Neural Tree Expansion for Prioritized Multi-Agent Path Deconfliction

Khushal Brahmbhatt
Oregon State University
Corvallis, OR
brahmbhk@oregonstate.edu

Andrew Festa
Oregon State University
Corvallis, OR
festaa@oregonstate.edu

Kavinda Senewiratne
Oregon State University
Corvallis, OR
senewiry@oregonstate.edu

ABSTRACT

Coordinating search efforts in high-risk and time sensitive environments has the potential for applications ranging from rescuing lives to exploring previously inaccessible locations. This problem is made extremely difficult due to the inherent communication restrictions, real-time requirements, and potential for collisions of any such solution, along with the fact that not all tasks may be equal in priority. Previous work has focused on solving path deconfliction in an efficient manner by using a decentralized planner that allows each agent to individually learn how to optimize a global value. However, these approaches do not tend to allow for prioritizing the task of particular agents. Even for approaches that allow for prioritization between different agents, these prioritization schemes generally rely on some physical property of the agent, such as speed or size, as a means for the agent to infer its priority. In this paper, we propose a solution to allow a system designer to explicitly assign a total ordering among the agents in the system in order to allow for valuing completion of more time-critical tasks before lower critical tasks. This is done by having a priority assigned to a task, allowing for a variable number of agents (with differing priorities) in an agent's field of view, and then teaching each agent when to yield right of way but only in the case of a potential conflict. This allows for a system designer or dispatcher to prioritize solving important tasks when presented with a large number of tasks that must be completed in parallel. The results show that agents trained using the prioritized reward function were able to have a similar average system throughput while having a much lower weighted system throughput as based on the priority of the agents.

KEYWORDS

multi-agent, decentralized, Monte-Carlo tree search, priority path planning

1 INTRODUCTION

Due to rising global temperatures, there has been a significant increase in the number of wildland fires in the USA over the past several decades alone. In a span of 30 years, the number of annual wildfires has doubled from approximately 500 to 1000 [2]. In terms of area, there were about 1.3 million acres burned by wildland fires in 1983, and this number grew to approximately 10 million acres by 2020 [3]. In 2018, the estimated insured losses totaled to about $10Bn in the USA, and the Federal Government spent $3Bn for wildfire suppression[9, 18]. In 2021, the Bootleg Fire which started in Southern Oregon, burned almost 500,000 acres of land [1]. This is almost half the number of acres totally burned in 1983 across the entire USA. More importantly, there were countless lives (civilian and firefighters) lost [4]. If we can better fight these fires and predict how they are going to evolve over time, these major losses may be able to be mitigated[20].

The challenge here comes in several parts. First, wildfires are highly unpredictable and the boundaries often shift based on changing weather conditions[10]. Second, due to the smoke and variable terrain, reliable or long range communication is not a guarantee[13]. This all amounts to both terrestrial and satelite-based solutions not being able to provide the information required to accurately track and fight these fires[28]. Persistent terrestrial solutions would require some form of infrastructure to support their operations which may not exist [5] and the information received from satelites have a low temporal resolution [19].

Instead, robots could map and move towards strategic locations to track the progress of the wildfire fronts. This solves the issue of placing humans in potentially dangerous environments and allows for navigating the uncertain terrain. Drone swarms have been proposed in helping with monitoring and suppressing wildland fires [6]. However, as the size of the swarm increases and the communication remains tenuous, it quickly becomes intractable for the drones to plan non-conflicting paths[17].

This work contributes to the problem of finding near-optimal prioritized non-conflicting paths among multiple agents with limited communication by defining an explicit objective-based prioritization scheme that can be applied to the work outlined in [22]. In this way, different aspect of a meta-problem can be addressed with high importance even when the number of agents and objectives in the system becomes intractable for directly computing an optimal solution.

We use a Neural Tree Expansion (NTE) algorithm [22] that leverages Monte-Carlo Tree Search and policy and value networks to learn optimal paths for each drone from a starting position to a given goal position on the fire front[1]. To plan for collision-free paths between the drones, we need a way to deconflict colliding paths. To this end, we use a priority-based reward function to determine which drone gets priority along its planned path and which one has to wait or find a different path. The priorities are assigned to the goal positions and change with time to represent the criticality of that part of the fire front. This priority is used in a shared system reward that is passed to each NTE model associated with locally planning each drone's trajectory.

The rest of the paper is organized as follows. Section 2 introduces related works and gives the theoretical background needed to understand the algorithm in this paper. Section 3 gives an explanation

[1]The code to run these experiments can be found at the following link: https://github.com/kavinda14/learning_based_control_project

of the proposed approach, and section 4 gives an overview of the experiments run and how they were evaluated. Finally, section 5 presents the results along with an analysis. The paper then concludes in section 6 with the feasibility of the approach. Section 7 highlights next steps to improve the performance and increase the range of use cases.

