by an agent be consistent with the moves of agents yet to be chosen[6].

This modified process is also capable of generating every legal successor of a standard node regardless of the choice of agent order, and so the process is also admissible, however this process is preferred to the latter because it prunes illegal

[3]For example if there are no obstacles, and each agent is separated then the algorithm will exhibit this behavior, however such problems become easy when operator decomposition is used.

[4]The algorithm also becomes incomplete when rotational moves are allowed.

[5]If there is no solution to the problem, no method outlined in this paper will have a significant impact on the discovery of this fact.

[6]This process is actually a method for solving a constraint satisfaction problem; each agent must be assigned a move, and there are constraints between pairs of moves. More sophisticated methods for solving such constraint satisfaction problems are not considered in this paper

nodes at higher depths.

Since agents can be considered in any order, it is natural to ask which order should be preferred. Empirically a dynamic agent order that chooses agents first by the lowest number of legal moves, second by the fewest neighbors, and third by the heuristic value of the best move, seems to work well.

The above process can be viewed as a change in problem representation. Instead of representing a state as the position of each agent, and legal operators as a tuple of moves, we represent each state as not only the position of every agent, but also keep track of which agents have already moved during this timestep and what move they made. An operator becomes a pair of an agent, and a move. An operator is legal if the agent has not yet moved, and the move does not conflict with the moves of any agent that has already moved. Search in this modified problem space is equivalent to search in the standard space, and a series of n moves corresponds to a single move in the standard space. If any move must be executed in the standard space to ensure an optimal solution, it will be executed as a series of n moves in the new search space because of the admissibility of A* search.

In fact, this technique can be seen as an instance of a more general technique. As a toy example, consider the single agent pathfinding problem. Instead of a move being one of $(N, NE, E, SE, S, SW, W, NW, 0)$, it can first be the the choice between $(N, NE, E), (SE, S, SW)$, and $(W, NW, 0)$ and then be the choice between the three remaining options. In this example the branching factor went from 9 to 3, but the depth of a solution doubled. We call this technique operator decomposition because the standard operators are decomposed into a series of operators. There is no clear advantage to this approach unless pruning can be done partway between standard moves, or standard heuristic evaluation functions can be made to have sub-timestep resolution. Fortunately, the multi-agent pathfinding problem has both properties as the value of the heuristic can change after every agent makes a move, and the moves of agents can rule out possible moves of subsequent agents. Thus operator decomposition has a drastic impact on the performance of the standard algorithm.

Considering each agent one at a time has other benefits as well.

Partial Expansion

The operator decomposition technique is useful for reducing the number of moves considered by the standard algorithm, but even the moves considered by the algorithm with operator decomposition are often unnecessary. This section describes a way to further reduce the number of considered moves if operator decomposition is applied.

In (Yoshizumi, Miura, and Ishida 2000), the authors outline a technique helpful for reducing the memory requirements of problems with large branching factors. The technique works like this: when a node is removed from the open list for expansion, the algorithm generates each possible child and determines the $k_1 + 1$ children with the lowest heuristic values. The algorithm only puts the k_1 best children onto the open list in the hopes that one will lead to the optimal solution. The rest of the children are discarded and the original node is put back onto the open list with the heuristic value of the $(k_1 + 1)^{st}$ child for possible re-expansion. If the node is re-removed from the open list, then the next best k_2+1 children of that node are determined and the best k_2 of those are put onto the open list and so on. Thus, even if any optimal path must go through a child that does not initially look promising (and therefore is not initially put onto the open list), the parent will be reconsidered, and the children will be put back onto the open list.

During this process, nodes are often regenerated and their heuristics are recomputed. If memory is the bottleneck, then this is not very important, however for video games and other real time applications, memory is often a secondary consideration. Furthermore, the branching factor of the algorithm with operator decomposition is only 9. Thus it may seem that the idea of partial expansion provides little benefit for a problem such as this, however if we use operator decomposition for this problem it is possible to precompute which $k + 1$ children will have the lowest heuristic values and store the result for real-time use.

