Synthesizing Priority Planning Formulae for Multi-Agent Pathfinding

Shuwei Wang, Vadim Bulitko, Taoan Huang, Sven Koenig, Roni Stern
Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada
{shuwei4, bulitko}@ualberta.ca
Ben Gurion University of the Negev
sternron@post.bgu.ac.il
University of Southern California, Los Angeles, USA
{taoanhua, skoenig}@usc.edu

Abstract

Prioritized planning is a common approach for multi-agent pathfinding. It prioritizes the agents according to a given priority function and then plans paths for them in the order of their priorities such that the path of each agent does not collide with the paths of any agents with higher priorities. The effectiveness of prioritized planning depends critically on choosing good priority functions. Research from last year has successfully learned good priority functions with support vector machines. In this paper, we explore a different technique for learning priority functions, namely program synthesis in the space of arithmetic formulae. We synthesize priority functions expressed as arithmetic formulae over a set of features via a genetic search in the space specified by a context-free grammar. We regularize the fitness function by formula length to synthesize compact human-readable formulae. This readability may help explain the importance of features. Our experimental results show that our formula-based priority functions often outperform the state-of-the-art support vector machine approach on the standard benchmarks in terms of success rate, run time, and solution quality without requiring more training data.

1 Introduction

Multi-agent pathfinding (MAPF) is the problem of moving a group of agents to a set of goal locations while avoiding collisions. The two common objectives of MAPF are to minimize the makespan (the largest arrival time of any agent at its goal location) and to minimize the sum of costs (the sum of the arrival times of all agents at their goal locations). In this paper, we use the sum of costs as our optimization objective. MAPF is NP-hard to solve optimally (**???**) but has numerous real-world applications, including moving game characters in formation in video games (**?**), transporting goods in automated warehouses (**?**), routing pipes in gas plants (**?**), coordinating self-driving cars in intersections (**?**) and embedding virtual network requests in computer networks (**?**).

Prioritized planning (PP) (**????**) is a popular approach for solving MAPF suboptimally. It prioritizes the agents according to a given priority function and then plans shortest paths for them from their start locations to their goal locations in the order of their priorities such that the path of each agent collides neither with the already planned paths of all agents with higher priorities (moving obstacles) nor with the blocked cells in the environment (static obstacles). A good priority function typically results in a small sum of costs while a bad one could prevent one from finding a set of collision-free paths.

Some researchers have manually designed priority functions (**?????**). Such functions are understandable by humans but since they are designed for specific problem settings, they cannot be applied across a wide spectrum of MAPF instances. Thus research from last year successfully learned priority functions with support vector machines from a set of features of MAPF instances and training data (**?**). Such automatically learned priority functions may be able to achieve better performance than manually designed ones but are typically also less readable to humans and thus less explainable.

In this paper we attempt to combine the human-readability of manually designed priority functions and high performance of machine-learned ones. We adopt the approach by **?** to learn priority functions automatically. They are expressed as arithmetic formulae that map MAPF features to priorities. They are learned via a genetic search in the space of formulae specified by a context-free grammar. We regularize the fitness function by formula length to synthesize compact readable formulae. We then compare the performance of our formula synthesis method with the support vector machine learning method from **?**.

Overall, we make the following **contributions**. First, we describe how we adapt an existing automated program synthesis method to the important problem of creating a priority function for solving MAPF with prioritized planning. Second, we show by experimental results that our formula-based priority functions often outperform the state-of-the-art support vector machine on the standard benchmarks in terms of success rate, run time, and solution quality without requiring more training data. Third, we show how the readability of formulae may help explain the importance of specific MAPF features and how to combine them into good priority functions more easily than the state-of-the-art machine-learning approach. Our work is significant because we demonstrate that by synthesizing priority formulae we gain readability without sacrificing performance, thus the synthesized formulae can be analyzed by humans to give tradition-defying insights into priority function design.

2 Problem Formulation

We define the MAPF problem and the PP approach for solving it. Then we frame finding a high-performance priority function as an optimization problem which is the focus of this work.

The **multi-agent pathfinding (MAPF) problem** p is defined by a tuple $(\mathcal{A}, G)$ where $\mathcal{A} = \{a_1, a_2, \dots, a_n\}$ is a set of agents and $G = (V, E)$ is a non-weighted undirected graph with a set of vertices V and a set of edges E. Each agent a_i has a start location $s_i \in V$ and a goal location $g_i \in V$. Time advances in discrete steps. At each step an agent can either move to an adjacent location or wait at its current location. We consider two types of conflicts: vertex conflicts and edge conflicts. A vertex conflict occurs when two agents attempt to be at the same vertex $v_i \in V$ at the same time step. An edge conflict occurs when two agents attempt to traverse the same edge $(v_i, v_j) \in E$ in the opposite direction at the same time step. A solution to a MAPF problem p is a set of conflict-free paths $\{\rho_1, \rho_2, \dots, \rho_n\}$ to move each agent from its start location to its goal location. The cost of the agent a_i's path in the solution is the number of time steps needed for a_i to complete its path from s_i to g_i and remain motionless at g_i. The performance measure is the *sum of the costs* of all agents in a solution to a MAPF problem p. Solutions with lower sums of costs are preferred.

In PP a *priority function* assigns priority to each agent which is used to order the agents. Then a single-agent pathfinding algorithm, such as A* (**?**), is repeatedly invoked in the order of agent priorities to compute a plan for each agent. When searching for a plan for an agent a_i the algorithm considers and avoids conflicts with the plans of higher priority agents $\{a_1, a_2, \dots, a_{i-1}\}$ and ignores all the lower priority agents $\{a_{i+1}, a_{i+2}, \dots, a_n\}$. To do so the single-agent search algorithm is modified to take into account not only the static structure of the search graph G but also already planned paths of the higher priority agents.