2 BACKGROUND

In multi-agent path planning, multiple agents attempt to find the shortest paths to reach their individual goals. However, in order to avoid colliding, each agent may have to follow a sub-optimal path compared to if they were the only agent in the system. The problem thus becomes how to search a set of non-conflicting paths in order to jointly minimize the time each agent takes to reach its goal and the total throughput of all the agents in the system. To add to the challenge of the problem, there often exists a priority among the agents. That is, it would be desired that a certain agent, or subset of agents, reach their targets faster than those with a lower priority. At the extreme ends, solving this optimization problem is often done in one of two ways: centralized and decentralized. In a completely centralized (cooperative) approach, each agent is able to communicate with the other agents in the system[24, 25], while in a completely decentralized (non-cooperative) solution, each agent only performs its own computations or a local search without additional knowledge from other components in the system[11, 24, 25]. There is an inherent trade-off in the two approaches in terms of optimality and resource cost[33], and so most approaches fall somewhere in between [30].

The centralized approach is able to compute the optimal solution that would jointly minimize the multiple objectives[15], but it comes at the cost of intractability in the number of agents in the system [11]. In fact, this problem has been shown to be NP-hard [17] and thus cannot be reliably computed for any scenario with a sizeable number of agents. Alternatively, in a decentralized approach, each agent only searches a local space and has no global information regarding the other agents in the system. The issue comes in that multiple agents may make local decisions which conflict with the local decisions of other nearby agents[21].

2.1 Prior Work

In order to combat the intractability inherent in computing a global, optimal solution, many approaches construct heuristics that seek to efficiently find near-optimal paths. A*, and variations on it, have been shown to yield decent results [25, 29], and conflict-based search (CBS) [24] is widely used and extended in order to cut down on the number of path conflicts that must be searched through. The general idea that CBS introduces is searching at two levels: a global tree with lower resolution and a low level search for each agent. The high-level search is meant to find the points in a path where the agents may collide. The agents themselves are responsible for searching locally in order to resolve those path conflicts. In doing this, the total system is able to search only over the parts of the paths that give rise to the conflicts, and thus it greatly cuts down on the search space. However, this approach falls short when there may be multiple objectives or each agent may have a unique objective [21]. Several attempts have been made toward this particular challenge, [11, 21, 27], but they fail to consider the situation where a priority may exist between the agents in the systems. Instead, they seek to minimize the total time it takes for all of the agents to reach their objectives rather than a weighted average for each individual agent, which is a necessary consideration in many situations, as discussed previously. In the case of UAVs tracking wildfire spread, the agents may not all be able to reliably communicate and different points along the boundary may be more critical due to proximity to urban centers. Thus, a fully centralized approach is not suitable, and a prioritization scheme is necessary in order to ensure that critical parts of the overall problem are addressed first.

Some work has been proposed to address this specific gap. In [15], the path planning problem is formulated as a mixed integer problem defined using temporal logic and uses a search optimization method to find the optimal collision-free paths based on a fairness function. In [32], the authors use a prioritization heuristic based on the number of path choices available to an agent to deconflict paths. However, this relies on knowing the robot's path options upfront and assign the priority based on characteristics of the agent and environment rather than the task of the agent(s).

There has been work that has targeted the assignment of priority based on the environment and the mobility of the agent. In [24], policies are created for the agents in a decentralized manner by considering kinematic, dynamic or environmental constraints. This method however, fails to capture the assignment of priority based on the task, which is important when monitoring a dynamic fire front.

2.2 Centralized vs. Decentralized Path Planning

When discussing multi-agent path planning, there are two extremes to most approaches, as previously noted. In a fully centralized approach, all agents have global knowledge and global communication[16]. They all act and plan in a cooperative environment, which allows them to find the globally optimal non-conflicting paths. This optimality comes at a cost, however, as it quickly becomes intractable to search for the optimal solutions even in a small grid with a limited number of agents and obstacles[30, 31]. Another common approach is to decompose the computation of the conflicting paths into a series of individual searches[25]. This removes the centralized components which gives rise to the exponential branching factor, but may cause the search to find non-optimal paths. At the extreme end of this, no agent has any explicit information about any other agent in the system, allowing for the search paths of each agent to be explored independently[31].

2.3 Conflict-Based Search

This decentralizing of the problem also gives rise to another common framework known as conflict-based search (CBS)[24]. This approach is motivated by the idea that the trouble in finding a non-conflicting path is dominated by the points in the search space where a conflict happens. The algorithm works on a two-level process. The high level consists of searching a Conflict Tree, which contains all the conflicts between individual agents. The low-level is where fast, individual solvers compute a resolution to the conflict for each node in the tree. In this way, the search for non-conflicting paths is dominated only on the parts of the paths where it matters: the places where a conflict may occur. However, in some cases, it still fails to compete with general heuristic based approaches, such as A* [24].

This approach has been extended in various fashions to account for multiple objectives [21] and to group multiple agents together as one unit, termed a meta-agent [23]. [21] attempts to use this same idea of a conflict-based search to bypass the curse of dimensionality that arises for multi-agent systems that try to solve for multiple objectives. [23] is a direct extension of the original work that proposed CBS and attempts to rectify the situation where heuristic based approaches may outperform CBS. These extensions suggest that CBS is helpful as reducing the search space and focusing on aspects of the problem that matter, but it comes at the cost of finding the optimal solution.

