A Deterministic Search Approach for Solving Stochastic Drone Search and Rescue Planning Without Communications

Evgeny Mishlyakov[1], Mikhail Gruntov[1], Alexander Shleyfman[2], Erez Karpas[1],

[1]Faculty of Data and Decision Sciences, Technion
[2]Department of Computer Science, Bar-Ilan University
ym@campus.technion.ac.il, gruntovm@campus.technion.ac.il, shleyfa@biu.ac.il, karpase@technion.ac.il

Abstract

In disaster relief efforts, delivering aid to areas with no communication poses a significant challenge. Unmanned aerial vehicles (UAVs) can be utilized to deliver aid kits to survivors in hard-to-reach areas; unfortunately, in some areas, lack of communication and infrastructure presents a key problem. In this paper, we address a stochastic planning problem of planning for a set of UAVs that deliver aid kits to areas that lack communications, where we do not know in advance the locations where aid kits need to be delivered, but rather have probabilistic information about the locations of aid targets. Our main insight is that, despite the stochastic nature of this problem, we can resolve it through deterministic search by monitoring the expected reward for each partial solution. This insight enables the application of deterministic planning techniques, empirically demonstrating a notable improvement in efficiency and response speed. Our approach presents a promising solution to addressing the challenge of delivering aid in regions with limited radio infrastructure, as well as similar planning problems.

Introduction

In disaster relief scenarios, the efficient delivery of aid to areas grappling with severed communication networks stands as a formidable and pressing challenge (Carlson and Murphy 2005). When traditional pathways are blocked and essential supplies remain out of reach, autonomous agents become imperative to ensure the timely assistance and survival of those affected. Unmanned Aerial Vehicles (UAVs) can reach otherwise inaccessible locations to assist in such efforts.

Drones have been extensively used in the aftermath of floods (Boccardo et al. 2015; Jiménez-Jiménez et al. 2020; Andreadakis et al. 2020; Rottondi et al. 2021), landslides (Watson, Kargel, and Tiruwa 2019; Chang et al. 2020), rockfalls (Giordan et al. 2015), forest fires (Tran et al. 2020), hurricanes (Schaefer et al. 2020), tsunami (Marfai, Fatchurohman, and Cahyadi 2019), volcanic eruptions (Tarígan et al. 2017), earthquakes (Lei et al. 2018) and even the Chernobyl radiological disaster (Connor et al. 2020). UAVs provide significant value in disaster management and offer an avenue for delivering aid kits to survivors in remote and hard-to-reach zones (Yakushiji et al. 2020; Aggarwal et al. 2023; Alawad, Halima, and Aziz 2023).

In this paper, we tackle the planning problem inherent in disaster relief with UAVs in communication-deprived environments. A key challenge in these settings is the absence of precise information about target locations for aid kit delivery. Rather than assuming exact knowledge of these targets, we work with partial knowledge about their potential locations. This stochsticity make the problem more challenging than its deterministic counterpart.

The main insight in this paper is that although the planning problem we are trying to solve is stochastic, we can still use deterministic planning techniques, with some modifications, to solve it. This is possible because without communication during plan execution, the drones can not coordinate among themselves, and thus the choice of actions can not be conditioned on any information that is obtained during execution. We explain more about the planning and execution setup in the next section.

However, even understanding that we want non-conditional plans is not enough, as actions have uncertain outcomes, and thus the result of such a plan is not a single state but rather a distribution on possible states. Thus, we also developed a technique for efficiently tracking these distributions, so instead of solving a stochastic planning problem on world states, we can solve a deterministic planning problem on distributions of world states.

As we show later, this transformation does not increase the size of our search space, as each partial plan leads to a unique distribution on states. Furthermore, we show how this insight can be used to (a) obtain provably optimal solutions using deterministic algorithms like Branch and Bound (BnB), and (b) speed up sampling-based search algorithms like Monte Carlo Tree Search (MCTS).

We emphasize that the abovementioned insight is not limited to planning for disaster recovery using UAVs, but can be used to solve similar stochastic planning problems where a non-conditional plan is sought. In a way, this is similar to conformant planning (Cimatti and Roveri 1999). However, our approach also deals with rewards and probabilities, while conformant planning is limited to achieving a goal without specifying exact probabilities on rewards.