The order of agents in PP can have a substantial effect on the sum of costs of the computed paths as well as whether a valid path is found for each agent at all. An optimal ordering allows PP to succeed in computing solutions of lowest sum of costs. Such an ordering normally depends on the problem instance at hand. Thus the problem we tackle in this paper is to find a priority function that minimizes sums of costs on a set of MAPF problem instances. Such a priority function $f : \mathbb{R}^n \to \mathbb{R}$ maps a vector of agent features and returns the agent priority which is used to order the agents. Formally we attempt to solve the following optimization problem:

$$f_{\min} = \operatorname*{argmin}_{f \in F} \ell(f, P) \tag{1}$$

where F is a space of priority formulae, P is a set of MAPF problems and ℓ is a *loss function*: mean logarithmic sum of costs:

$$\ell(f, P) = \frac{1}{|P|} \sum_{i=1}^{|P|} \ln(\zeta(f, p_i)) \tag{2}$$

Here $\zeta(f, p_i)$ computes the sum of costs of the MAPF problem p_i when PP uses the priority function f. We scale the sum of costs logarithmically to emphasize the impact of the regularizer (Section 4.1).

We prefer solutions to the optimization problem to be (i) generated automatically, (ii) be human-readable and (iii) be portable (i.e., yield low sum of costs solutions on novel MAPF problem instances not seen during its generation).

We adopt the evaluation settings from **?**: deterministic ranking and stochastic ranking with random restarts. We reproduce them below for the reader's convenience.

In the case of **deterministic ranking** we rank the agents by their predicted priority scores: $a_i \prec a_j$ if and only if $\hat{y}(\mathbf{x}_i) > \hat{y}(\mathbf{x}_j)$ where $\hat{y} : \mathbb{R}^n \to \mathbb{R}$ is the priority function that takes as input the feature vector $\mathbf{x} \in \mathbb{R}^n$ of an agent and outputs a priority score z_i and $\mathbf{x}_i$, $\mathbf{x}_j$ are feature vectors of agent a_i, a_j respectively.

In the case of **stochastic ranking** with random restarts we use the computed priority scores to form a probability distribution and generate a total priority ordering from agents with high priority to agents with low priority. Specifically, we normalize the predicted scores $\hat{y}_I$ using the softmax function. Agents with higher normalized scores are more likely to be selected earlier and thus assigned higher priorities.

To test the perfromance of the PP methods, given a map $M \in \mathcal{M}$ and a number of agents n we generate a set of test MAPF instances $\mathcal{I}_{Test}^{(M)}$, one from each scenario (**?**), by using the first n pairs of start and goal indices. We use the following measures to evaluate the algorithms, namely, (i) *success rate* (the percentage of successfully solved problems), (ii) *run time to first solution* (only applicable to stochastic ranking with random restarts, the time it takes to find a solution to a problem), (iii) *normalized sum of costs* and (iv) *average solution rank* (we rank the PP methods in ascending order by their sum of costs starting from index 0, and the index of a PP method on a problem is the method's solution rank on that problem).

3 Related Work

While PP is not guaranteed to be complete (i.e., always finds a solution when one exists) or optimal (i.e., minimizes the sum of costs), it is widely adopted and can produce solutions with near-optimal sum of costs (**?**). Therefore improvements to PP have been proposed over the years. Windowed Hierarchical Cooperative A* (**?**) and Rolling-Horizon Collision Resolution (**?**) use PP iteratively in an online manner, considering in every planning period only conflicts in the next few steps.[*]

? run PP multiple times, each with a different, randomly generated priority over the agents. Priority-Based Search (PBS) applies a conflict-directed approach to search in the space of possible agent priorities, defining a partial order over the agents to resolve conflicts between the agents (**?**).

? proposed to use the distances between an agent's start and goal locations as a priority function. **?** proposed a greedy conflict-oriented prioritization mechanism, where an

[*]RHCR is a general framework that has been used with PP as well as with other MAPF solvers.

Algorithm 1: Parallel multi-trial synthesis; adopted from the work of **?** (**?**).

```
input  : training problem set $P$, formula space $F$,
         regularized loss function $\ell^\lambda$, number of trials $T$,
         per-trial time limit $\kappa$, limit of number of
         consecutive stagnate generations $\mu$, population
         size $n$
output: synthesized formula $f$
1 for $t = 1, \dots, T$ in parallel do
2     $f_t \leftarrow \texttt{trial}(F, \ell, \lambda, P, \kappa, \mu, n)$
3     $\ell_t^\lambda \leftarrow \ell^\lambda(f_t, P)$
4 return $f = \text{argmin}_f \, \ell_t^\lambda$
```

agent is given a priority based on how much reserving its paths will increase the sum of costs of all other agents.

Learning in the context of MAPF has been explored in different ways including learning which MAPF algorithm to use for a given problem (**????**), learning which sets of agents should re-plan (**?**), learning which conflicts should be resolved first (**?**) or which search tree node to expand first (**?**) in tree search algorithms for MAPF (**??**). The closest learning approach to the work in this paper was to learn a support vector machine as the priority function (**?**). In their work, they tested the performance of models trained on MAPF PP instances of 500 agents on problems with number of agents larger than 500 and demonstrated that the models can be applied to problem never seen during synthesis. We show that our trained formulae also have generalizability by testing the performance of a synthesized formula trained on a particular map with a specific number of agents on other maps with other numbers of agents thus demonstrate that the formulae are generalizable because it performs well in problems not seen during synthesis.

