Preventing the Spread of Wind-Driven Wildfires with Partially Observable Markov Decision Processes

Kai Fronsdal
Department of Computer Science
Stanford University
kaif@stanford.edu

Jake Hofgard
Department of Mathematics
Stanford University
whofgard@stanford.edu

Aditi Talati
Department of Computer Science
Stanford University
atalati@stanford.edu

***Abstract*—With wildfires becoming more and more common in the Western United States in recent, research into the prevention and tracking of wildfires is increasingly important. Of particular interest are wind-driven wildfires, in which unpredictable wind conditions exacerbate the challenges that fire prevention and suppression agencies already face. We extend previous methods of tracking and fighting wildfires that model fires as partially observable Markov decision processes (POMDPs) to the domain of wind-driven wildfires. By comparing the performance of several state-of-the-art online belief state POMDP solvers, we propose multiple robust, reinforcement learning-based methods for handling the uncertainty associated with wind-driven wildfires. All online methods that we test perform better than the baseline policy of greedily suppressing wildfires in high-cost areas. We also provide a runtime comparison for the approaches that we tested, as efficiency is critical for real-time application of the online algorithms that we adapted to the context of wind-driven wildfires. All source code can be found in our wind-driven wildfires repository.**

***Index Terms*—wildfires, reinforcement learning, partially observable Markov decision processes, Markov decision processes, Monte Carlo tree search, determinized sparse tree search**

I. Introduction

With wildfires becoming more and more common in the Western United States in recent, research into the prevention and tracking of *wind-driven* wildfires is increasingly important. In such events, high winds cause rapidly-spreading, unpredictable fires such as the 2018 Woolsey Fire, which resulted in the evacuation of 295,000 people and destroyed nearly 2,000 structures in southern California [1]. Prevention and mitigation of wind-driven fires relies almost entirely on preventing the fires from starting in the first place through fire bans during extreme wind events and careful power line placement [2]. Once a wind-driven fire begins, it is of utmost importance that extreme wind events can be tracked accurately in real time [3].

We propose a novel methodology for tracking and fighting wind-driven wildfires using techniques from online reinforcement learning. Building on the methodology for modeling wildfires using partially observable Markov decision processes (POMDPs) presented by Diao et al. [4], we aim to provide several robust methods for containing wind-driven wildfires. All of our methods rely on online belief-state planning, with the ultimate goal of providing wildland firefighters with realtime tracking and planning capabilities during wind-driven wildfires. In particular, we compare the performance of several online algorithms for solving POMDPs, including Monte Carlo tree search (MCTS) and determinized sparse tree search. For each method, we gauge performance on wildfires of different sizes, with different wind conditions, and with a variety of different hyperparameters relevant to each algorithm.

II. Related Work

Our research primarily builds on work done by Diao et al. [4]. In their 2020 paper "Uncertainty Aware Wildfire Management," Diao et al. provide a foundation for modeling wildfires as POMDPs and solving the resulting models using online belief-state planning algorithms. We adapted the open-source code developed by Diao et al. to the case of wind-driven wildfires, as their work did not consider the impact that shifting winds might have on wildfire spread during wind-driven events. Our underlying model, presented below, is therefore similar to that of Diao et al. in that we model fire spread on a discretized grid, with each cell having a fixed cost and fuel level. However, our model includes the uncertainty associated with wind, including variable direction and strength, that our belief-state planning must take into account. Diao et al. compares one strategy – Monte Carlo search trees coupled with a particle filter *without* rejection – with standard firefighting strategy of targeting high cost cells, concluding that online algorithms have the potential to significantly outperform typical strategies for wildfires with fixed wind [4]. By extending the work presented by Diao et al. to the complicated realm of wind-driven wildfires, we aim to provide more robust tools for tracking and containing wildfires regardless of wind conditions.

Mern et al. also present a method for containing wildfires using online methods for solving POMDPs in their paper "Improved POMDP Tree Search Planning with Prioritized Action Branching," which utilizes MCTS with double progressive widening (as implemented in the `POMDPs.jl`solver `POMCPOW`) [5]. Again, the work presented by Mern et al.

focuses on wildfires with fixed wind conditions, and the authors further assume a Gaussian distribution to model beliefs, allowing them to update beliefs via a Kalman filter rather than with particle filtering. Additionally, Mern et al. modeled fire containment rather than modeling the spread of a fire through a potentially inhabited area.