Problem Definition

We begin by formally defining the problem and explaining the planning and execution paradigm under consideration. Consider a scenario involving teams of rescue drones dispatched to search for survivors in the aftermath of a catastrophe, such as an earthquake. Operating without communication due to destroyed infrastructure or challenging terrain, each drone is equipped with an aid kit to be dispensed to located survivors. Before the mission, teams receive crucial information about the target area, including the likelihood of encountering survivors at each location and an estimate of potential survivor counts Our objective is to formulate a plan guiding the teams to optimize the delivery of aid kits to survivors, maximizing their impact within a specified timeframe.

Centralized Planning with Decentralized Execution

In the planning and execution paradigm we consider, the planning algorithm runs offline before the drones are launched. After the planning phase ends, each drone gets its own part of the plan to execute. Thus, our planning is centralized. However, once the drones are launched and execution begins, there is no communication – not between the drones, and not between the drones and some ground control station. Specifically, the drones operate in teams, where each team is given a sequence of locations to scan for survivors. As there is no communication between the teams, there is no branching based on whether other teams have discovered targets or not. Furthermore, there is no communication between the drones within the same team. Instead, we assume the drones in each team see each other, can recognize other drones, and scan the location with potential survivors, implementing team behavior using swarming algorithms (Dovrat and Bruckstein 2017). When a team of drones flies over a location, the drone that spots the survivors first will fly down towards them to dispense its aid kit – and remain there (as a power bank, for example). For each of the remaining drones in the team, one of several things may happen: (a) the drone might observe the first-to-land drone on its route to the survivors (as it will be at a lower altitude and heading towards the target), (b) the drone might observe the survivors with the drone that has already landed, or (c) the drone might not detect the survivors at all. This behavior allows each team to send one drone to each group of survivors it discovers. However, because not all drones have seen all targets, there is no common knowledge about this, and our plan cannot even branch on whether survivors were discovered or not.

It is worth noting that our work is part of a larger project, in collaboration with industrial and other academic partners. Other research groups will implement the lower-level layers, including object recognition, localization and navigation, and swarming behavior algorithms. In this paper, we focus on the offline mission planning algorithm only, keeping in mind that the plan will be executed by a set of drone teams in a decentralized manner. Having explained our planning and execution paradigm, we can now formalize our planning problem.

Problem Formalization

To formalize our problem (denoted DRONDEL), we assume the map is given as a weighted graph $G = \langle V, E, \mathsf{cost} \rangle$, and we must plan a path through the graph for each team of drones. Therefore, we treat each team of drones as an agent. Furthermore, we assume that the survivors in each location (node in the graph) cluster together, and thus we need to deliver at most one aid kit to each location.

Before we specify this formally, we define some notation we will use. Let $R_{\max} \geq 0$, and let $\mathcal{P}$ be the space of all discrete probability spaces with maximal reward bounded by $R_{\max}$, i.e.,

$$\mathcal{P} = \{\{(p_j, x_j)\}_{j=0}^{n} \mid n \in \mathbb{N}, \forall 0 \leq j \leq n : 0 \leq x_j \leq R_{\max}, \sum_{j=0}^{n} p_j = 1\}.$$

For each $X \in \mathcal{P}$ it holds that $\mathbb{E}[X] := \sum_{j=0}^{n} p_j x_j \leq R_{\max}$. The probability of drawing zero, $x_0 = 0$, is p_0.

The **input** elements of a DRONDEL problem are:

- a set $A = [k] := \{1, 2, \ldots, k\}$ of k agents that we plan for in a centralized fashion, each agent representing one team of drones;
- a function $u_0 : A \to \mathbb{N}_+$, where $u_0(a)$ is the number of drones, and thus aid kits, in team a at the start. u_0 is referred to as the utilities. We assume $u_0(a) \leq |V|$, since each location requires at most one drop-off;
- a weighted digraph[1] $G = (V, E, \mathsf{cost})$ that represents the map, where V is the node set, E is the edge set, and cost is the non-negative weight function. The nodes represent the possible locations of the agents, while the edges are the connections between such positions;
- a function $l_0 : A \to V$ that associates each agent with its starting point;
- the reward function $\mathcal{R} : V \to \mathcal{P}$ is a given distribution on the number of survivors at each location;
- and $t^{\max} : A \to \mathbb{R}_{0+}$ is the agents' time-limit, i.e., the amount of fuel or energy each agent has (which translates to the amount of time it can spend in the air).
- $\delta_{\mathsf{D}} \in \mathbb{R}_{0+}$ is the time required for an aid kit drop off.