Program synthesis in heuristic search has been explored by **????** who ran various algorithms to search a space of arithmetic formulae to represent a heuristic function in single-agent pathfinding. A similar approach by **?** was used to synthesize Artificial Life (A-life) agents represented as arithmetic formulae. Both are cases of program synthesis, a growing research area attempting to synthesize/learn programs (or their parts) from data.

4 Our Approach

We adopt the approach of **?** (**?**) to synthesize priority functions for PP expressed as arithmetic formulae. We define the space of the formulae using a context-free grammar and then search this space via a genetic algorithm with regularized loss ℓ as the fitness function. The resulting synthesized formula computes a priority value for each agent, taking as input a set of the agent's features (defined by **?** (**?**) and explained in Table 1). This priority value is then used by PP to order the agents and solve a given MAPF instance.

4.1 Multiple Synthesis Trials

In line with **?** (**?**) we take advantage of parallel computing hardware. Each candidate formula from the space F is evaluated with the PP algorithm on multiple MAPF problem instances in parallel on utilizing multiple cores on a single cluster node. Each of the genetic algorithm (or trial) can be run in parallel on different cluster nodes. Each trial will synthesize a single priority function. The best of them can be selected from the results of multiple evolution trials.

Algorithm 2: Single synthesis `trial`; adopted from the work of **?** (**?**).

```
input  : $F$, $\ell$, $\lambda$, $P$, $\kappa$, $\mu$, $n$
output: synthesized formula $f$
1  $f_{1,\dots,n} \sim F$
2  $l_{\text{historic best}} \leftarrow \infty$
3  $\nu \leftarrow 0$
4  repeat
5      $f', l' \leftarrow \texttt{best-of}_{\ell^\lambda(f_i, P)}(f_{1,\dots,n})$
6      if $l' < l_{\text{historic best}}$ then
7          $l_{\text{historic best}} \leftarrow l'$
8          $f_{\text{historic best}} \leftarrow f'$
9          $\nu \leftarrow 0$
10     else
11         $\nu \leftarrow \nu + 1$
12     $f_1 \leftarrow f'$
13     $f_{2,\dots,n} \leftarrow \texttt{offspring}(f_1)$
14 until $\kappa$ is reached or $\nu > \mu$
15 return $f_{\text{historic best}}$
```

We run T independent synthesis trials in parallel in line 1 of Algorithm 1. We regularize the loss function defined in Section 2 in the same way as **?** (**?**):

$$\ell^\lambda(f, P) = \ell(f, P) + \lambda |f| \tag{3}$$

to bias the genetic search towards shorter formulae which may be easier to read and less likely to overfit the training data. Here $|f|$ is the number of vertices in an abstract syntax tree that would represent the formula and λ is the regularizer constant.

4.2 A Single Synthesis Trial

Each synthesis trial (i.e., a run of the genetic algorithm) is carried out by Algorithm 2.

Our adaptations to the original algorithm by **?** (**?**) lie with the addition of κ, ν and using a single problem set P.

From now on we use $x_{1,\dots,n}$ as a shorthand for $(x_1, \dots, x_n)$. We start with a population of n formulae randomly drawn from the space F in line 1 of Algorithm 2. In line 2 we initialize a historically best/lowest loss to ∞ so that it gets updated immediately during the synthesis. The main loop ending in line 14 terminates if either the time allowance κ expires or the number of consecutive stagnate generations ν (i.e., generations which did not produce a better solution than the current historic best formula) exceeds μ.

In each generation all formulae in the population are evaluated by computing $\ell^\lambda(f_i, P)$ and the formula f' with the lowest loss is selected in line 5 as the candidate. The historic best formula is updated and the number of stagnate generations ν is reset.

Table 1: The 26 features $\mathbf{\Phi} = \{x_1, \ldots, x_{26}\}$ for the agent a_i as defined by **?** (**?**). The *cardinal conflict* and *multi-value decision diagram* (MDD) come from the work of **?** (**?**).

Feature	**Description**
x_1, x_2, x_3	width of each level (excluding the first and the last levels) of MDD$_i$: their mean, max and min
x_4, x_5, x_6	graph distance between s_i and the start locations of the other agents: their mean, max and min
x_7, x_8, x_9	graph distance between g_i and the goal locations of the other agents: their mean, max and min
$x_{10}, x_{11}, x_{12}, x_{13}$	graph and Manhattan distances between s_i and g_i: graph distance, Manhattan distance, the ratio of the graph distance over the Manhattan distance and the absolute difference between the graph distance and the Manhattan distance
x_{14}	the sum of the widths of all levels of MDD$_i$
x_{15}	the number of locations in MDD$_i$ that are also in the MDD of at least one other agent
x_{16}	the number of unit-width levels of MDD$_i$
x_{17}, x_{18}	the number of vertex conflicts between any shortest path of a_i and any shortest path of one of the other agents: counted by agent pair and counted by raw conflict count
x_{19}	the number of the other agents whose goal locations are in MDD$_i$
x_{20}	the number of the other agents whose start locations are in MDD$_i$
x_{21}	the number of the other agents whose MDDs contain g_i
x_{22}	the number of the other agents whose MDDs contain s_i
x_{23}, x_{24}	the number of edge conflicts between any shortest path of a_i and any shortest path of one of the other agents: counted by agent pair and counted by raw conflict count
x_{25}, x_{26}	the number of cardinal conflicts between any shortest path of a_i and any shortest path of one of the other agents: counted by agent pair and counted by raw conflict count

The champion f' is put in the next generation (line 12) while the rest of the population is formed from the offspring of f' (line 13). Each offspring is f' mutated as follows. A node in f' is selected by random and either deleted or modified while keeping the new formula valid. For instance, if the selected node is a unary operator (i.e., $|\cdot|$), we may either replace it with another unary operator, replace it with a random binary operator with randomly initialized children nodes or pypass it by deleting the current node and connect its child node with its parent node (only delete the node if it is the root node).

