Automatic Algorithm Selection in Multi-Agent Pathfinding

Devon Sigurdson
Vadim Bulitko
University of Alberta
dbsigurd, bulitko@ualberta.ca

William Yeoh
Washington University in St. Louis
wyeoh@wustl.edu

Carlos Hernández
Universidad Andres Bello
carlos.hernandez.u@unab.cl

Sven Koenig
University of Southern California
skoenig@usc.edu

Abstract

In a multi-agent pathfinding (MAPF) problem, agents need to navigate from their start to their goal locations without colliding into each other. There are various MAPF algorithms, including Windowed Hierarchical Cooperative A*, Flow Annotated Replanning, and Bounded Multi-Agent A*. It is often the case that there is no single algorithm that dominates all MAPF instances. Therefore, in this paper, we investigate the use of deep learning to automatically select the best MAPF algorithm from a portfolio of algorithms for a given MAPF problem instance. Empirical results show that our automatic algorithm selection approach, which uses an off-the-shelf convolutional neural network, is able to outperform any individual MAPF algorithm in our portfolio.

1 Introduction

Pathfinding is a common task for many applications, such as robotics, transportation, and video games. Often, paths need to be planned for multiple game characters, for example, when each one of several game characters needs to move from its current location to a given goal location. Single-agent pathfinding is a simpler problem since shortest single-agent paths can be found optimally in polynomial time with search methods like A* (Hart, Nilsson, and Raphael 1968). Multi-agent pathfinding (MAPF) is a more complex problem because one needs to avoid collisions among the agents. Artificial intelligence and robotics have developed a large number of MAPF algorithms. Suboptimal complete MAPF algorithms often implement specific movement rules for agents that guarantee completeness and run fast since they avoid search. An example is Push and Swap/Rotate (Luna and Bekris 2011; de Wilde, ter Mors, and Witteveen 2013). Some optimal or bounded-suboptimal MAPF algorithms reduce the MAPF problem to other combinatorial problems. Examples include reductions to CSP (Ryan 2010), SAT (Surynek 2012), ILP (Yu and LaValle 2013a), and ASP (Erdem et al. 2013). Others are based on heuristic search. Examples are M* (Wagner and Choset 2015), Conflict-Based Search (Sharon et al. 2015; Boyarski et al. 2015; Cohen, Uras, and Koenig 2015) and many others. A longer recent overview of MAPF algorithms is provided by Felner et al. (2017).

While finding collision-free paths for multiple agents can be done in polynomial time (Kornhauser, Miller, and Spirakis 1984), in practice, these paths should also be reasonably short. Finding collision-free paths that optimize solution quality measured as makespan (the largest arrival time of any agent at its goal location) or flowtime (the sum of the arrival times of all agents at their goal locations) is NP-hard (Yu and LaValle 2013b). In some cases, even approximating the optimal solution quality is NP-hard (Ma et al. 2016). Unfortunately, paths with a reasonably good solution quality often need to be found quickly (e.g., when planning paths for game characters online). Researchers in artificial intelligence have developed a variety of MAPF algorithms that can be used for this purpose, such as Windowed Hierarchical A* (Silver 2005), Flow Annotated Replanning (Wang and Botea 2008) and Bounded Multi-agent A* (Sigurdson et al. 2018). To the best of our knowledge, none of the MAPF algorithms universally dominates all others.

As a starting point of determining a good MAPF algorithm, we are primarily interested in ensuring that all agents are successful, in the sense that they reach their goal locations by a given deadline. Unfortunately, it has recently been shown to be NP-hard to determine whether all agents can be successful on arbitrary graphs (Ma et al. 2018). We thus use real-time MAPF algorithms that do not guarantee that all agents are successful even if this is possible and evaluate the algorithms by the number of agents that are successful. One could experimentally determine the best MAPF algorithm over a representative set of MAPF instances but one can do better by asking which MAPF algorithm to use when, a question that has been studied more generally in the context of the algorithm selection problem (Rice 1976). This decision also has to be made quickly, which suggests using classification algorithms from machine learning to define a fast-to-calculate mapping from features of the MAPF instance to the best performing MAPF algorithm. Consequently, the contribution of our research is applying automated algorithm selection techniques to increase completion rate over using a single algorithm for every problem.