Recall that we perform our planning offline, given only stochastic information about the reward $\mathcal{R}(v)$ at each node v. Once planning concludes, nature draws the realization $r^v \sim \mathcal{R}(v)$. With this in mind, we can define the possible states of the world in our problem.

Problem states To characterize the state of the world during execution, we need to maintain information about both the agents and the rewards, including whether the rewards have been collected. Let s be a function representing a state encompassing both agents and map locations, with the domain $A \cup V$. The state of an agent a is described by a tuple $s(a) = \langle l(a), u(a), t(a) \rangle$, where $l(a) \in V$ is its location on the graph, $u(a) \leq u_0(a)$ denotes its current utility (number of remaining drones), and $t(a) \in [0, t_{\max}(a)]$

[1]This is done for convenience only, all the complexity statements below hold also for an undirected version of the problem.

signifies the personal time of the agent in the air, corresponding to the fuel it consumes. For location v, the state $s(v)$ represents the current reward realization. It is either r^v, randomly drawn according to the distribution $\mathcal{R}(v)$ or 0 if at least one agent with a positive utility budget made a drop-off in v. For simplicity, we use the shorthand notations $s_{\mathsf{loc}}(a) := l(a)$, $s_{\mathsf{utl}}(a) := u(a)$ and $s_{\mathsf{T}}(a) := t(a)$ for $a \in A$, and $s_{\mathsf{rew}}(v) := s(v)$ for $v \in V$. The *initial state* of the problem is: $s^0(a) = \langle l_0(a), u_0(a), 0\rangle$, and $s^0(v) = r^v \sim \mathcal{R}(v)$.

The agents can end their individual plan at any point. Thus, we assume that every map graph G has a terminal vertex. This vertex is reachable at zero cost with zero reward. While it is a convenient shortcut, its inclusion is optional and aims to prevent aimless traversal. For brevity, we exclude this vertex in our theoretical discussion, assuming the search can terminate at any reachable state s.

Actions of the problem When navigating a map, agents have two types of actions:

$\mathsf{move}(a, v, v')$ moves the agent a along a specific edge $(v, v') \in E$, incurring a time cost (fuel budget) of $\mathsf{cost}(v, v')$. It is applicable at state s if $s_{\mathsf{loc}}(a) = v$, $(v, v') \in E$, and $s_{\mathsf{T}}(a) + \mathsf{cost}(v, v') \leq t^{\max}(a)$, i.e., the agent is in v, there is an edge connecting v to v' and the agent has enough time (fuel) to travel to v'. It then results in a state s', where $s'(a) = \langle v', u(a), t(a) + \mathsf{cost}(v, v')\rangle$ for a, and $s'(x) = s(x)$ for any $x \in A \cup V \setminus \{a\}$. The reward of applying this action is 0.

To represent hovering at a given location, we assume that the graph G includes a self-loop for every vertex $v \in V$ with $\mathsf{cost}(v, v) = \delta_{\mathsf{W}}$, where δ_{W} is the minimal step in the discretization of action costs, i.e., each cost can be represented as an integer multiple of δ_{W}. We refer to this as waiting, and denote it by $\mathsf{wait}(a, v)$.

$\mathsf{drop\text{-}off}(a, v)$ agent a scans the location v for survivors to deliver them an aid kit. Recall that the first drone that finds a survivor lands and delivers its aid kit – that drone does not rejoin the team, other drones that find the survivor notice the landed drone and do not land.

The **key assumption** in this problem is that *the agents are not allowed to leave survivors unattended*. Specifically, if an agent decides to perform a drop-off action in a chosen location and realizes that the reward is greater than zero (one of the drones in the team finds a survivor), that drone is obligated to proceed with landing and delivering an aid kit. However, if the reward is zero, signifying the absence of survivors, then no aid kits are delivered.