5 Empirical Evaluation

We compare performance of our synthesized formulae to the five methods of ordering agents in PP evaluated by **?** on the same 6 maps with the same numbers of agents. The maps have different sizes and structures and they are from the MAPF benchmark suite (**?**). The five priority functions are: (1) **LH**: a query-distance heuristic where agents with longer start-goal graph distances have higher priority (**?**); (2) **SH**: a query-distance heuristic where agents with shorter start-goal graph distance have higher priority (**?**); (3) **RND**: a heuristic that generates a random total priority ordering (**?**); (4) **ML-T**: a machine-learned total priority ordering (**?**); (5) **ML-P**: a machine-learned partial priority ordering (**?**).

We generated training data for ML-T and ML-P in the same way as **?**. That is we ran PP 100 times, once with LH, once with SH, and 98 times with RND to solve each MAPF instance $I \in \mathcal{I}^{(M)}_{\text{Train}}$. We picked the PP run with the least sum of costs for ML-T and the top 5 PP runs with the least sum of costs for ML-P. We used LH and SH to generate agent rankings as the training data for the support vector machines.

We extended the C++ implementation used by **?** with a support for arithmetic formulae used as priority functions within PP. We then used the training data for ML-T and ML-P used by **?** and additionally generated training data for higher number of agents for two large maps. Specifically we generated training data for `lak303d` and `ost003d` with number of agents $n \in \{600, 700, 800, 900\}$ while the previous work only went up to 500 agents.

Unlike the supervised machine-learning methods used before, our synthesis algorithm does not require an *a priori* solved MAPF problems. This is due to the fact that it solves MAPF problems using PP *during* the synthesis to compute the loss of each candidate formula (line 5 in Algorithm 2). While this removes the requirement to build a supervised-style training data the actual synthesis itself is made slower since multiple MAPF problems are solved with PP during synthesis.

In line with **?** we experimented in both the deterministic ranking and stochastic ranking with random restarts settings for the baseline PP algorithms LH, SH, RND, ML-T, ML-P and our formula synthesis method on MAPF test instances on all six maps.

The control variables for the synthesis algorithm are set as follows: the population size $n = 20$, the regularizer constant $\lambda = 0.05$, the per-trial time limit $\kappa = 1$ hour, the limit of consecutive stagnate generations $\mu = 15$.

5.1 Space of Priority Functions

We define the formula space as $F = \{f(\mathbf{x}) = S\}$ where $\mathbf{x} = (x_1, \cdots, x_{26})$ is the feature vector of an agent. The formula body S is generated by the following context-free grammar, adopted and modified from previous work **?**. This context-free grammar was designed in attempt to balance formula expressiveness and formula space size.

$$\begin{aligned} S &\rightarrow T \mid U \mid B \\ T &\rightarrow x_1 \mid x_2 \mid \ldots \mid x_{26} \mid C \\ U &\rightarrow \sqrt{S} \mid |S| \mid -S \mid S^2 \\ B &\rightarrow S + S \mid S - S \mid S \times S \mid \frac{S}{S} \mid \max\{S, S\} \mid \min\{S, S\} \end{aligned}$$

Here x_1 to x_{26} are the agent features (Table 1), $C \in \{1.0, 1.1, 1.2, \ldots, 10.0\}$. Note that any support vector machine learned by ML-T and ML-P approaches (**?**) can be represented in our space of formulae albeit with numeric weights rounded to the values in C. Also note that our formula space includes the two types of query-distance heuristics LH and SH as x_{10} and $-x_{10}$.

5.2 Synthesis of Priority Formulae

In order to generate training MAPF instances that follow a similar distribution as the test MAPF instances, given a scenario with a map $M \in \mathcal{M}$ and a number of agents n, we generated a training instance $I \in \mathcal{I}^{(M)_{\text{Train}}}$ by randomly selecting n start locations from all start locations in the scenario, randomly selecting n goal locations from all goal locations in the scenario, and then randomly combining them into n pairs of start and goal locations. Unlike the way **?** prepared the training data by solving each instance before training, we did not solve each generated training instance because we only needed the features of the n agents of each problem instance. We then normalized the feature values of the agents to make them to be between 0 and 1 across all agents.

For the three small maps `random-32-32-20`, `room-32-32-4` and `maze-32-32-2` (referred to as `random`, `room` and `maze` in the tables) we generated 10 training MAPF instances from each of the 25 scenarios, thus $|\mathcal{I}^{(M)}_{\text{Train}}| = 250$. For the medium sized map `warehouse-10-20-10-2-1` (referred to as `warehouse` in the tables) and the two large maps `lak303d` and `ost003d`, we generated 2 training MAPF instances from each of the 25 scenarios, thus $|\mathcal{I}^{(M)}_{\text{Train}}| = 50$. The number of problems decreases with the map size as problem instances become computationally more expensive to solve.

We ran 32 trials for each number of agents of each map and select the trial with the lowest/best loss as the best trial. The formula produced by each best trial is used to compute agent priority scores for testing. Each trial runs on a separate 16-core cluster node with 64Gb of RAM.

5.3 Results: Deterministic Ranking

In terms of both success rate and average solution rank, the formula synthesis method outperformed the other methods for 3 out of 6 maps and number of agents. The average normalized sum-of-costs measure only considers the problem instances that are solved and ignores the ones that are not solved. For instance, if out of 25 problems a PP method solves only one, its average normalized sum of costs is the sum of costs of that one problem. Our formula synthesis method achieved average normalized sum of costs values similar to others.

We omit results for `random-32-32-20` with number of agents 250 because none of the PP methods solved any test problem instances in the deterministic setting.