Besides the two results mentioned above, the majority of POMDP-related work relevant to wildfire planning and prevention instead considers *tracking* wildfires with unmanned aerial vehicles (UAV). For instance, Julian and Kochenderfer combine a discretized grid model for wildfires (very similar to that of Diao et al.) with a dynamics model for a UAV, incentivizing close tracking of the boundary of a spreading wildfire [6]. Fixed wind conditions were assumed, although the authors did experiment with a set of different fixed wind conditions. Similarly, Shobeiry et al. present a robust method for tracking the spread of wildfires under variable wind conditions with multiple UAVs [7]. Like Mern et al., Shobeiry et al. assume a Gaussian dynamics model for fire spread, allowing for faster computation.

III. Approach

A. Simulation

Given a fixed number of resources (e.g., firefighters, equipment, water pressure, and so on), we provide a method for prioritizing areas for wildfire suppression during an extreme wind event in which wind conditions change rapidly and fire spreads quickly as a result. Each cell in the discretization of the area accessible to the fire is associated with a cost of either -10, -5, or -1, corresponding to developed land, forested land, and open space, respectively. When a grid is initialized in our simulator, approximately 20% of the cells have of -10, 30% of the cells have a cost of -5, and the remainder have a cost of -1, closely matching the proportions of developed land, forested land, and other land in California [4]. This modeling assumption ensures that successful policies will prioritize structure protection and preservation, just as wildland firefighters do [8].

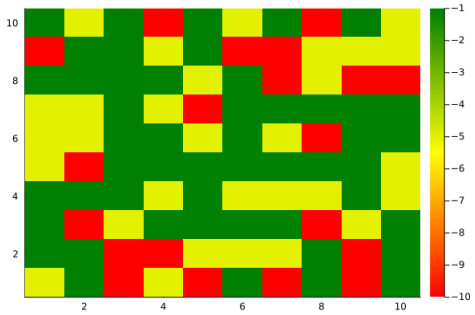

Fig. 1. A randomly-initialized cost map, illustrating the dispersion of low, medium, and high cost cells throughout the discretized world.

In our model, the probability that a wind-driven wildfire spreads from one cell to a neighbor is governed by a generative transition model, expanding on the work of Diao et al. [4], that takes into account variable wind speed and the fuel-level of each cell during an extreme wind event [3]. This problem is best modeled as a POMDP because of the inherent uncertainty involved in monitoring a large geographic area during a wind-driven wildfire; drone and satellite surveillance are often imperfect, so authorities that monitor wildfires cannot always be certain of the perimeter of a fire [1].

B. POMDP Structure

Building upon the work of Diao et al. [4], we constructed a wrapper around `POMDPs.jl` called `FirePOMDP` [9]; open-source code for the `FirePOMDP` class was released by Diao et. al, and we heavily modified the code to accurately represent wildfires with rapidly-shifting wind conditions. An instance of the `FirePOMDP` class is initialized with a grid size n, which corresponds to an $n \times n$ grid, and a probability t that firefighting efforts are successful. Additionally, each instance of the `FirePOMDP` class inherits POMDP structure from `POMDPS.jl`, of the form $(\mathcal{S}, \mathcal{A}, \mathcal{O}, T, R, O, \gamma)$, and our procedure for maintaining a distribution over beliefs is described in Section D below.

In our experiments, we vary $n = 4, 6, 8, 10$, and set $t = 0.8$. We also assume that we can take firefighting actions at $\lfloor n/4 \rfloor$ cells per timestep. Upon initialization, the `FirePOMDP` then randomly initializes an $n \times n$ cost grid, according to the probability specifications mentioned in the previous section.

Each state in the `FirePOMDP` has three attributes: an $n \times n$ BitArray, where 1s indicate cells on fire and 0s indicate cells not on fire, a $n \times n$ "fuel levels" array, which indicates how much fuel is left for the fire to burn through, and the wind, which has a strength and direction. When the POMDP is initialized, the fuel level for every cell is at 5 units, and the wind strength and direction is randomly assigned, where the wind is in one of 8 directions according to the cardinal directions. Finally, in the initial state, a center cell is randomly chosen to be the starting point of the fire, and then cells with a Euclidean distance of at most 3 from the center each independently have a 0.9 probability of also being on fire. Both of these values are constants within the code which can be modified. This is meant to more accurately simulate the way fires begin and spread from a central inciting point, improving upon the previous work which randomly initialized fire throughout the grid [4]. The state space $\mathcal{S}$ consists of possible combinations of burning cells, fuel levels, and wind directions and strengths. Similarly, the observation space $\mathcal{O}$ consists of all possible sets of BitArrays representing fire configurations.