This action requires a specified time duration denoted as δ_{D}. $\mathsf{drop\text{-}off}(a, v)$ requires only $s_{\mathsf{loc}}(a) = v$ to be applicable, that is, that the agent is at v. However, the result of applying $\mathsf{drop\text{-}off}(a, v)$ in state s depends on whether there are drones left in a ($s_{\mathsf{utl}}(a) > 0$) and survivors in v ($s_{\mathsf{rew}}(v) > 0$). Specifically, if $s_{\mathsf{utl}}(a) > 0$ and $s_{\mathsf{rew}}(v) > 0$ the resulting state s' has $s'(a) := \langle v, u(a) - 1, t(a) + \delta_{\mathsf{D}}\rangle$ and $s'_{\mathsf{rew}}(v) := 0$. In this case, the agent collects utility $s_{\mathsf{rew}}(v)$. Otherwise, $s'(a) := \langle v, u(a), t(a) + \delta_{\mathsf{D}}\rangle$ and $s'_{\mathsf{rew}}(v) := s_{\mathsf{rew}}(v)$, and the reward is 0. Note that the states of all other agents and locations remains unchanged.

Synchronized and individual plans Above we have defined the actions in our problem and their effects. However, because our plan consists of actions for different agents, and different actions have different durations, we must define the *action dynamics* more precisely. Specifically, let s be a state and π be a plan (a sequence of actions). We want to define the state which is reached by applying π at state s.

Assume that π is a sequence of consecutively applicable actions (either move or drop-off) in the DRONDEL instance. The sub-sequence of actions in π associated with an agent a is called an individual plan for that agent and denoted by $\pi(a)$. Note that while the agents do not interact with each other, as we will see later on, the order of the deliveries is important. Thus, we compute the starting time of each action as follows.

Assume $\pi(a) = \langle o_1, o_2, \ldots, o_m\rangle$. We denote the starting time of action o_i by $t(o_i) := \sum_{j=1}^{i-1} \mathsf{cost}(o_j)$. In other words, the sum of costs (durations) of the actions of agent a before o_i, where the starting time of the first action is 0.

We can now define the result of applying a sequence of actions π of all the agents (a joint plan) by computing the starting time for each action in π, and applying the action effects in order of starting time (breaking ties using some predefined rule).

Optimization function The quality measure of the plan π can be seen as the expectation of the sum of rewards collected by all agents. Since we can compute the expectation of all rewards on the map, the reward collected by the agents corresponds 1-1 to the expectation of the sum of rewards that have not been collected. Thus, we can minimize the expected rewards that are left on the map, i.e., we maximize the expected number of survivors that got an aid kit.

Since the rewards for the vertices in $v \in V$ are generated independently, we can say that the initial rewards $s^0_{\mathsf{rew}} : V \to \mathbb{R}_{0+}$ are generated according to distribution $\times_{v \in V} \mathcal{R}(v)$. Note also that each plan π can be executed regardless of the generated rewards. Thus, our goal is to find a plan π from s^0 to s that *maximizes the function*

$$z(\pi) := R_{\text{tot}} - \sum_{v \in V} \mathbb{E}[s_{\mathsf{rew}}(v) \mid \pi],$$

where the expectation of $s_{\mathsf{rew}}(v)$ is computed with respect to π over all possible realisations of $\times_{v \in V} \mathcal{R}(v)$, and $R_{\text{tot}} := \sum_{v \in V} \mathbb{E}[s^0_{\mathsf{rew}}(v) \mid \pi]$. Thus, our goal is to find a plan π from s^0 to s that *maximizes the function* $z(\pi)$.

Modeling as MDP Each DRONDEL instance can be easily represented as a POMDP, where the initial belief state is based on $\mathcal{R}$. Thus, it is a simple exercise to represent it as a finite horizon MDP over belief states. One simplification we can make is to transform our random variables $\mathcal{R}$ to Bernoulli distributed variables. This is possible because under the assumption of no unattended survivors, each distribution $\mathcal{R}(v) = (p^v_j, r^v_j)^n_{j=0}$ impacts agents' utility budgets only when $u_0(v) := s^0_{\mathsf{rew}}(v)$ drawn from $\mathcal{R}(v)$ is strictly greater than zero. Using the linearity of expectation, in terms of the optimization function $\mathbb{E}[\sum_{v \in V} s_{\mathsf{rew}}(v) \mid \pi]$,

we replace the discrete probability $\mathcal{R}(v)$ with the Bernoulli distribution $\mathcal{B}^{p,r}(v) = (0,q),(p,r)$, where $q+p=1$, $q=p_0$ is the probability of drawing zero from $\mathcal{R}(v)$, and $r = \frac{1}{p}\sum_{j=1}^{n} p_j^v r_j^v$. Both distributions have the same expectation. Subsequently, $\mathcal{B}^{p,r}$ denotes the Bernoulli distribution of obtaining r with probability p, where 0 denotes the distribution where the value 0 is achieved with probability 1.