5.4 Stochastic Ranking with Random Restarts

We report the success rate, solution ranks and run time to find the first solution during restarting in Table 3. We adopted the value for the hyper parameter from **?** ($\beta = 0.5$) when testing the trained support vector machines and our synthesized formulae. We omit results of small numbers of agents for the six maps because the success rates and average solution ranks are similar across all PP algorithms.

The random restart technique is used to boost the success rate of the two ML-guided methods while preserving their average solution ranks (**?**). While it is clear that the two ML-guided methods have the best performance in all three measures among the PP algorithms, the synthesized formulae are not benefiting as much from the random restart scheme as the two ML-guided methods are and sometimes the formulae perform even worse with random restarts compared to without.

In the stochastic setting with random restarts, ML-P remains the best-performing algorithm in success rate, average run time to the first solution and average solution rank. Although not shown in Table 3, ML-P also performs the best among all methods in terms of the average normalized sum of costs.

5.5 Explainability of Synthesized Formulae

Here we show all synthesized formulae for all maps and all numbers of agents n in Table 4. Consider that a trained support vector machine $f(\mathbf{\Phi}) = \mathbf{w}^\intercal \mathbf{\Phi}$ (**?**) when converted into a formula in space F, contains 26 features, 26 weights multiplied to each feature, 26 multiplication operators and 25 addition operators, thus in total $26+26+26+25 = 103$ nodes. In contrast, the average number of nodes in the synthesized formulae in Table 4 is 6.95. Therefore the synthesized formulae may be more readable than the support vector machines.

Synthesized formulae that outperform other algorithms in terms of the success rate and average solution rank frequently include x_7 or $-x_7$. The feature x_7 is the mean graph distance between the agent i's goal g_i and the goal locations of other agents. This suggests that proximity of the agent's goal to other agents' goals is an important factor in determining the agent's priority. Whether x_7 or $-x_7$ is more effective depends on the problem configuration. Indeed, $-x_7$ appears most often in formulae synthesized for `random-32-32-20` while x_7 appears most often in formulae synthesized for `warehouse-10-20-10-2-1`.

Consider the following formula 4:

$$f_6 = -\left(\frac{x_7}{10 - x_1 + x_{18}^2}\right)^2 = \frac{x_7^2}{-(10 - x_1 + x_{18}^2)^2}$$

Table 2: Success rate, average solution rank and average normalized sum of costs for deterministic ranking. The best results achieved among all algorithms are shown in bold. The results are obtained by training and testing on the same map with the same number of agents n. ∞ indicates that no problem was solved.