2 Problem Formulation

An automatic algorithm selection optimization problem (Rice 1976) is defined by a tuple $\langle \mathcal{I}, \mathcal{P}, Q \rangle$, where $\mathcal{I} = \{i_1, i_2, \ldots\}$ is a set of problem instances; $\mathcal{P} = \{p_1, p_2, \ldots\}$ is a portfolio of algorithms that can be used to solve each instance $i \in \mathcal{I}$; and $Q : \mathcal{P} \times \mathcal{I} \to \mathbb{R}$ is a function that returns the quality of the solution found by an algorithm $p \in \mathcal{P}$ when solving problem instance $i \in \mathcal{I}$.

A solution to this problem is a mapping $\pi : \mathcal{I} \to \mathcal{P}$ that maps each problem instance $i \in \mathcal{I}$ to an algorithm $p \in \mathcal{P}$. The quality of a solution π is the sum of the qualities of the solutions found by the algorithm prescribed by π for each problem instance:

$$Q(\pi) = \sum_{i \in \mathcal{I}} Q(\pi(i), i). \tag{1}$$

An optimal solution is one that maximizes this value:

$$\pi^* = \operatorname*{argmax}_{\pi} Q(\pi). \tag{2}$$

Our set of problem instances $\mathcal{I}$ are multi-agent pathfinding (MAPF) problems on video game maps. We consider a portfolio $\mathcal{P}$ of MAPF algorithms as candidate algorithms, and we consider *completion rate*, defined as the number of agents that successfully reach their goals within a time limit (Silver 2005; Wang and Botea 2008), as our quality metric Q[1]. We break ties in Q in favor of algorithms that have short *distances traveled*, defined as the sum of distances travelled over all agents (Silver 2005), followed by algorithms that have small *goal achievement time*, defined as the average wall-clock time it takes for agents to reach their goal from the start of the problem (Silver 2005). Goal achievement times are recalculated when an agent returns to their goal and are undefined if the agent is not in their goal.

3 MAPF Problem

We provide a description of the *multi-agent pathfinding* (MAPF) problem that closely follows the description by Sigurdson et al. (2018). A MAPF problem is defined by a pair (G, A), where $G = (N, E, c, h)$ is a undirected weighted graph of nodes N that are connected to each other by edges $E \subset N \times N$. The set of edges includes self-loops, that is, $\forall n \in N : (n, n) \in E$, which allows all agents to remain on their current node (i.e., wait). We assume that all edge weights $c : E \to \mathbb{R}$ are strictly positive except self-loop edges, which have a weight of 0. The edge weights are symmetrical $\forall (n, n') \in E : c(n, n') = c(n', n)$. $A = \{a_1, \ldots, a_n\}$ is a set of NPC agents, where each agent $a_i \in A$ is specified by a pair $(n^i_{\text{start}}, n^i_{\text{goal}})$ that indicates its start node n^i_{start} and its goal node n^i_{goal}. Each goal is reachable from the start node. The graph is also safely traversable by virtue of being undirected. An estimate of shortest travel distance between any two nodes, the *heuristic* $h : N \times N \to \mathbb{R}$, is available to each agent. We use the shorthand $h(n)$ for $h(n, n^i_{\text{goal}})$ if the goal is understood in the context. An agent may modify h as it sees fits but does not share the modified version with other agents.