At each state, we receive negative reward corresponding to the items that are on fire. Specifically, we have the reward function

$$r = \sum_{i=1}^{n} \sum_{j=1}^{n} b[i,j] * c[i,j],$$

where $b[i,j]$ is 1 if a cell is burning and 0 otherwise, and $c[i,j]$ is the cost associated with that cell. Thus, we have zero reward only at the terminal state where nothing is on fire, and

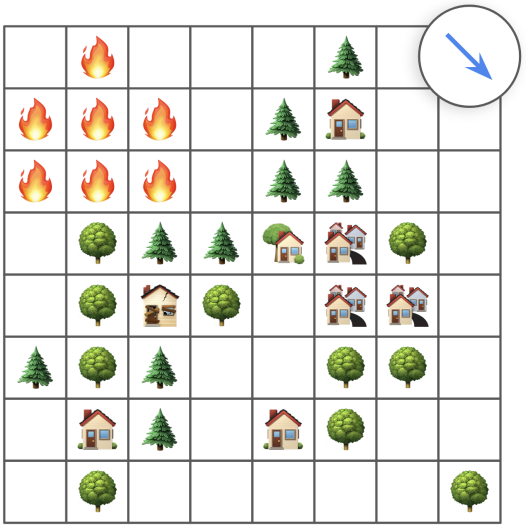

Fig. 2. An 8×8 gridworld representation of a random initial state. Houses represent cells with high costs and tree represent cells with medium costs. Fire represent cells that are on fire. The blue arrow corresponds with the current wind direction.

otherwise we have negative reward, which is made worse by more cells being on fire or developed land being on fire.

At each timestep, the action is a list of $\lfloor n/4 \rfloor$ indices, corresponding to the cells where the agent tried to fight the fire. The action space $\mathcal{A}$ consists of all such actions for all possible fire configurations.

From this construction, our transition model T works as follows:

- First, we decrement the fuel level of every cell which is currently on fire.
- Then, at each cell where the firefighting action is applied, if there is a fire, we successfully put it out with probability t.
- We consider all neighbors of cells which are currently on fire, in all 8 cardinal directions, and spread fire to this neighbor with some probability. Formally, we write that the probability of a fire spreading from i to j, where i is on fire and j is a neighbor of i, is $P_{ij} = 1 - \exp(-swf)$, where s is the strength of the current wind, w is 1 if cell j is in the direction of the wind from cell i, 0.5 if cell j is near the direction of the wind (e.g. cell j is northeast from cell i, but the wind is pointing north), and 0 otherwise. Finally, f is the percentage of fuel left on cell i.
- The wind direction and strength are also modified randomly, to a neighboring direction and strength.
- Finally, any cell which is on fire but has no fuel is no longer on fire, since there is no fuel remaining for the fire to burn through.

We note that though our transition model is supposed to output a probability distribution over all possible new states, we found that outputting a probability distribution over the $2^{n \times n}$ cells each being on fire was too computationally heavy to maintain. Thus, we instead treated the transition model as a generative model that samples implicitly from the distribution of next states. Note that we can still compute the probability of our generated state.

Finally, we model this problem as a POMDP because firefighters often do not have complete observability over the entire terrain, and therefore only know a probability distribution over which states are on fire. Our `FirePOMDP` has a false positive rate of 0.1 and a false negative rate of 0.2, corresponding to the chances of false reports of fire or not being aware of fire in a certain area. The observation model O is then an implicit distribution (similar to the transition function) over the possible `burning` grids, where each cell in the grid is independently flipped with probability 0.1 (if it is currently not on fire) or with probability 0.2 (if it currently is on fire). However, we assume that we do have visibility over the cells we just took an action on, so that for all cells where we did try to fight a fire, the observation reflects that cell's true state. We assume a fixed discount factor of $\gamma = 0.9$.