Map	**n**	**Success rate (%)**						**Average solution rank**						**Avg. normalized sum of costs**					
		LH	SH	RND	ML-T	ML-P	FML	LH	SH	RND	ML-T	ML-P	FML	LH	SH	RND	ML-T	ML-P	FML
random	50	**96**	12	76	92	52	**96**	2.56	3.76	**1.48**	1.60	1.92	2.00	1.48	1.10	1.29	1.46	**1.09**	1.47
	100	**96**	24	48	40	16	92	1.76	2.12	1.96	1.76	2.64	**1.32**	1.59	1.16	1.43	1.15	**1.15**	1.47
	150	52	4	20	**64**	4	60	1.12	1.88	1.76	1.28	1.92	**0.72**	1.60	**1.19**	1.34	1.67	1.24	1.41
	175	36	0	8	44	40	**48**	1.20	1.76	1.48	1.12	0.96	**0.76**	1.71	∞	**1.40**	1.64	1.66	1.45
	200	12	0	0	32	28	**40**	0.96	1.12	1.12	0.68	0.84	**0.40**	1.70	∞	∞	1.71	1.68	**1.64**
	225	0	0	0	4	4	**8**	0.16	0.16	0.16	0.12	0.12	**0.08**	∞	∞	∞	1.82	**1.39**	1.44
room	50	**88**	16	40	60	16	**88**	1.92	2.28	1.96	**1.16**	2.24	1.32	1.66	1.38	1.44	1.45	**1.35**	1.57
	75	**88**	0	4	24	32	80	1.16	2.28	2.24	1.60	1.48	**0.60**	1.69	∞	2.09	**1.58**	1.60	1.63
	100	52	0	4	56	64	**68**	1.72	2.44	2.36	1.40	0.88	**0.60**	1.78	∞	1.87	1.80	**1.73**	1.75
	125	20	0	0	24	20	**28**	**0.48**	0.92	0.92	0.60	0.64	**0.48**	1.88	∞	∞	1.91	1.88	**1.86**
	150	0	0	0	0	4	**8**	0.12	0.12	0.12	0.12	0.08	**0.04**	∞	∞	∞	∞	2.13	**1.81**
maze	50	**84**	0	16	76	76	**84**	1.60	3.36	2.68	1.36	**1.16**	1.56	3.88	∞	**3.55**	4.00	3.99	4.00
	70	**84**	0	0	80	80	**84**	1.40	3.28	3.28	1.56	**1.16**	1.40	4.11	∞	∞	4.03	**3.94**	3.95
	90	**72**	0	0	56	64	68	1.20	2.60	2.60	1.48	1.20	**1.08**	4.25	∞	∞	4.49	**4.25**	4.30
	110	**48**	0	0	24	**48**	24	**0.64**	1.44	1.44	0.92	0.76	1.00	4.68	∞	∞	5.12	4.56	**4.52**
	130	12	0	0	12	**16**	8	0.36	0.48	0.48	0.32	**0.24**	0.40	5.51	∞	∞	4.86	**4.63**	5.17
warehouse	100	**92**	24	80	88	80	20	2.84	2.84	1.56	2.00	**0.52**	3.08	1.76	**1.01**	1.39	1.62	1.02	1.27
	200	**92**	36	40	52	56	32	2.16	1.36	2.20	1.40	**0.96**	2.40	1.89	**1.03**	1.53	1.09	1.06	1.40
	300	**64**	20	24	16	36	28	1.24	1.16	1.48	1.56	**1.00**	1.36	1.88	**1.05**	1.49	1.07	1.08	1.49
	350	**36**	8	16	20	12	28	0.84	1.00	0.96	0.84	0.92	**0.80**	1.96	**1.07**	1.77	1.29	1.22	1.43
	400	12	4	4	**32**	8	**32**	0.80	0.80	0.88	0.48	0.72	**0.36**	2.00	**1.10**	1.53	1.94	1.18	1.45
	450	16	0	4	12	8	**24**	0.48	0.64	0.56	0.48	0.56	**0.32**	2.04	∞	1.62	1.97	**1.46**	1.47
	500	0	0	**4**	0	0	**4**	0.08	0.08	**0.04**	0.08	0.08	**0.04**	∞	∞	1.87	∞	∞	**1.56**
	550	0	0	0	0	**4**	**4**	0.08	0.08	0.08	0.08	**0.04**	**0.04**	∞	∞	∞	∞	2.06	**1.53**
lak303d	300	**100**	24	**100**	96	80	96	3.68	3.64	**1.16**	2.76	1.20	2.32	2.73	**1.71**	2.15	2.60	1.76	2.58
	400	**100**	40	96	88	76	**100**	3.56	2.72	2.32	1.76	**1.44**	2.72	2.65	**1.71**	2.36	2.07	1.75	2.64
	500	**96**	24	80	84	72	92	3.16	3.12	2.24	1.44	**1.32**	2.44	2.71	**1.74**	2.37	2.03	1.79	2.59
	600	**92**	48	84	**92**	64	88	3.32	1.88	1.92	2.56	**1.52**	2.36	2.71	**1.78**	2.54	2.73	1.82	2.62
	700	**92**	36	76	52	48	88	2.68	2.08	2.04	**1.44**	2.08	2.28	2.86	1.82	2.66	**1.82**	1.87	2.72
	800	80	16	56	76	40	**88**	2.04	2.76	**1.80**	2.16	1.88	1.88	2.99	**1.81**	2.68	2.97	1.94	2.84
	900	60	8	28	56	16	**64**	1.28	1.92	1.44	1.48	1.80	**1.00**	3.09	**1.83**	2.73	3.18	1.97	3.05
ost003d	300	**100**	32	96	**100**	80	84	3.68	3.16	1.80	2.52	**1.56**	1.92	2.70	**1.71**	2.17	2.56	1.78	1.98
	400	**100**	36	96	84	88	92	3.76	3.00	2.56	1.80	**1.20**	2.24	2.74	**1.75**	2.41	1.84	1.76	2.08
	500	**100**	36	80	80	72	**100**	3.20	2.56	1.88	2.04	**1.56**	2.52	2.79	**1.75**	2.18	2.25	1.81	2.60
	600	**96**	40	80	76	68	92	3.08	2.28	1.88	1.88	**1.56**	2.88	2.67	**1.78**	2.32	2.34	1.79	2.83
	700	**100**	36	72	88	60	**100**	2.76	2.44	2.00	2.52	**1.80**	2.36	2.79	**1.82**	2.37	2.90	1.82	2.88
	800	88	24	52	88	40	**96**	2.52	2.56	2.20	2.28	2.24	**1.24**	2.91	1.87	2.30	2.92	**1.84**	2.82
	900	**88**	12	44	20	12	**88**	1.28	2.12	1.44	2.08	2.20	**1.12**	2.95	1.87	2.33	1.86	**1.85**	2.93

which outperforms other PP methods in success rate and average solution rank on random-32-32-20 with number of agents 225 in Table 2.

The numerator x_7^2 gives low priority to the agents that have goals close to other agents' goals. Consider the denominator $-(10 - x_1 + x_{18}^2)^2$. Suppose $x_1 < 10$ then $10 - x_1$ becomes smaller as x_1 grows and because of the negation sign, it makes the denominator larger which leads to smaller priority scores. So as x_1 grows (given $x_1 < 10$) the agent's priority score becomes larger. Suppose $x_1 > 10$ then $10 - x_1 < 0$ but the square makes it positive. Therefore by the same reasoning, as x_1 grows, the agent's priority score becomes larger. When $x_1 = 10$ the denominator becomes $-x_{18}^4$. Changes in x_1 is less significant to the value of the priority score than that of x_{18} because x_{18} appears with a power of 4 in the formula while x_1 does not, so we focus more on x_{18}.

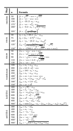
As the component x_{18}^2 grows, the denominator becomes smaller because of the negation sign, and the priority score becomes larger. Thus the larger x_{18}, the larger the priority score and agents that are more likely to have conflicts with other agents will be prioritized.

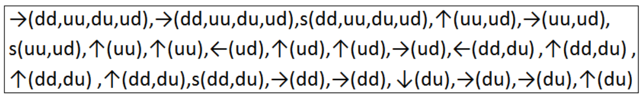
We will now evaluate performance of f_6 on the three small maps and the one medium sized map with deterministic ranking to demonstrate an example of the formula's portability. Instead of computing the agents' priority scores using the formulae synthesized specifically for that number of agents and that map, we use f_6 on all the maps and with all numbers of agents. As per Figure 1 the formula

Table 3: Success rate, average solution rank and average run time to find the first solution for stochastic ranking with random restarts. The best results achieved among all algorithms are shown in bold. The results are obtained by training and testing on the same map with the same number of agents n. ∞ indicates that no problem was solved.