However, our goal is not just to simplify the MDP formulation, but to convert DRONDEL into a deterministic problem DRONDELDP which we can solve using deterministic search tools. Later, we demonstrate that the rewards on the map are just some of the random variables we need to track. The other random variables we need to monitor are the utility budgets of the agents. Given a plan π for a DRONDEL instance, the utility budgets of agents in the states along the execution of π can be seen as random variables dependent on the realization of the rewards. Before this, we show that the problem DRONDEL is not simple even for one agent, and cannot be easily decomposed into a non-synchronized set of individual agent solutions.

Hardness of DRONDEL

We start with formulating DRONDEL as a decision problem:
PROBLEM. Drone Delivery (DRONDEL)
INSTANCE. An instance of DRONDEL and a number K.
QUESTION. Is there a π s.t. $z(\pi) \geq K$?

NP-hardness of the deterministic problem

To obtain the NP-hardness result for DRONDEL we show a reduction to the well-known KNAPSACK decision problem.
PROBLEM. KNAPSACK
INSTANCE. Items $[n]$ with weights $\{w_i\}_{i=1}^n$ and values $\{\nu_i\}_{i=1}^n$, weight limit W, and a number K.
QUESTION. Is there $A \subseteq [n]$ s.t. $\sum_{i\in A} w_i \leq W$ and $\sum_{i\in A} \nu_i \geq K$?

Theorem 1. DRONDEL *decision problem is NP-hard.*

Proof. Given a KNAPSACK problem we construct a deterministic single agent DRONDEL problem. Assume that we have weights $\{w_i\}_{i=1}^n$, values $\{\nu_i\}_{i=1}^n$, weight limit W, and some max value K (all values here are positive). Since we construct a single agent problem, we have $A = \{a\}$. Construct the directed graph[2] $G = (V, E, \mathsf{cost})$ as follows:

- $V := \{v_i \mid i \in [n]\} \cup \{v_0, v_F\}$, where $l_0(a) = v_0$ is the initial location of the agent, v_i locations that correspond to items in the KNAPSACK problem, and v_F is the final point where a should terminate.
- The edges E constitute a star graph that contain edges (v_i, v_0) and (v_0, v_i) for each i, and $\{v_0, v_F\}$.
- The cost function over E is given by $\mathsf{cost}(v_i, v_0) = \mathsf{cost}(v_0, v_i) := w_i/2$ for each i, $\mathsf{cost}(v_0, v_F) := W$, and zero for all other edges (see Figure 1).

We set the time required for drop-off, $\delta_{\mathsf{D}} = 0$. The rewards of the vertices are given by $r^{v_i} = \nu_i$ for each i, $r^{v_0} = 0$, and $r^{v_F} = \nu^{\mathsf{sum}}$, where $\nu^{\mathsf{sum}} := \sum_{i=1}^n \nu_i$. All rewards appear with probability 1. The fuel is set to be $t^{\max}(a) = 2W$ and the utility budget of the agent is given by $u_0(a) = n+1$, since it is the sum of rewards of all vertices. Lastly, the maximisation constant for the decision is $K' := \nu^{\mathsf{sum}} + K$.

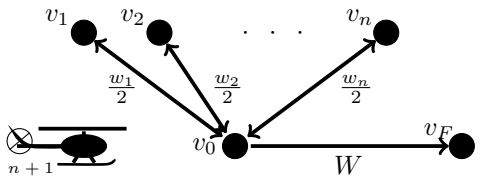

Figure 1: Constructing a graph of the DRONDEL problem based on a KNAPSACK problem. We omitted the terminal vertex v_* since it is reachable from every vertex. $n+1$ is the utility budget of the agent.