C. Default Policy

For this problem, we use several different POMDP solvers to try to find an optimal policy. We compare the policy produced by these solvers to a default policy meant to reflect the actions of firefighters today, which first orders the cells from highest to lowest cost, and then takes action on the cells in that order, if it believes they are on fire. Indeed, wildland firefighters typically prioritize putting out fires in developed land or land directly adjacent to structures [8].

D. Online Methods

We compare the performance of three online belief-state planning algorithms to the default policy outlined in the previous section.

1) Monte Carlo Tree Search: Monte Carlo tree search (MCTS) for POMDPs produces trees of depth d formed by sequences of actions and observations [10]. The general MCTS algorithm aims to estimate a value function $Q(h, a)$, where h is a history (i.e., a sequence of actions and observations) and a is an action. From a generative transition model, the algorithm updates counts $N(h, a)$ for each history-action pair as it explores, selecting nodes that maximize the Upper Confidence Bound 1 (UCB1) heuristic, given by

$$\text{UCB1}(h, a) = Q(h, a) + c\sqrt{\frac{\log N(h)}{N(h, a)}},$$

where c is a constant that balances exploration with exploitation (i.e., choosing the action with the highest value $Q(h, a)$ for a given history). Above, $N(h) = \sum_a N(h, a)$.

Although the version of MCTS presented above is effective for POMDPs with small, discrete state and action spaces, it results in very shallow search trees for problems with larger state and action spaces [10]. Because the wind-driven wildfire problem has sizeable state and action spaces (especially for medium and large discretized grids), we rely on a technique called *double progressive widening* to avoid a search tree in

which action nodes are seldom or never revisited. This technique, which was first introduced by Sunberg and Kochenderfer, limits the number of children of a node in a Monte Carlo search tree to $kN(h)^\alpha$, where k and α are hyperparameters and $N(h)$ represents the number of visits to the given node [10]. Additionally, in double progressive widening, we apply the above heuristic to sequences of histories *and* to action nodes in the Monte Carlo search tree. For history sequences, the progressive widening process entails selecting a randomly-generated observation o' (with probability proportional to the number of times $M(o')$ that observation has been simulated) when the number of child nodes exceeds $kN(h)^\alpha$ upon simulating an observation o from a given node [10]. Algorithm 1 below provides pseudocode for the progressive widening step in the action space [10]. Below, $C(h)$ represents the number of children of a node at history h while STEP adds another action to the history h.

Algorithm 1 Action-Space Progressive Widening

Input: History h, parameter k, parameter α
Output: Action a

if $|C(h)| \leq kN(h)^\alpha$ **then**
  $a \leftarrow \text{STEP}(h)$
  $C(h) \leftarrow C(h) \cup \{a\}$
end if
$a_{\text{new}} \leftarrow \text{UCB1}(h, a)$
return a_{new}

The `POMDPs.jl` solver `POMCPOW`, developed by Sunberg and Kochenderfer, applies MCTS with double progressive widening, with a high degree of success in POMDPs with large or continuous state and action spaces [10]. We apply this solver directly to the wind-driven wildfire problem, only slightly adapting the system for updating beliefs that `POMCPOW` uses. In particular, the original solver weights belief updates *and* adds each simulated state into the weighted particle collection that the solver uses to update beliefs via particle filtering [10]. Conversely, we utilize a particle filter without rejection (see Algorithm 2) as presented by Diao et al. [4]; both methods aim to avoid particle deprivation. This technique utilizes our generative transition and observation models to appropriately reweight beliefs according to their likelihood under the observation model. Because of the size of the observation space for the wind-driven wildfire problem, Algorithm 2 reweights all samples uniformly in the event of particle deprivation (i.e., all observations have weight zero) [4]. In practice, Algorithm 2 is implemented with multiple execution threads to improve efficiency. Specifically, due to the independence of each particle, we can update each one on its own thread. We also tested `BasicPOMCP`, a more straightforward implementation of MCTS without double progressive widening [9], [11].