Map	**n**	**Success rate (%)**						**Average solution rank**						**Avg. run time to the first solution**					
		LH	SH	RND	ML-T	ML-P	FML	LH	SH	RND	ML-T	ML-P	FML	LH	SH	RND	ML-T	ML-P	FML
random	175	100	100	100	100	100	100	3.16	**1.88**	2.84	2.40	1.96	2.76	**0.61**	3.64	3.82	1.67	1.41	1.38
	200	88	84	92	**100**	**100**	**100**	2.92	2.36	2.60	2.76	2.24	**2.00**	14.57	21.30	15.79	3.58	**2.91**	3.70
	225	32	16	20	84	**96**	16	2.00	2.20	1.84	1.28	**0.48**	2.08	50.36	52.78	51.32	24.49	**13.39**	52.72
	250	0	4	0	12	**80**	0	0.96	0.88	0.96	0.72	**0.16**	0.96	60.00	58.31	60.00	56.65	**20.77**	60.00
room	75	100	100	100	100	100	100	3.40	2.44	2.36	**2.08**	2.48	2.16	**0.21**	0.29	0.59	0.55	0.38	0.33
	100	80	80	80	96	**100**	92	2.80	2.40	2.92	2.00	**1.80**	2.24	12.56	19.48	20.66	5.00	**1.21**	10.08
	125	24	0	12	76	**92**	12	1.44	2.16	1.92	0.92	**0.56**	1.88	47.74	60.00	55.45	27.70	**8.60**	55.10
	150	0	0	0	8	**76**	**76**	1.60	1.60	1.60	1.48	0.64	**0.32**	60.00	60.00	60.00	57.58	27.41	**16.76**
maze	50	100	100	100	100	100	100	3.52	2.52	**1.52**	2.80	2.20	2.36	0.40	1.63	4.30	1.36	**0.26**	0.74
	70	88	76	72	96	**100**	**100**	3.00	2.40	2.36	**1.96**	2.68	2.04	7.37	20.49	21.75	5.78	**0.45**	7.03
	90	64	20	20	84	**100**	40	1.64	2.64	2.76	1.40	**0.80**	2.24	24.92	49.35	52.59	17.64	**0.57**	39.37
	110	40	0	0	40	**100**	28	1.36	2.08	2.08	1.40	**0.40**	1.36	36.04	60.00	60.00	43.63	**1.86**	50.97
	130	8	0	0	16	**84**	0	1.00	1.08	1.08	0.84	**0.08**	1.08	55.22	60.00	60.00	53.89	**15.81**	60.00
warehouse	350	**96**	**96**	92	**96**	**96**	**96**	3.28	**2.12**	2.36	2.24	2.20	2.20	9.39	14.76	11.23	**8.69**	11.25	14.56
	400	80	**88**	84	80	84	84	3.20	2.04	2.48	2.28	2.04	**1.84**	19.38	**16.41**	23.30	23.04	19.92	22.93
	450	64	64	68	60	**84**	68	2.40	2.08	2.12	2.44	**1.56**	1.96	31.90	35.31	32.71	37.21	**30.30**	31.70
	500	36	20	32	**56**	20	28	1.36	1.60	**0.88**	0.92	1.24	1.08	44.60	55.25	46.89	**42.88**	53.22	49.29
	550	16	12	8	**24**	16	12	0.52	0.60	0.76	**0.48**	0.60	0.64	55.96	57.82	58.56	**52.44**	56.65	56.98
lak303d	500	**100**	**100**	**100**	**100**	96	**100**	4.48	2.08	2.48	**1.72**	2.24	2.00	**5.25**	11.56	22.66	9.41	34.79	8.91
	600	100	100	100	100	100	100	4.64	2.56	1.88	2.40	**1.12**	2.40	**7.49**	40.40	36.11	32.90	99.87	21.10
	700	96	**100**	**100**	96	**100**	**100**	4.32	**1.48**	1.76	2.04	2.40	3.00	41.98	**38.08**	73.10	101.03	72.38	68.01
	800	96	96	92	96	**100**	92	4.12	2.68	2.08	2.24	**1.56**	2.04	**60.65**	148.05	158.82	117.32	126.01	124.54
	900	**92**	**92**	84	84	80	76	2.88	**1.80**	2.00	2.44	1.88	2.64	**130.59**	232.18	249.02	260.38	276.37	255.47
ost003d	500	**100**	96	**100**	96	92	**100**	4.44	2.52	1.84	2.40	**1.16**	2.48	**2.40**	39.21	20.80	31.62	60.03	10.79
	600	**100**	**100**	96	**100**	96	**100**	4.08	2.44	2.28	1.80	**1.36**	3.00	**4.46**	16.58	37.82	18.79	62.19	13.55
	700	**100**	96	96	**100**	96	96	4.32	1.96	2.40	2.00	**1.40**	2.64	**13.79**	47.78	46.36	44.20	83.66	47.17
	800	**100**	96	96	96	96	96	3.56	1.92	2.60	2.72	**1.44**	2.36	**32.53**	95.01	82.01	80.12	110.79	72.39
	900	**100**	92	96	96	92	92	3.68	2.32	2.16	2.08	**2.04**	2.32	**72.49**	164.68	153.91	160.27	172.58	169.36

Table 4: Formulae synthesized for each number of agents of each map. All formulae are manually simplified and the numeric constants are rounded. When PP with a formula outperforms all other PP methods in both success rate and average solution rank for a given map and a number of agents we mark the line with an asterisk.