It is left to show that any plan π that results in $\sum_{v\in V} s_r^0(v) - \sum_{v\in V} s_r(v) \geq K'$ corresponds to a solution of the KNAPSACK problem where the sum of picked item exceeds or is equal to K. Note that each such π should make a drop-off in v_F, since the initial sum of all initial rewards is $2\nu^{\mathsf{sum}}$. Thus, a terminates at v_F, since v_F is reachable only from v_0, and any optimal solution complies a to visit v_F.

Prior to visiting v_F, the agent a can visit other vertices. There is no point in visiting the same vertex v_i twice, and due to the star shape of the graph, the agent visits vertex v_i via the path $v_0 \to v_i \to v_0$, the cost of this path is w_i. a can not terminate from v_i since it terminates from v_F, i.e., in every optimal solution the agent arrives to v_F.

Thus, to clear the reward ν_i from the map, we need to use w_i fuel. Therefore, the optimal solution for the KNAPSACK problem, can be derived from π. Pick item i for the knapsack, if a makes a drop-off at vertex v_i. □

The importance of patience

While the agents on the map cannot collide and may occupy the same location (vertex), the problem DRONDEL is not trivially decomposable. The agents not only need to decide on the locations of drop-offs but also require synchronization of their actions, because the order in which agents visit locations (and collect the reward there) can make a difference. To illustrate this, we present an instance of the problem where one agent has to deliberately wait for the other to collect the maximum expected reward (minimize the rewards left on the map).

If there are no survivors in location v ($r^v = 0$), the agent refrains from making a drop-off. Consequently, if agent a executes a drop-off action in location v, it implies that either $u(a) = 0$ or $r^v = 0$, or both. Therefore, it is unnecessary for an agent to execute a drop-off in the same location twice. However, since agents are unaware of each other's utility budgets, multiple agents can independently perform a drop-off action at the same location, as the condition $u(a) = 0$ for certain realizations is unknown in advance for a given agent a.

Example 1. *Let $A = \{a_1, a_2\}$ be a two-agent* DRONDEL *problem. In this scenario we refer to a single agent*

[2]While we formulate this result for a directed graph, the same proof with minor changes holds also for an undirected graph.

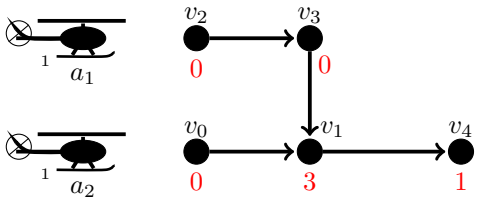

Figure 2: Depiction of graph G in Example 1. The red numbers are the rewards in the realisation where the optimisation results for $\pi^{\text{no-wt}}$ and π^{wt} differ.

rather than a group of drones for the sake of simplicity. The meaning however stays the same: within a single group, i.e. agent, there can be multiple autonomous drones that are referred as $s_{\mathsf{utl}}(a)$ and only one of the group executes drop-off action when we say "agent does drop-off". Let G be a unit-cost graph (see Figure 2). We assume that agent a_1 starts at $l_0(a_1) = v_2$ and agent a_2 starts at $l_0(a_2) = v_0$. The fuel budgets are $t^{\max}(a_1) = 2$ and $t^{\max}(a_1) = 3$, and each agent consists of a single drone, i.e., $u_0(a_1) = u_0(a_2) = 1$. The reward probabilities on the maps are $\mathcal{R}(v_0) = \mathcal{R}(v_2) = 0$, $\mathcal{R}(v_3) = \mathcal{B}^{1/2,4}$, $\mathcal{R}(v_1) = \mathcal{B}^{1/2,3}$, and $\mathcal{R}(v_4) = \mathcal{B}^{1,1}$. For simplicity we assume here a zero drop-off *time, i.e., $\delta_{\mathsf{D}} = 0$. Assume also that a_1 acts before a_2 in the tie-breaking.*