2) Determinized Sparse Tree Search: We also evaluate the performance of determinized sparse partially observable tree search (DESPOT), another online method for solving POMDPs with large state and action spaces proposed by Ye et al. [12]. In DESPOT, m particles are maintained, each representing a scenario of depth d. These particles each follow deterministic sequences of actions and observations, called scenarios, for any sequence of d actions. A determinized sparse tree represents all policies that are possible given a set of K scenarios, and it is a randomly-sampled subtree of the full belief tree of a POMDP (i.e., the tree containing all possible belief trajectories for the POMDP) [12]. This construction significantly reduces the size of the belief search tree corresponding to the relevant POMDP, even though the number of scenarios required to adequately approximate the true belief tree could be as large as the belief space of the POMDP [12].

Algorithm 2 Particle Filter Without Rejection

Input: Belief b, action a, observation o
Output: Updated belief b'

for $i \leftarrow 1, \ldots, |b|$ **do**
  $s_i \leftarrow \text{rand}(b)$
  $s_i' \leftarrow T(s_i, a)$
  $w_i \leftarrow O(o \mid s_i', a)$
end for
if $\sum_{i=1}^{|b|} w_i = 0$ **then**
  $w_i \leftarrow \frac{1}{|b|}$
end if
$b' \leftarrow \emptyset$
for $i \leftarrow 1, \ldots, |b|$ **do**
  $k \leftarrow \text{rand}(1, \ldots, |b|; w_1, \ldots, w_{|b|})$
  $b' \leftarrow \text{push}(b', w_k)$
end for
return b'

In the notation of Ye et al., a scenario for a belief b is a sequence $(s, \phi_1, \phi_2, \ldots)$ where s_0 is sampled according to the belief b and each ϕ_i is sampled uniformly and independently from the uniform distribution on $[0, 1]$. Under the scenario $(s, \phi_1, \phi_2, \ldots)$, an action sequence $(a_1, a_2, \ldots)$ will always produce the same history $(a_1, o_1, a_2, o_2, \ldots)$ from the root of the corresponding belief tree. All nodes and edges of the trajectory $(a_1, o_1, a_2, o_2, \ldots)$ are thus added to the determinized sparse search tree, up to depth d. Each node of the determinized sparse search tree then corresponds to a set Φ_b of encountered scenarios, and upon exploring histories a history $(a_1, o_1, \ldots, a_t, o_t)$ and reaching belief b_t, we iteratively add scenarios of the form $(s_t, \phi_{t+1}, \phi_{t+2}, \ldots)$ to Φ_{b_t} to produce the full determinized sparse tree (in addition to adding the root scenario $(s_0, \phi_1, \phi_2, \ldots)$ to Φ_{b_0}, the set of scenarios of the root of the tree) [12]. This determinizing process reduces the size of the belief tree of depth d from $O(|\mathcal{A}|^d|\mathcal{O}|^d)$ to $O(|\mathcal{A}|^d K)$ nodes, and as shown by Ye et al., it can still produce successful policies for POMDPs with very large discrete state spaces [12]. Thus, the DESPOT approach is well-equipped for handling the wind-driven wildfire problem.

Ye et al. also pair DESPOT with a heuristic search method that helps avoid constructing the full determinized sparse search tree [12]. In particular, motivated by gap heuristic

search, we must provide upper and lower bounds on the value $U^\star(b)$ of the optimal policy for the POMDP. As suggested by Ye et al., in the context of the wind-driven wildfire problem, we provide an upper bound of

$$\overline{U}_0(b) = \frac{R_{\max}}{1-\gamma} = 0,$$

as the maximum reward obtained in the wind-driven wildfire problem is zero if no cells are on fire. Conversely, we provide a lower bound of $\underline{U}_{\pi_0}(b)$, where π_0 is a randomly-generated default policy. This heuristic allows us to prune the determinized sparse search tree by only selecting actions that reduce the gap $\overline{U}_0(b_0) - \underline{U}_{\pi_0}(b_0)$ at the root node b_0, allowing for more efficient computation [12]. Once $\overline{U}_0(b_0) - \underline{U}_{\pi_0}(b_0) \leq \varepsilon$ for some tolerance $\varepsilon > 0$, the DESPOT search can be terminated. As with MCTS, DESPOT is implemented in `POMDPs.jl` via the solver `ARDESPOT` (Anytime Regularized DESPOT), allowing us to immediately apply this method to the wind-driven wildfire problem [9]. We couple `ARDESPOT` with a particle filter without rejection (see Algorithm 2 above) to update beliefs.