It is now appropriate to discuss the type of heuristic function used for this formulation of the multi-agent pathfinding problem. Ideally, the heuristic would be the sum of the true individual distances from the current position of each agent to the agent's goal position (i.e. ignoring the positions of other agents), and indeed it is possible to arrive at this quantity with very little overhead. The process, known as Reverse Resumable A* (RRA*), is outlined in detail in (Silver 2005), but briefly the idea is to run the single agent A* algorithm backward from each agent's goal position caching the greedy distance to the goal of every tile it encounters. When the search yields the desired quantity (the distance between the current agent's position during the execution of the algorithm to the agent's goal position), RRA* is paused but can be resumed when another needed quantity is not found in the cache.

It is not complicated to update RRA* to store along with the true distance to the goal for a tile in the cache, a pointer in the direction toward the goal. It can even be updated to store the $k + 1$ directions that result in having the lowest distance to the goal, thus yielding the desired quantities.

Now an agent can know exactly which moves result in the $k + 1$ states with the best heuristic values, and can generate only those nodes in time linear in k.

Our implementation uses $k_1 = 1$, $k_2 = 3$, and $k_3 = 5$, and thus only has to store pointers for the best 5 moves using 15 bits total. With this scheme, the ratio of the final size of the open list to the final size of the closed list is reduced from about 7 to less than 3, resulting in a substantial constant improvement in runtime, as well as a large reduction in memory use.

Independent Subproblems

Although partial expansion and operator decomposition allow the algorithm not to consider a substantial portion of the search space, the resulting algorithm is still exponential in the number of agents.

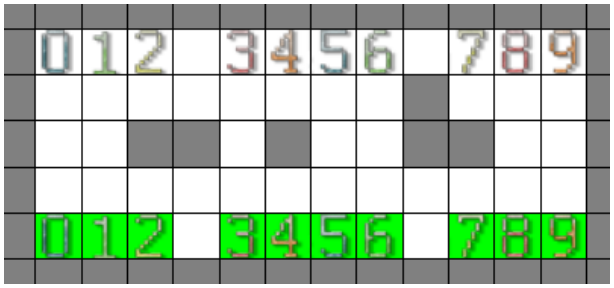

Figure 5: If the agents are grouped like so: (0,1,2) (3,4,5,6) (7,8,9), paths for each group can be computed separately.

Consider as a motivating example, the problem of planning world automobile traffic. The problem of planning traffic in Australlia is independant from the problem of

It can often be decided, however, that the path that can be taken for a group of agents is irrelevant to the path of another group. In other words, if we have exhaustive and mutually exclusive subsets of the set of all agents in a problem, and we discover that the shortest paths bringing each subset of agents to their respective goals do not interfear with one another, then we can construct a shortest path bringing every agent from its start state to its goal state. If such subsets can be discovered, the running time of the algorithm becomes $O(9^{kd})$ where k is the size of the largest subset (group).

Thorough analysis of the problem depicted in Figure 1 shows that there are seven independent multi-agent pathfinding problems present in this example (i.e. the greedy path taken by agent 5 is compatible with the paths taken by each other agent, while the greedy path taken by agent 3 will be incompatible with the greedy path taken by agent 0). Figure 5 makes this point more clear.

In order to take advantage of such independence, we developed Algorithm 2 which discovers independent subproblems by assuming each path will be independent, and jointly recomputing the conflicting paths if the assumption fails. Because the planning time is exponential in the size of the largest group, the final re-plan dominates the computation, and so the time to compute initial conflicting plans is unimportant.