In our model, time advances in discrete steps. At time step t, each agent a_i occupies a node $n^i \in N$, also referred to as n^i_{current} when talking about a specific agent's location. When pathfinding, each agent generates a set of moves it plans to execute from its current state. Agents provide these moves to a controller that attempts to have the agent execute its plans one step at a time. Plans are represented as a set of node pairs $P = \{(n, n')\}$. The pairs represent agent's planned actions (i.e., edge traversals) meaning that when the agent is on node n it intends to go to a neighboring node n' by traversing the edge $(n, n') \in E$.

In the event that the agent's plan is not executable, either because another agent is occupying the node where it wishes to move or because the plan does not have an action planned for the agent's current node, the agent waits in its current node (i.e., traverses the self loop). Agents can traverse edges with the following restrictions: (i) two agents cannot swap locations in a single time step and (ii) each node can be occupied by at most one agent at any time.

As common in video game pathfinding literature, our search graphs in this paper are based on rectangular grids with each grid cell being a single node in the graph (Sturtevant 2012). Each grid cell has up to eight immediate neighbors. It is connected to them via cardinal edges of cost 1 and diagonal edges of cost $\sqrt{2}$.

4 MAPF Algorithms

While researchers have proposed a number of algorithms to solve MAPF problems (Wang and Botea 2011; Sharon et al. 2015; de Wilde, ter Mors, and Witteveen 2013), in this paper, we focus on using online methods that are better suited for environments, where agents must take actions within a very small amount of time(e.g., video games). Our portfolio is then comprised of three A*-based algorithms: *Windowed Hierarchical Cooperative A** (WHCA*) (Silver 2005), *Flow Annotation Replanning* (FAR) (Wang and Botea 2008), and *Bounded Multi-Agent A** (BMAA*) (Sigurdson et al. 2018). We choose these algorithms because they use different strategies to solve MAPF problems and, as a result, can excel on different MAPF problems.

4.1 Windowed Hierarchical Cooperative A*

Silver (2005) proposed a family of A*-based algorithms for solving MAPF problems: In *Cooperative A**, each agent runs an A* search in a three dimensional graph (x-coordinate, y-coordinate, and time) to reach its goal and shares its plan with other agents through reservation tables. Therefore, the agents are able to avoid collisions since each agent knows where all the other agents will be and when they will be there. To improve scalability, *Hierarchical Cooperative A** uses hierarchical search, where each agent uses the length of the shortest path found in an abstracted state space as a guiding heuristic. Finally, to further improve scalability, *Windowed Hierarchical Cooperative A** (WHCA*) limits the search depth of each agent to within a window. Once a partial path within the window is found, the agent

[1] We equivalently optimize the completion rate averaged over all instances in $\mathcal{I}$ instead of the cumulative one in Equation (1) above.

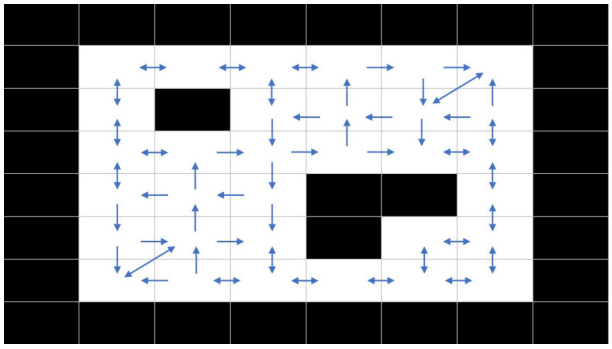

Figure 1: Example of Flow Annotations.

follows it and searches for the next partial path by shifting the window along the current path.

4.2 Flow Annotation Replanning

Like WHCA*, *Flow Annotation Replanning* (FAR) (Wang and Botea 2008) also takes account of other agents plans. However, instead of searching for a new path when the current one is blocked, agents in FAR simply wait at their current nodes until they can reserve their next set of moves in a reservation table. FAR also detects deadlocks, where agents would wait on each other indefinitely, and forces them to move away from their current nodes. They then must replan paths back to the node they were forced off of before resuming their original path to their goal.