IV. Results

With sufficient hyperparameter tuning, MCTS with double progressive widening (as implemented by `POMCPOW`) and particle filtering without rejection outperformed the default strategy of extinguishing the highest cost burning cells for all grid sizes ($n = 4, 6, 8, 10$). MCTS without double progressive widening (as implemented by `BasicPOMCP`) performed better than the default greedy policy for $n = 8$. Additionally, `ARDESPOT` outperformed the default strategy for $n = 4$; computational resource limitations significantly hindered its performance for $n = 6, 8, 10$.

In Table I, we present a comparison of policy found by MCTS with double progressive widening (via `POMCPOW`) and the default greedy policy. For each grid size n, we tuned the UCB1 parameter c as a hyperparameter, selecting the value of c that maximized the advantage of MCTS over the default policy. The average advantage δ of MCTS is given by the average difference between the discounted reward of the policy found by MCTS over 25 trials (reinitializing the POMDP each time) and the discounted reward of the default policy for the same set of 25 POMDP instantiations. The results of tuning

TABLE I
Advantage Over Baseline for POMCPOW

Grid Size (n)	UCB1 Heuristic c	Average Advantage δ
4	10.0	7.90
6	1.0	26.2
8	0.2	52.6
10	2.0	86.5

the UCB1 parameter c are shown in Figures 3 for grid sizes $n = 4, 6, 8, 10$. For all trials, we held the other parameters of the `POMCPOW` algorithm constant, including the maximum depth of the search tree (at $d = 10$), the number of tree queries per step (at $M = 100$), and maximum allowable time for each

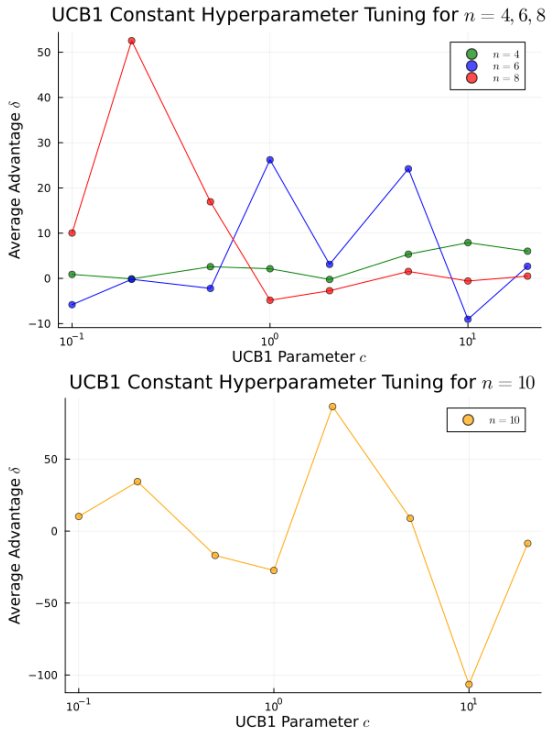

Fig. 3. Average advantage δ for each grid size n as a function of UCB1 parameter c. For larger grid sizes, smaller c performs better while for $n = 4$, larger c tends to perform better.

tree query (at $T = 10$ seconds). In general, exploration is preferable for smaller n (corresponding to a lower value of c) whereas policies that prioritize exploitation (smaller c) exhibits better performance in comparison to the default policy as n increases. However, the optimal value of c depends heavily on n, as exhibited by the above plot for $n = 10$. Figure 4 demonstrates the advantage of the policy found by `POMCPOW` over the default policy for $n = 8$ across all 25 trials at $c = 0.2$. Although `POMCPOW`'s policy performed significantly better than the default policy on average, on specific wildfire instantiations, its performance was indistinguishable from that of the default policy (specifically, when the wildfire starts in a high-cost area of the grid).

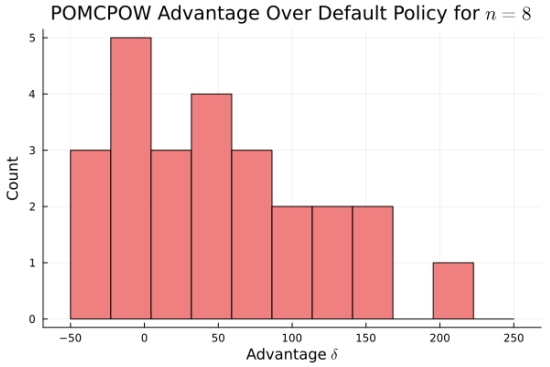

Fig. 4. Advantage δ for $n = 8$ for $c = 0.2$ across 25 POMCPOW trials.