2.4 Monte-Carlo Tree Search

As mentioned before, [22] uses a Monte-Carlo Tree Search(MCTS) to plan out the non-conflicting paths between agents. MCTS is an approach that gained a lot of popularity and attention after AlphaZero used it master the games of Chess, Shogi, and Go. It was able to beat the world champion of Go at the time, Lee Sedol [26]. At it's core, MCTS is a search algorithm that iteratively explores the best child nodes from a particular state[8, 12]. It first selects a node to consider for expansion. That node is then used as a starting point in a simulation for a specified number of steps. Once it reaches this root node, it evaluates that final state and backpropagates the result to the starting node, and a value for the starting action from that node is assigned based on the evaluation performed on the root node. An important detail is how the algorithm handles the exploration versus exploitation trade-off. It greedily selects to expand the net-best node, but this selection is regularized by a "ignored" factor. That is, choosing which node to expand it not based solely on the quality of the node, but also how often it has been ignored out of the total number of simulations run. As a node has been ignored more, it is more likely to be selected for expansion.

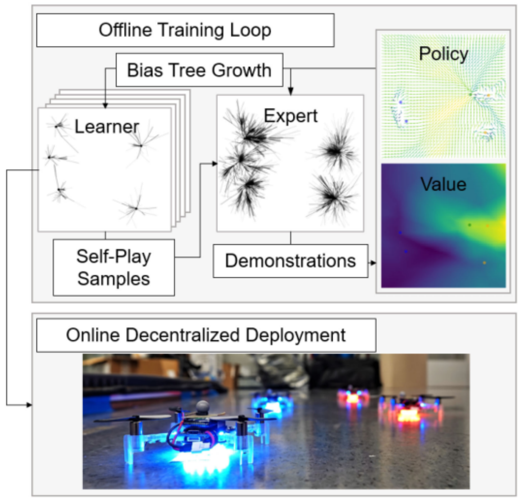
Specifically for multi-agent path planning, MCTS has shown promising results [7, 14, 34]. For cooperative path planning, [34] pairs MCTS with difference evaluations, which allows the agents to learn to coordinate in a centralized manner while maintaining scalability up to 1000 agents in 100x100 grid. [7] uses nested MCTS trees to cooperatively solve for the optimal paths simultaneously between agents. However, in both these works, the agents work in a cooperative (centralized) manner to solve for the non-conflicting paths.

2.5 Neural Tree Expansion

The approach in [22] uses a dual Monte-Carlo Tree Search to both search the global space and the local space of each agent. The global search, depicted as the Expert in figure 1, is only used for training each local agent and is not used when the agents are deployed.

This work extends the idea proposed in CBS of searching along conflicting paths to apply the sequential search to the agents rather than the paths, and it outlines two core loops, as depicted in figure 1: an offline training loop, where each agent is trained to imitate an expert, and an online decentralized deployment loop, where the

Figure 1: Neural Tree Expansion Training Loop [22]



trained agents are deployed to the real world. The training loop contains both the learning agents, all of which only have partial information, and an expert oracle, that has complete and global information. Both the learning agents and the oracle use Monte Carlo Tree Search (MCTS) to explore the possible local paths to a particular depth and is further biased by a value function (during the selection step of MCTS) and a policy function (during the expansion step of MCTS). The learners explore various states (which are all labeled by the expert) and are then used to train the policy and value networks. This loop is repeated until the networks have converged or achieved an acceptably low loss.

Inherently, this creates a partial ordering among the agents as the agents which are searched first will have fewer constraints limiting their potential paths than those agents which search after other agents have already decided on a path. While this makes the solution non-exponential in the search space, the rigidity of the ordering does not allow for maximizing a throughput metric with respect to the relative ordering of the agents. Additionally, the prioritization is not explicitly defined. The proposed research focuses on this aspect of the problem: exploring methods for applying a dynamic priority based on the task, the agent, and the environment.

3 APPROACH

This work is an extension of the solution proposed in [22]. Specifically, it attempts to allow for a system designer to explicitly assign a priority to an agent (or group of agents) based on the importance of their predetermined task. This is in contrast to the work done in [22] in which the priorities of the agents were a total ordering based solely on the order in which they were traversed when resolving conflicts or in [32] where the priorities were based on some characteristic of the environment. The reward function must take into account several potentially competing criteria when encouraging an agent to exhibit a particular behavior in a particular situation.

Variable	Meaning
x_{ai}	x-coordinate of agent i
y_{ai}	y-coordinate of agent i
p_{ai}	priority of agent i
x_{gi}	x-coordinate of the goal of agent i
y_{gi}	y-coordinate of the goal of agent i
$\{d_1 ... d_8\}$	distance to the closest agent in each octant $\{o1 ... o8\}$
$\{p_1 ... p_8\}$	priority of the closest agent in each octant $\{o1 ... o8\}$

Table 1: State Variables

First, the high-level goal of the agents in the system is to reach their assigned goal as quickly as possible without crashing. This is the basis of optimizing non-conflicting paths.

The second is the key contribution of this paper. The agent must yield right of way when encountering an agent of a higher priority.

Finally, the agent is assumed to not have complete knowledge about its environment or the other agents in the system. That is, it does not have any knowledge about the map it is operating in, and more importantly, it does not have a priori knowledge of how many other agents are in the system or their relative priorities. Any given agent is only capable of seeing its immediate surroundings, but it has full communication between any other agents within its range of view.