To reduce the likelihood of agents blocking each other, FAR annotates at each node with the direction that agents at that node should follow. These annotations combined create "highways," where agents can quickly move from one end of the map to another without ever stopping to wait for other agents. Figure 1 shows an example map annotated by FAR which uses the following strategy: It first creates an edgeless annotated graph G' that has the same set of nodes as the original graph G. Then, edges are added to G' in alternating directions, that is, even-numbered rows are assigned west-bound edges and odd-numbered rows are assigned east-bound edges. Similarly, even-numbered columns are assigned north-bound edges and odd-numbered columns are assigned south-bound edges. Additional edges are added in special cases. For example, nodes on corridors that are only one-node wide retain their bi-directional connectivity. Self-loops are always retained as agents always have the ability to wait, however self-loops are not considered in the search.

4.3 Bounded Multi-Agent A*

*Bounded Multi-Agent A** (BMAA*) (Sigurdson et al. 2018) is based on a *Real-Time Adaptive A** (RTAA*) (Koenig and Likhachev 2006), a well-known *single-agent* real-time heuristic search algorithm. RTAA* runs the following procedures iteratively until the agent reaches its goal: (1) perform a bounded-depth A* search from the agent's current position; (2) update the heuristic values of all nodes in the CLOSED list of that A* search to make them more informed; and (3) move the agent along the partial path returned by that A* search. Sigurdson et al. (2018) extended RTAA* to a *multi-agent* setting, where other agents are treated as (moving) obstacles during the search. Additionally, each agent is able to request other agents that are currently located on its goal cell to vacate. The vacating agent will move to any available neighboring node and resume its regular search procedures from its new location.

5 Related Work on Algorithm Selection

Performance of planning algorithms can vary substantially based on the problem (Kotthoff 2014). In particular, selecting a heuristic search algorithm specific to the problem can lead to a substantial boost in performance (Bulitko 2016a; Bulitko 2016b). The latter work showed that the margin for performance improvement over a fixed algorithm grows with the granularity of the selection: Selecting an algorithm per problem instance has a better performance potential than selecting an algorithm per group of problem instances (e.g., problem instances can be grouped based on maps in video games). Follow-up research exploited the potential for performance improvement through machine learning techniques by mapping a problem instance to the best algorithm for that problem instance (Sigurdson and Bulitko 2017). This work however was limited to the single agent domain and does not provide insight in to how the method would perform in the more realistic multi-agent environment.

Traditionally, machine learning techniques require one to intelligently define input features in order to be successful. However, more recently, deep learning techniques have been able to automatically extract features which allows one to provide low-level problem descriptions. Remarkably, even mapping ASCII codes of a textual problem description in SAT and CSP problems to grey-scale pixel values in a square image provided sufficient for deep convolutional neural networks (Loreggia et al. 2016). In pathfinding problems, the encoding can be even simpler since the map itself is naturally represented by a two-dimensional image (Sigurdson and Bulitko 2017). In this paper, we continue the recent line of work and adapt the latter approach from single-agent to multi-agent pathfinding.

6 Our Approach

Before solving our algorithm selection problem defined in Section 2, there are two key design decisions that must be made: (1) What algorithms to include in the portfolio $\mathcal{P}$ of algorithms? (2) How should problem instances in $\mathcal{I}$ be represented as an input to the selection algorithm?

Portfolio Algorithms: Ideally, the selection of algorithms in the portfolio should be sufficiently diverse so that for each possible problem instance, there exists at least one algorithm in the portfolio that does well on that problem instance. Larger portfolios, however, may slow down the learning process as well as result in lower performance due to errors when selecting algorithms. Therefore, in this paper, we consider a relatively small but diverse set of algorithms: Windowed Hierarchical Cooperative A* (WHCA*), Flow Annotation Replanning (FAR), and Bounded Multi-Agent A*