We set $\delta_{\mathsf{W}} = 1$. Let us look at the optimal solution of this problem with and without the waiting action. One may check that an optimal plan with wait *is*

$$\pi^{wt} = \langle \mathsf{move}(a_1, v_2, v_3), \mathsf{wait}(a_2, v_0), \mathsf{drop\text{-}off}(a_1, v_3), \mathsf{move}(a_1, v_3, v_1), \mathsf{move}(a_2, v_0, v_1), \mathsf{drop\text{-}off}(a_1, v_1), \mathsf{drop\text{-}off}(a_2, v_1), \mathsf{move}(a_2, v_1, v_4), \mathsf{drop\text{-}off}(a_2, v_4) \rangle.$$

The plan with no waits $\pi^{no\text{-}wt}$ is the same as π^{wt}, but with the action $\mathsf{wait}(a_2, v_0)$ removed. There are four equiprobable realizations of the map, we evaluate $z(\pi^{no\text{-}wt})$ and $z(\pi^{wt})$ for all realisations:

$\mathbb{P}$	$s^0_{\mathsf{rew}}(v_1)$	$s^0_{\mathsf{rew}}(v_3)$	$z(\pi^{no\text{-}wt})$	$z(\pi^{wt})$
$1/4$	0	0	0	0
$1/4$	3	0	1	0
$1/4$	0	4	0	0
$1/4$	3	4	1	1
$\mathbb{E}$			$1/2$	$1/4$

In Figure 2 we can see that a_2 has to wait for a_1 to make a drop-off *to obtain a better solution for the problem. Thus, the* wait *action is required to synchronise the agents.*[3]

Determinization of the Problem

So far, we have defined a stochastic planning problem. Thus, the techniques that come to mind naturally to address it are stochastic planning techniques like value iteration (Bellman 1957), policy iteration (Howard 1960), or Monte Carlo Tree Search (Kocsis and Szepesvári 2006; Keller and Helmert 2013). Instead, we introduce a method for representing the DRONDEL problem as a deterministic search problem. This allows us to use techniques like Breadth First Search and Branch and Bound (Land and Doig 1960) to tackle this problem. As we see in the empirical evaluation, this pays off.

[3]Note that a similar example can be constructed for an undirected graph.

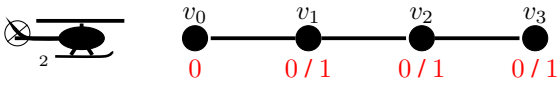

Figure 3: The graph G and the rewards in Example 2.

For each agent and reward location, we maintain a separate factored probabilistic space and use dynamic programming to compute updates for these distributions. We delve into this technique shortly, but first present an example.

Example 2. *Let $A = \{a\}$ be a single agent in the* DRONDEL *problem. Let G be a unit-cost four vertex line graph (see Figure 3). We assume that the agent starts at $l_0(a) = v_0$, has a fuel budget of $t^{\max}(a) = 3$, and has $u_0(a) = 2$ drones. Let $\mathcal{R}(v_1) = \mathcal{R}(v_2) = \mathcal{R}(v_3) = \mathcal{B}^{1/2,1}$ and $\mathcal{R}(v_0) = 0$ be the initial reward distributions. For simplicity we assume here* drop-off *and* wait *times of* 0.

There are 8 equiprobable map realisations. Intuitively, finding an optimal solution for the problem is simple: go to v_1, make a drop-off*, go to v_2, make a* drop-off*, go to v_3 and make a* drop-off *(that is, if there are still drones left). Note that the plan is non-conditional (it does not branch) – the* drop-off *action is included in the plan anyway, but the effects of the* drop-off *are stochastic.*

More rigorously, we are interested in the updated utility distribution after the first two drop-off*s. After the first* drop-off*, we know for sure that the updated reward of v_1 is zero with probability 1. However, the utility budget of a (number of drones) can be seen as a distribution $\{(0.5, 1), (0.5, 2)\}$. After the second* drop-off *the updated reward of v_2 is zero with probability 1, and the number of drones remaining is distributed as $\{(0.25, 0), (0.5, 1), (0.25, 2)\}$. Lastly, in v_3 the reward distribution after the* drop-off *is $\{(0.875, 0), (0.125, 1)\}$ (the only realisation where there is still a reward is when all v_1, v_2, and v_3 vertices have the reward 1), and the remaining drones distribution is $\{(0.5, 0), (0.375, 1), (0.125, 2)\}$.*

Thus, the optimal solution is $z(\pi) = \sum_{v \in V} (\mathbb{E}[s^0_{\mathsf{rew}}(v)] - \mathbb{E}[s_{\mathsf{rew}}(v) \mid \pi]) = 1.5 - \mathbb{E}[s_{\mathsf{rew}}(v_3)] = 1.375$.