Algorithm 2 Compute Paths Independently

```
1: groups ← one group per agent
2: plan paths for each group
3: repeat
4:     merge groups with conflicting paths
5:     re-plan merged groups
6: until no conflicts occur
7: return total plan
```

Experiments

Experiments were run on an Intel Core 2 @2.4 Ghz with 8GB of ram. Two hundred random 32x32 grids were generated with random obstacles (each cell is not passable with

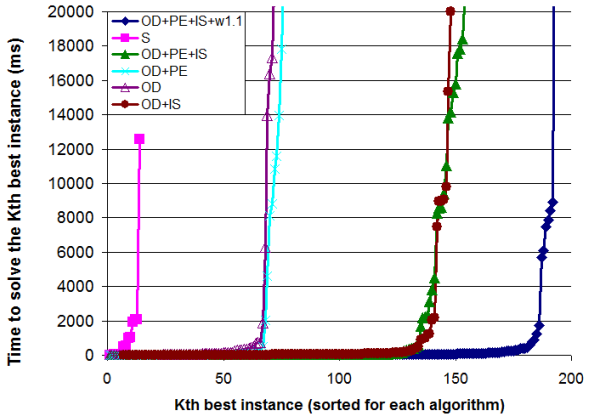

Figure 6: The results of each algorithm on 200 randomly generated instances.

20% probability), and a random number of units (between 2 and 30) each in random locations with random destinations. The standard algorithm (S), and several combinations of improovments (operator decomposition (OD), partial expansion (PE), independent subproblems(IS)) were run. Figure 6 shows the running time of each algorithm on instances that took less than twenty seconds to solve, sorted by ascending time.

The data shows that adding each technique results in improved performance. The data also shows that when all three techniques are employed, more than half of the instances become easy, and can be solved exactly in under a second. If one is willing to settle for solutions guaranteed to be within 10% of the optimal solution (typically they will be much better), 90% of the instances can be solved in under a second by using a weighted heuristic function with $w = 1.1$.

It may seem from looking at the plots that partial expansion offers little benefit, but the average runtime the algorithm with partial expansion is almost four times better when solutions are found.

Conclusion

This paper describes the general problem of multi-agent pathfinding. The pitfalls of modern inadmissible algorithms for the multi-agent pathfinding problem were also discussed, and three improvements on the standard admissible algorithm were presented. The first, node splitting, effectively reduced the branching factor of the problem at the cost of depth, and was effective at allowing the algorithm to consider only a small fraction of the children of each node. The second, partial expansion, was able to further reduce the number of children generated to a certain depth by a substantial constant factor. The third, independent subproblems, sometimes allowed the paths of groups of agents to be computed independently without sacrificing completeness.

A concrete variant of the multi-agent pathfinding problem modeled after typical video game designs was presented, and used to test the presented algorithms as well as clarify the algorithm descriptions. The results of the tests show

that the standard admissible algorithm can be substantially improved. And when the three techniques are used in combination, the algorithm can be made practical for small numbers of agents.

Because this variant of the standard algorithm is still a variant of A* search, weighted A* or a number of other techniques can be used to approximately solve larger instances, with tighter quality guarantees than existing algorithms.

References

Hopcroft, J.; Schwartz, J.; and Sharir, M. 1984. On the Complexity of Motion Planning for Multiple Independent Objects; PSPACE- Hardness of the "Warehouseman's Problem". *The International Journal of Robotics Research* 3(4):76–88.

Jansen, R., and Sturtevant, N. 2008. A new approach to cooperative pathfinding. In *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, 1401–1404. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Silver, D. 2005. Cooperative pathfinding. In Young, R. M., and Laird, J. E., eds., *AIIDE*, 117–122. AAAI Press.

Svestka, P., and Overmars, M. H. 1996. Coordinated path planning for multiple robots. Technical Report UU-CS-1996-43, Department of Information and Computing Sciences, Utrecht University.

Wang, K.-H. C., and Botea, A. 2008. Fast and memory-efficient multi-agent pathfinding. *AAAI 2008*.

Yoshizumi, T.; Miura, T.; and Ishida, T. 2000. A* with partial expansion for large branching factor problems. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, 923–929. AAAI Press / The MIT Press.