3.1 State and Actions

A key challenge of representing the state arises from the unknown number of agents in the system. It is desired that the state contains information of the other agents, especially the ones nearby as those are more likely to give rise to conflicting paths in the near future. However, an agent does not know the position and priority of all the other agents in the system. Instead, there must be some mechanism for being able to account for a variable number of agents, and this variability must be conducive to the limited sensing range of each agent. At a high level, the state representation must effectively compress the full information describing the agent's surroundings into a fixed length vector, and this fixed length representation must be able to take into account a variable number of observed agents.

The state representation shown later in equation 1 is based on the competing factors mentioned previously: a desire to reach the goal, not crash, and yield right of way to an agent with a higher priority. The first notion to unpack is that the area around each agent is divided based on a polar representation of the world with the agent positioned at the origin. In this way, an agent can view the world around it based on information in a specific region in a specific direction.

The second important detail is that the agents in the system that are most likely to alter the immediate path of the agent in order to avoid a conflict are those that are closest to the agent. For example, if an agent is moving at 3 meters per second and it sees another agent 8 meters away in a specific direction and another that is 2 meters away (in the same direction), then only the one that is 2 meters away would alter the immediate path of the agent.

$$s_t = \langle x_{ai}, y_{ai}, p_{ai}, x_{gi}, y_{gi}, \{d_1 ... d_8\}, \{p_1 ... p_8\} \rangle \qquad (1)$$

Based on these details, the important information for an agent is captured in a fixed length vector that is composed of two types of information: information about an agent and information about the agent's surroundings. This is shown in equation 1, and table 1 gives a brief description as to the meaning of each variable. For the information about the agent, this is captured by the agent's position, priority, and goal location. For the information about the agent's surroundings, this is the distance to (and priority of) the closest agent in a particular sub-region, where each sub-region is a polar region with the agent at the origin, and the boundaries between each region occurs every $\frac{\pi}{4}$ radians. This divides the area around the agent into 8 sub-regions. Thus, the state space is always 21-dimensional, irrespective of the number of agents in the system.

This idea of polar regions surrounding the agent also motivates the actions allowed for an agent. In this approach, an agent is allowed to stay still or to move in a direction corresponding to any of the sub-regions surrounding it. And at each step, it moves a fixed distance. Thus, the action space is 9-dimensional.

Figure 2 shows an example configuration where the agent senses several agents around it. In this case, an agent sees other agents in octants o_1, o_3, o_5, o_6, o_7, and o_8. However, note that in octants o_3 and o_7 where there is more than 1 agent present, only the closest agents is selected. Thus, for all regions with agents highlighted in green, the corresponding entry in the state vector would be the distance to those agents (in green) and their priorities. For all the other regions, the "distance to the closest" agent would be set to the maximum sensing distance, and the priority of that region would be set to the minimum priority. This can be thought of as a signal to the agent that they have free reign to move around in that region.

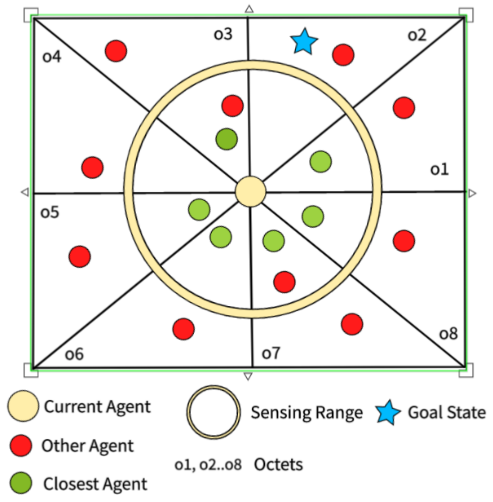


Figure 2: State Representation

3.2 Reward Function

The major contribution of this paper to the NTE approach for planning is the allowance for including an explicit prioritization between agents in a system. This contribution is reached through the formulation of the reward function. As the reward is dependent on priorities, when agents are close to each other, the agent with a higher priority should be encouraged (given right of way) to continue along its optimal path towards its goal, while an agent with lower priority is discouraged from continuing along the path which would conflict with the higher priority agent's path. To reiterate, when defining the reward function, it should satisfy certain criteria.

- Drive the agent towards reaching its goal
- Encourage agents not to collide
- Cause a lower priority agent to yield its path to a nearby agent of a higher priority

$$R_x = \frac{1}{d_g} + \sum_{i=1}^{8} (P_x - P_i)\left(1 - \frac{d_{xi}}{r_x}\right) \tag{2}$$

Variable	Meaning
R_x	reward for agent x
d_g	normalized distance to goal for agent x
P_x	priority of agent x
P_i	priority of closest agent in region i
d_{xi}	distance of agent x to closest agent in region i
r_x	sensing range of agent x

Table 2: Reward Variables

The actual reward function used in this approach is shown in equation 2, and table 2 defines all the variables in the equation. The effect of each parameter can be individually examined to ensure it causes the agent to exhibit the desired behavior. The first term $\frac{1}{d_g}$ increases as the agent gets closer to its goal. Additionally, this growth is non-linear, and so an agent that is closer to its goal is more incentivized to continue on its path. This is a desired behavior based on the idea of freedom of movement. As an agent gets closer to its target, it has fewer options about how it may be able to avoid other agents while moving closer to its goal location.