Table II demonstrates the performance of BasicPOMCP and ARDESPOT for smaller grid sizes. For $n = 4, 6$, BasicPOMCP was virtually indistinguishable from the default policy in terms of discounted reward. For $n = 8$, however, it was able to obtain marginally better results than the default policy on average. On the other hand, ARDESPOT outperformed the default policy (and POMCPOW) for $n = 4$, but proved to be computationally intractable on larger grids.

TABLE II
ADVANTAGE OVER BASELINE FOR BASICPOMCP AND ARDESPOT

Algorithm	Grid Size (n)	Average Advantage δ
BasicPOMCP	4	1.51
	6	-3.12
	8	21.10
ARDESPOT	4	8.42

In Table III below, we also present average runtimes for all algorithms (where applicable). Average runtimes were computed by running 10 simulations on a grid of size $n \times n$ for $n = 4, 6, 8, 10$. Given the formulation of the wind-driven wildfire POMDP, however, larger grid sizes were intractable using ARDESPOT, and BasicPOMCP was unable to run experiments for $n = 10$. Although runtimes for POMCPOW occasionally exceeded one minute, it proved to be a much more efficient algorithm than ARDESPOT, with similar runtime efficiency to BasicPOMCP for $n = 4, 6, 8$.

TABLE III
AVERAGE RUNTIMES ACROSS ALGORITHMS AND GRID SIZES

Algorithm	Grid Size (n)	Average Runtime (s)
POMCPOW	4	0.421
	6	2.33
	8	11.9
	10	30.7
BasicPOMCP	4	0.302
	6	2.81
	8	7.62
ARDESPOT	4	16.3

V. DISCUSSION

By modeling wind-driven wildfires as POMDPs and applying well-established online belief-state planning methods such as Monte Carlo tree search and determinized sparse tree search, we were able to significantly outperform the default greedy policy that targets fire in high-cost areas. By incentivizing exploration through the UCB1 heuristic, our usage of the POMCPOW algorithm [10], coupled with a particle filter without rejection, obtained particularly strong results, reducing the cost of wildfires on all grid sizes $n = 4, 6, 8, 10$ relative to the default policy. As the size of the grid increased, the relative advantage of POMCPOW over the default policy also increased. This effect is likely caused by the fact that on larger grids, fire can spread farther as the simulation continues, meaning the a pure exploitation strategy such as the default policy will be inadequate. The default policy performed particularly poorly when the fire *started* in low-cost areas before transitioning to high-cost areas, emphasizing the importance of some degree of exploration.

MCTS via BasicPOMCP obtained a similar, albeit less pronounced, advantage over the default policy for grid sizes $n = 8$ and no apparent advantage over the default policy for smaller n. Although we encountered runtime limitations with determinized sparse tree search as implemented by ARDESPOT, we were still able to obtain promising results for $n = 4$. In terms of runtime, POMCPOW and BasicPOMCP were much more efficient than ARDESPOT. Even though ARDESPOT attempts to produce an approximating subset of the full belief tree, the sheer size of the belief space for the wind-driven wildfire problem as n increases made this approach computationally intractable. This indicates that although determinized sparse tree search may be applicable when the area of interest is small (i.e., a single neighborhood or several acres of forest), MCTS-based methods are more appropriate for tracking and preventing wildfires over large geographic areas.

Finally, we propose several possible improvements of our approach to modeling wind-driven wildfire events. First, while using a particle filter without rejection helped mitigate particle deprivation by some degree, the total number of belief particles is still relatively small because Algorithm 2 limits the total number of belief particles to the number of initial belief particles. Thus, it may be possible to enhance the performance of all three algorithms even further by utilizing particle filtering with adaptive injection, which balances injecting random particles to prevent particle deprivation while simultaneously avoiding the introduction of inaccurate belief particles by tracking the average weight of all belief particles [13]. This method could be implemented and used in conjunction with an algorithm such as POMCPOW or ARDESPOT for belief updates. Conversely, it may be possible to directly use the POMDPs solver Adaptive Online Packing-guided Search (AdaOPS), proposed by Wu et al., which couples an adaptive injection approach to particle filtering with a heuristic for efficiently pruning belief trees [13].