→(dd,uu,du,ud),→(dd,uu,du,ud),s(dd,uu,du,ud),↑(uu,ud),→(uu,ud),
s(uu,ud),↑(uu),↑(uu),←(ud),↑(ud),↑(ud),→(ud),←(dd,du) ,↑(dd,du) ,
↑(dd,du) ,↑(dd,du),s(dd,du),→(dd),→(dd), ↓(du),→(du),→(du),↑(du)

Figure 1: Success rate and average solution rank of f_6 on the three small maps and the medium sized map compared to existing PP algorithms.

f_6 has the highest success rate and the best average solution rank for most numbers of agents on `random-32-32-20` and `room-32-32-4`. It performs worse on `maze-32-32-2` and `warehouse-10-20-10-2-1`. This shows that the synthesized formulae can outperform existing PP algorithms even on maps not seen during synthesis.

6 Future Work

While performing well, our priority functions were synthesized for the map and the number of agents. Future work will synthesize priority functions with training data from different maps and/or numbers of agents. The effectiveness of our approach may also be increased by considering additional building blocks for the arithmetic formulae. In particular, these building blocks can include previously synthesized formulae (**?**), leading to iterative expansion of the space of formulae. Finally, future work may aim to scale our approach to larger training sets by using other synthesis methods (**????**).

7 Conclusions

We adopted the approach by **?** to learn priority functions to solve multi-agent pathfinding problems with prioritized planning. The priority functions are expressed as arithmetic formulae and synthesized via a genetic search. They are short and human-readable and often outperform the state-of-the-art machine-learning approach in terms of success rate, run time and solution quality without requiring more training data.

We also showed that our synthesized formula can outperform existing PP algorithms in both success rate and average solution rank even on maps and problems not seen during synthesis and can provide insight in its functions.

Acknowledgments

We appreciate support from Compute Canada and financial support from NSERC.

Commodi ipsa harum dolores non ex, accusantium itaque dolore inventore corrupti dolor est quisquam voluptas mollitia enim fugit, deserunt aspernatur temporibus quidem debitis tenetur dolore tempora recusandae, sit earum consectetur, libero velit aspernatur deserunt?Sequi architecto officia minima rem aut impedit mollitia nemo, quaerat tenetur iure necessitatibus, illum deleniti omnis alias tempore quos quibusdam autem labore dolor illo rerum, molestias aperiam ipsum molestiae eum eveniet impedit distinctio earum perferendis, quas esse molestias accusamus deserunt eligendi id mollitia?Dicta temporibus pariatur iure, repellat aspernatur voluptas explicabo consectetur soluta magnam ipsum voluptates.Id dignissimos porro architecto expedita ea quasi labore officiis nulla iste qui, maiores recusandae laboriosam odio voluptatum debitis quibusdam rem doloribus esse molestiae, facilis praesentium quae reprehenderit fuga consequatur minima, quo odit aliquid totam ut eum vero quae commodi earum, consequatur dolorem architecto nisi minus natus.In rerum aliquid dolor, corrupti quasi at assumenda.Repellendus animi nostrum facilis, voluptatem quis molestias vitae fugiat in quos aliquid earum beatae, corrupti sunt culpa excepturi tempora nobis placeat tempore, optio ducimus vel sunt perferendis corrupti animi voluptatibus neque dolor eligendi?Commodi minima eos explicabo adipisci modi praesentium dolorem, nobis quae maiores pariatur eos error exercitationem repellendus eaque.Necessitatibus numquam eum recusandae rem voluptatem hic aperiam qui iste minima, nemo neque voluptatum perspiciatis dolorum laudantium saepe adipisci distinctio odio, natus veniam earum non unde quod sit facilis possimus assumenda, consequuntur illo nisi provident quis nemo?Tenetur praesentium numquam reprehenderit nihil, tempore possimus quas quo repudiandae vitae maiores consectetur enim.Debitis tempore quos ipsa illum consectetur quidem impedit vitae modi ea, ad odio nam laboriosam, magni magnam temporibus repellendus tempora nemo mollitia expedita, eligendi quaerat rerum odio tenetur eos autem ad eveniet.Quidem facere est, fugiat eligendi ea excepturi cum perspiciatis delectus voluptas et laboriosam corrupti corporis, architecto tempora porro expedita corrupti dolor, at unde vel vero ad corporis?Debitis dignissimos ut laudantium non laboriosam dolor, voluptas fugiat explicabo aut repudiandae amet veniam nihil ea unde, rerum voluptates tempora, quisquam ad dolor explicabo vero quam exercitationem, perferendis soluta dolore ut amet iure provident explicabo enim quibusdam?Hic illo quia eos quisquam quasi eum veritatis consequatur perspiciatis officiis, esse aut itaque quod eaque id praesentium quas quaerat distinctio laborum?Natus non unde quod facere est accusantium aperiam repudiandae, dolores minus harum placeat quaerat, adipisci totam delectus voluptate assumenda beatae praesentium saepe ducimus facere atque ad.Quam corporis nulla porro quaerat nobis, hic ratione illum dicta unde repudiandae, recusandae aliquam sequi quis, ipsam totam corporis provident perspiciatis labore in temporibus, autem molestiae ab nobis repudiandae sunt rem ipsam?Dicta mollitia exercitationem veritatis itaque ex voluptas, omnis possimus vero temporibus quas laudantium amet totam provident minus explicabo, eos dolores molestiae eligendi maxime necessitatibus odit, facere porro sequi quisquam illo error ut libero magnam delectus obcaecati, labore vero cupiditate.Minus vel perferendis totam doloribus numquam consequuntur itaque illo, quos perspiciatis ullam quam maiores temporibus quod?Rem quas magni cumque quidem, debitis magnam ex unde atque labore iusto minus voluptates illum ab, ut deserunt laboriosam repellat quasi facere molestias numquam natus, consequatur ducimus animi.