The second term can effectively be viewed as a regularization term in that it restricts the possible actions an agent may seek to take in order to achieve its goal. At a high level, the agent is encouraged, or discouraged, from entering a particular octant o_i based on the presence or absence of an agent in that octant. The desired behavior of this term is that an octant with a closer agent should contribute more heavily to discouraging the agent from moving in that direction, and an octant with a far (or no) agent should encourage the agent to explore that octant. This is captured by the $\left(1 - \frac{d_{xi}}{r_x}\right)$ term. When there are no agents in any surrounding octant, this term will be 0 for all the octants, and hence only the distance to goal contributes to the reward. Additionally, if the closest agent in that octant is a higher priority than the priority of agent x, then the agent should yield way for that other agent. However, if the priority of agent x is higher, it should have right of way in continuing along its current trajectory. This characteristic is captured by the difference between the two agents' priorities: $(P_x - P_i)$.

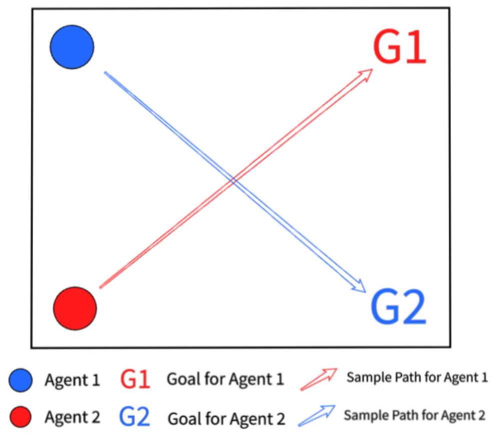


Figure 3: Agent Starting and Goal Locations

4 EXPERIMENTS

The experiments for the approach are dependent on two aspects: the individual performance of an agent and the performance of the system as a whole. That is, any single agent should not take an exorbitant amount of time to reach its goal, but an agent of lower priority should yield right of way to an agent of higher priority in the case of a potential path conflict. Additionally, the cumulative time for each agent to reach its goal should be jointly minimized.

To this end, the experiments were designed in order to compare the learned behavior of the agents. How well do the agents learn to complete the primary objective of reaching the goal? For this, the experimental setup depicted in figure 3 was used. Each agent was placed at a corner, and its goal location was in the opposite corner of the grid. This ensures there is a higher likelihood of the 2 agent paths conflicting. The agent in red has a lower priority of 0.1 and the agent in blue has a higher priority of 0.9. Both agents have a sensing radius of 5m. The environment size is 10x10m.

For all tests, the agents were trained for a maximum of 100 iterations, where a single iteration is the entire training loop depicted in figure 1. More exactly, a training loop consists of creating the expert demonstrations and then using the gathered data to update the global value network and the policy networks of each agent. Each expert generated its examples using a Monte-Carlo Tree Search with a depth of 25, and it generated 20 demonstrations when building both the value and policy datasets. When evaluating the child node of the tree search, the evaluation was based on the best node found in the search rather than subsampling or averaging the child nodes.

As for the neural networks themselves, the policy networks were 2 hidden layer feedforward Gaussian networks with Relu activations and 24 hidden units. The value network was likewise a feedforward network with a Relu activation, but with a deterministic output. Additionally, the value network had only a single hidden layer with 12 units. The input to both networks was the state representation. The output from the value network was 1-dimensional, representing the value for a given state, while the output from the policy network was also 1-dimensional, representing the action to take from a given state. During training, the learning rate was set to 0.001 with a batch size of 1. 80% of the training examples were used for updating the parameters of the model and the remaining 20% were used as the holdout set.

4.1 Goal-Based Reward

The first test is focused on establishing a baseline of performance. How do the agents interact without a prioritization term? This is meant to gauge the ability of the agent to just learn to find a path to a goal and does not consider path deconfliction or prioritization. Thus, when running this experiment, the regularization term was removed from the reward for each agent, following the reward equation shown in equation 3.

$$R_x = \frac{1}{d_g} \tag{3}$$

In formulating the reward in this way, it is an evaluation of the approach in [22] that this work is based on.

4.2 Priority-Based Reward

The second is aimed at showing the correct behavior of the interaction between nearby agents in the system. That is, it evaluates how much an agent's path deviates when encountering a single agent with a higher (and lower) priority. This experiment is conducted with the reward function in (2).

4.3 Evaluation

The performance of the system was based on several criteria, as mentioned above. For the individual performance of an agent, the performance is calculated based on the number of steps it would take the agent to get there in the optimal (straight) path. The system performance is then the mean average of the percent increases (in number of steps) it took for each agent to reach its goal location. These two metrics allow for evaluating both the individual performance of the agents as well as the total system throughput.

For the prioritized reward experiment, the paths obtained were also measured based on the prioritization criteria. That is, an agent with a higher priority should have right of way in any conflicts with an agent of a lower priority. As both agents are equidistant from their goal locations, this can be directly measured by comparing the number of steps each agent took to reach its goal. That is, the agent with a lower priority should take more steps to reach its goal than the agent with a higher priority.