In addition to improving performance via adaptive injection, it may be possible to drastically decrease runtime and therefore test all of the aforementioned algorithms on larger grids by restructuring the transition model to iterate over the discretized grid only once or enhancing the multithreading efforts for belief updating. Following in the footsteps of Mern et al. and Shobeiry et al., assuming Gaussian transition dynamics for wildfire spread could drastically improve computation time, but this approach would need to be adapted to the context of variable wind conditions [6], [7].

CONTRIBUTIONS

Kai constructed and fine-tuned parts the wind-driven wildfire POMDP simulator and implemented multi-threading, helped design the two online methods that we applied to the wind-driven wildfire problem, helped generate figures for our paper, and helped write the introductory, approach, and results sections of our paper.

Aditi wrote the observation and belief update models for the wind-driven wildfire POMDP simulator, implemented and ran experiments using `ARDESPOT` and `BasicPOMCP`, and wrote parts of the introductory, results, and discussion sections of our paper.

Jake wrote the transition model for the POMDP for the wind-driven wildfire POMDP simulator, performed a literature review, implemented and ran experiments using `POMCPOW`, and wrote parts of the introductory, related work, approach, results, and discussion sections of our paper.

REFERENCES

[1] Jon E. Keeley and Alexandra D. Syphard. Twenty-first century california, usa, wildfires: fuel-dominated vs. wind-dominated fires. *Fire Ecology*, 15(1):24, 2019.

[2] Max A. Moritz, Tadashi J. Moody, Meg A. Krawchuk, Mimi Hughes, and Alex Hall. Spatial variation in extreme winds predicts large wildfire locations in chaparral ecosystems. *Geophysical Research Letters*, 37(4), 2010.

[3] Yang Cao and Robert G. Fovell. Downslope windstorms of san diego county. part ii: Physics ensemble analyses and gust forecasting. *Weather and Forecasting*, 33(2):539 – 559, 2018.

[4] Tina Diao, Samriddhi Singla, Ayan Mukhopadhyay, Ahmed Eldawy, Ross D. Shachter, and Mykel J. Kochenderfer. Uncertainty aware wildfire management. *CoRR*, abs/2010.07915, 2020.

[5] John Mern, Anil Yildiz, Larry Bush, Tapan Mukerji, and Mykel J. Kochenderfer. Improved POMDP tree search planning with prioritized action branching. *CoRR*, abs/2010.03599, 2020.

[6] Kyle D. Julian and Mykel J. Kochenderfer. Distributed wildfire surveillance with autonomous aircraft using deep reinforcement learning. *CoRR*, abs/1810.04244, 2018.

[7] Poorya Shobeiry, Ming Xin, Xiaolin Hu, and Haiyang Chao. *UAV Path Planning for Wildfire Tracking Using Partially Observable Markov Decision Process*. Navigation, Estimation, Sensing, and Tracking I, 2021.

[8] Katherine Wollstein, Casey O'Connor, Jacob Gear, and Rod Hoagland. Minimize the bad days: Wildland fire response and suppression success. *Rangelands*, 44(3):187–193, 2022. Changing with the range: Striving for ecosystem resilience in the age of invasive annual grasses.

[9] Maxim Egorov, Zachary N. Sunberg, Edward Balaban, Tim A. Wheeler, Jayesh K. Gupta, and Mykel J. Kochenderfer. POMDPs.jl: A framework for sequential decision making under uncertainty. *Journal of Machine Learning Research*, 18(26):1–5, 2017.

[10] Zachary Sunberg and Mykel J. Kochenderfer. POMCPOW: an online algorithm for pomdps with continuous state, action, and observation spaces. *CoRR*, abs/1709.06196, 2017.

[11] David Silver and Joel Veness. Monte-carlo planning in large pomdps. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010.

[12] Nan Ye, Adhiraj Somani, David Hsu, and Wee Sun Lee. DESPOT: online POMDP planning with regularization. *CoRR*, abs/1609.03250, 2016.

[13] Chenyang Wu, Guoyu Yang, Zongzhang Zhang, Yang Yu, Dong Li, Wulong Liu, and Jianye Hao. Adaptive online packing-guided search for pomdps. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 28419–28430. Curran Associates, Inc., 2021.