Additionally, the percent increase was also calculated as a weighted average where the weight of each agent is its normalized priority. In this way, it is possible to not only observe the impact the regularization has on the system throughput, but also how much of an impact the prioritization has on skewing the optimal paths found for each agent. If the weighted percent increase is significantly different than the mean average, then this would suggest that the prioritization may have a larger impact on the performance than is desired (so long as the agents yield right of way in the correct scenarios).

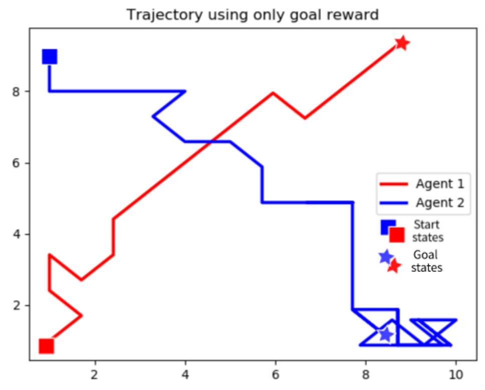


Figure 4: Trajectory Using Goal Rewards

5 RESULTS AND ANALYSIS

In all of the following results and figures, the agent in red has a lower priority of 0.1 and agent 2 in blue has a higher priority of 0.9. The final trajectories of the experimental setup discussed in section 4 using both the goal rewards and the priority reward functions are shown in figure 4 and figure 5 respectively. In the final trajectories, it can be seen that the path of each agent, shown in figure 4 is much more direct when not using the prioritized reward, while in figure 5, one of the agents looks to move around the other agents path as they meet towards the center. At a high level, this is the general behavior that the agents are expected to exhibit.

However, a single snapshot of the final trajectories of each agent only gives a partial view of how each agent acts in this scenario. Thus, figure 6 and figure 7 depict the path the agents took at each timestep. Figure 6 shows the agents not meeting all that closely towards the expected collision point, though the agents are still within the sensing range of each other. Thus, while the expected alteration may be minimal, it can still be seen that the agent paths are not altered at all based on the presence of another agent.

From figure 7, between steps 12 to 15, the lower priority red agent (agent 1) yields to the higher priority blue agent (agent 2). It then continues along its trajectory towards its goal location in steps 12 and 13. This behavior is consistent with the criteria listed in section

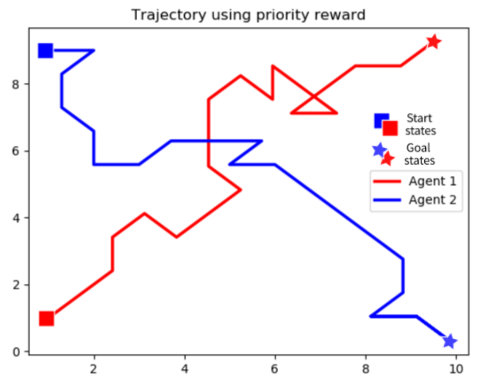


Figure 5: Trajectory Using Prioritized Rewards

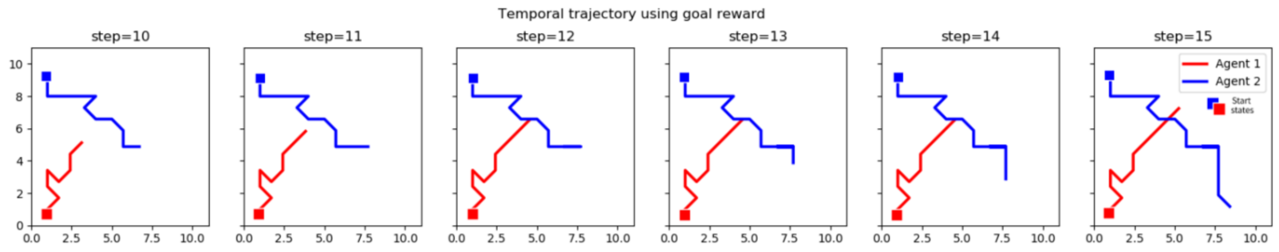


Figure 6: Agent historical trajectories using goal rewards

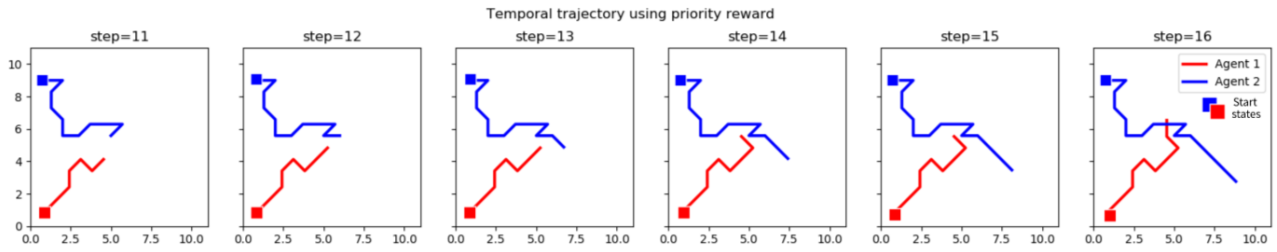


Figure 7: Agent historical trajectories using prioritized rewards

3.2. In figure 6, when using only the distance to goal as the reward, both agents continue towards their goal without adapting their behavior to each other when they come close. However, neither agent was able to learn to the optimal path in this situation which would be to go straight towards the goal location. Table 3 shows the number of steps each agent required to reach the goal location (averaged over 10 trials) after having trained for 100 iterations. In both cases, the agents had a minimal path of 12 steps, as calculated if each agent were to travel directly to its goal location.

Compared to the direct path costs, when using the goal rewards, Agent 1 learned a path that is 166% longer than the optimal path length. Agent 2 learned a path that is 100% longer than the optimal path. Between the two, the average cost increase was 133%. Using just the goal reward, each agent should have learned a policy that brought them directly to the goal location. While this is better than random actions, the behavior of each agent is far below optimal. When using the prioritized rewards, the results are equally less than stellar. Agent 1 learned a path that is 216% longer than optimal, and Agent 2 learned a path that is 58% longer than optimal. When using either reward functions,

The caveat comes when comparing how the prioritization comes into play. While the system throughput is roughly equal for both reward functions, when using the prioritized reward, the agent with a higher priority was able to learn to more more directly to its goal while the agent with the lower priority learned to yield right of way to the high priority agent. Additionally, it did this with only a marginal decrease in overall system throughput (1.7%) compared to using just the goal rewards. In fact, when using the normalized priorities as weights, the weighted average of the total costs is 20.9 for the prioritized reward. For the goal reward, the average cost is the same as the weighted cost since each agent can be thought of as having the same priority.

After training for 100 iterations, none of the agents are able to learn their optimal policy. However, their individual performances after the same number of training iterations was roughly similar, though the prioritized reward was able to achieve a much lower weighted average, suggesting that the reward function was effective in allowing for explicit prioritization amongst agents.

Reward	Agent 1 (priority=0.1)	Agent 2 (priority=0.9)	Average
Goal	32	24	28
Priority	38	19	28.5

Table 3: Steps for each agent to reach its goal

6 CONCLUSION

In this paper, we present a priority-based path deconfliction method using neural tree expansion for multiagent path planning that explicitly incorporates the effect of agents' task priorities in planning a collision-free path. In addition, we design a state representation that is able to capture information about other agents in a partially observable environment when it matters most, while being agnostic to the number of agents in the system.

We demonstrate that our priority-based reward function is able to plan collision-free paths by changing the behavior of the agents in the system according to their priorities and allowing agents with higher task priorities to get to their goal faster. We also show that the weighted-average system throughput using our reward function is more optimal than if using a reward based solely on the agent's distance to its goal location.

Due to the complexity of the neural tree expansion algorithm and the numerous hyperparameters that need to be tuned for both the policy and value networks as well as the Monte-Carlo Tree Search, training the model is very computationally expensive. It requires careful tuning of the hyperparameters and training over a significantly large number of iterations and search depth in order to achieve optimal results.

7 FUTURE WORK

These experiments only evaluated the performance of the approach on a simple 2-agent scenario. In future work, this could be extended to more complex scenarios where agents have variable speeds and sensing radii, common goals, and static obstacles may be present in the environment. In particular, we would like to test the idea that the same reward function could be used for obstacle avoidance by treating obstacles in the environment as agents with a maximum priority.

Additionally, in a real wildland fire, there are many dynamic obstacles as well, which agents will have to consider when planning. These include other manned aerial vehicles, trees, and even fire. Static obstacles therefore do not depict the full story of an actual wildland fire environment.

A key shortcoming of our reward function however is that it does not assign weightage to the goal objective versus the priority regularization. An agent that is very close to the goal may receive an undue higher reward to continue towards the goal even if it leads to a collision. This could be further explored by adding scaling parameters to the objective and regularization terms.

8 TEAM WORK DISTRIBUTION

	Andrew (%)	Kavinda (%)	Khushal (%)
Organization	60	20	20
Technical	33	33	33
Coding	50	30	20
Writing	45	30	25

REFERENCES

[1] [n.d.]. Bootleg Fire. https://inciweb.nwcg.gov/incident/7609/. Accessed: 2021-10-25.

[2] [n.d.]. Infographic: Wildfires and Climate Change. https://www.ucsusa.org/resources/infographic-wildfires-and-climate-change. Accessed: 2021-10-25.

[3] [n.d.]. Wildfires and Acres. https://www.nifc.gov/fire-information/statistics/wildfires. Accessed: 2021-10-25.

[4] [n.d.]. Wildfires in the United States 101: Context and Consequences. https://www.rff.org/publications/explainers/wildfires-in-the-united-states-101-context-and-consequences/. Accessed: 2021-10-25.

[5] L. Vergara A. Serrano. 2013. Multisensor Network System for Wildfire Detection Using Infrared Image Processing. *The Scientific World Journal* (2013), 10.

[6] Elena Ausonio, Patrizia Bagnerini, and Marco Ghio. 2021. Drone Swarms in Fire Suppression Activities: A Conceptual Framework. *Drones* 5, 1 (2021), 17.

[7] Bruno Bouzy. 2013. Monte-carlo fork search for cooperative path-finding. In *Workshop on Computer Games*. Springer, 1–15.

[8] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games* 4, 1 (2012), 1–43. https://doi.org/10.1109/TCIAIG.2012.2186810

[9] National Interagency Fire Center. 2021. *Suppression Costs*. https://www.nifc.gov/fire-information/statistics/suppression-costs

[10] Auburn Forestry Department. 2021. *Weather Elements that Affect Fire Behavior*. https://www.auburn.edu/academic/forestry_wildlife/fire/weather_elements.htm

[11] Vishnu R Desaraju and Jonathan P How. 2011. Decentralized path planning for multi-agent teams in complex environments using rapidly-exploring random trees. In *2011 IEEE International Conference on Robotics and Automation*. IEEE, 4956–4961.

[12] Mańdziuk J. 2018. MCTS/UCT in Solving Real-Life Problems. In *Advances in Data Analysis with Computational Intelligence Methods*, Rutkowski L. Yen G. Gawęda A., Kacprzyk J. (Ed.). Vol. 738. Springer, 277–292. https://doi.org/10.1007/978-3-319-67946-4_11

[13] Ronald W. Hodgson Jonathan G Taylor, Shana C Gillette. 2007. Informing the Network: Improving Communication with Interface Communities During Wildland Fire. *Special Section on Fire Human Ecology* 14, 2 (2007), 13.

[14] Mohammad Sina Kiarostami, Mohammad Reza Daneshvaramoli, Saleh Khalaj Monfared, Dara Rahmati, and Saeid Gorgin. 2019. Multi-agent non-overlapping pathfinding with monte-carlo tree search. In *2019 IEEE Conference on Games (CoG)*. IEEE, 1–4.

[15] Connor Kurtz and Houssam Abbas. 2020. FairFly: A Fair Motion Planner for Fleets of Autonomous UAVs in Urban Airspace. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 1–6.

[16] Ryan Luna and Kostas E. Bekris. 2011. Efficient and complete centralized multi-robot path planning. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 3268–3275. https://doi.org/10.1109/IROS.2011.6095085

[17] Bernhard Nebel. 2020. On the Computational Complexity of Multi-Agent Pathfinding on Directed Graphs. In *ICAPS*.

[18] US Department of the Interior: Bureau of Land Management. 2021. *The True Cost of Wildfire in the Western U.S.* https://www.blm.gov/or/districts/roseburg/plans/collab_forestry/files/TrueCostOfWilfire.pdf

[19] Kosmas Dimitropoulos Panagiotis Barmpoutis, Periklis Papaioannou and Nikos Grammalidis. 2020. A Review on Early Forest Fire Detection Systems Using Optical Remote Sensing. *MDPI* 20, Review (2020), 26.

[20] Mark Finney Patricia Andrews and Mark Fischetti. 2007. Predicting Wildfires. (01 2007), 8.

[21] Zhongqiang Ren, Sivakumar Rathinam, and Howie Choset. 2021. Multi-objective Conflict-based Search for Multi-agent Path Finding. *arXiv preprint arXiv:2101.03805* (2021).

[22] Benjamin Riviere, Wolfgang Hoenig, Matthew Anderson, and Soon-Jo Chung. 2021. Neural Tree Expansion for Multi-Robot Planning in Non-Cooperative Environments. *arXiv preprint arXiv:2104.09705* (2021).

[23] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. 2012. Meta-Agent Conflict-Based Search For Optimal Multi-Agent Path Finding. *SoCS* 1 (2012), 39–40.

[24] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219 (2015), 40–66.

[25] David Silver. 2005. Cooperative Pathfinding. *Aiide* 1 (2005), 117–122.

[26] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. 2017. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. arXiv:1712.01815 [cs.AI]

[27] Jur P Van Den Berg and Mark H Overmars. 2005. Prioritized motion planning for multiple robots. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 430–435.

[28] Danielle Venton. 2020. *Help From on High: Satellites Play Key Role in Fighting California Wildfires*. https://www.kqed.org/science/1970910/help-from-on-high-satellites-play-key-role-in-fighting-california-wildfires

[29] Glenn Wagner and Howie Choset. 2015. Subdimensional expansion for multirobot path planning. *Artificial Intelligence* 219 (2015), 1–24. https://doi.org/10.1016/j.artint.2014.11.001

[30] Koping Wang and Adi Botea. 2011. MAPP: a Scalable Multi-Agent Path Planning Algorithm with Tractability and Completeness Guarantees. *J. Artif. Intell. Res.* 42 (2011), 55–90.

[31] Ko-Hsin Wang, Philip Kilby, and Jussi Rintanen. 2009. Bridging the Gap Between Centralised and Decentralised Multi-Agent Pathfinding. (01 2009).

[32] Wenying Wu, Subhrajit Bhattacharya, and Amanda Prorok. 2020. Multi-robot path deconfliction through prioritization by path prospects. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 9809–9815.

[33] Jingjin Yu and Steven M. LaValle. 2012. Multi-agent Path Planning and Network Flow. In *WAFR*.

[34] Nicholas Zerbel and Logan Yliniemi. 2019. Multiagent monte carlo tree search. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. 2309–2311.