Note that there are two random variable types: rewards at vertices and the remaining utility budget (number of drones) for each agent. Thus, to transform the problem into its deterministic variant, we should track these two nondeterministic aspects of the states on the level of distributions. In the representation we discussed so far, $s_{\mathsf{utl}}(a)$ and $s_{\mathsf{rew}}(v)$ are responsible for keeping track of the utility budget of an agent and the current reward of the state, respectively.

The key point here is that since we are concerned with non-conditional plans – the action to execute at any given moment depends only on the current time, we can compute the distribution on each such state variable, following any such plan from the initial state. Consequently, to transition to the dynamic programming paradigm (DRONDELDP), the necessary steps involve substituting the instantiation of $s_{\mathsf{utl}}(a)$ and $s_{\mathsf{rew}}(v)$ with their respective probability distributions, as well as revising the mechanics underlying the

drop-off action. We denote the states of the augmented problem by $\bar{s}$ and the respective probability spaces by $\bar{s}_{\mathsf{utl}}(a)$ and $\bar{s}_{\mathsf{rew}}(v)$. The deterministic parts of the states in the original problem remains the same, i.e., $\bar{s}_{\mathsf{loc}} = s_{\mathsf{loc}}$ and $\bar{s}_{\mathsf{T}} = s_{\mathsf{T}}$. The action mechanics associated with this part of the state remains the same – the agents fly to location and expend their fuel exactly as they did in the original problem.

To represent the stochastic components of the states as distributions, we first need to define the event spaces for the respective distributions. For the rewards on the map, this is straightforward; for a state v with the Bernoulli reward distribution $\mathcal{R}(v_1) = \mathcal{B}^{p,r}$, the event space is $0, r$. Representing the distribution of the utility budget is slightly more intricate. Let $u_0(a)$ be the initial budget, and assume $u_0(a) < |V|$. Otherwise, the agent can make a drop-off in any location on the graph. Given that each time an agent a collects a reward, its utility budget decreases by one, and it cannot go below zero, the event space for the distributions representing the utility budget of a is $\{0, \ldots, u_0(a)\}$. We note that the event spaces of $s_{\mathsf{utl}}(a)$ and $s_{\mathsf{rew}}(v)$ never change in every state reachable from s^0, though some of these event spaces may degenerate with respect to the probability function.

Note that the stochastic parts of the state are affected only by the drop-off action. Thus, it remains to initialize the distributions on these event spaces, and then track the changes introduced by the drop-off action.

The distributions in the initial state are initialized to $\bar{s}^0_{\mathsf{rew}}(v) = \mathcal{R}(v)$ and $\bar{s}^0_{\mathsf{utl}}(a) = \{(1, u_0(a))\} \cup \{(0, i)\}_{i=0}^{u_0(a)-1}$. Here $\{(1, u_0(a))\}$ represents the fact that the agent a has $u_0(a)$ drones with probability one.

The update made by the drop-off action relies on the distributions in the parent state. Let $\bar{s}$ be a state of the problem, where agent a executes the drop-off action in location v, and let $\bar{s}'$ be the resulting state. As we mentioned before this action affects only $\bar{s}'_{\mathsf{utl}}(a)$, $\bar{s}'_{\mathsf{rew}}(v)$, and $\bar{s}'_{\mathsf{T}}(a)$, where the fuel amount left to the agent to fly is updated as in the original problem, $\bar{s}'_{\mathsf{T}}(a) = \bar{s}_{\mathsf{T}}(a) + \delta_{\mathsf{D}}$. Thus, assume that $\bar{s}_{\mathsf{rew}}(v) = \mathcal{B}^{p,r}$ and $\bar{s}_{\mathsf{utl}}(a)$ stores the distribution $\{(p_i, i)\}_{i=0}^{u_0(a)}$, where $\sum_{i=0}^{u_0(a)} p_i = 1$. Then, to compute $\bar{s}'_{\mathsf{utl}}(a) = \mathcal{B}^{p',r}$ and $\bar{s}'_{\mathsf{utl}}(a) = \{(p'_i, i)\}_{i=0}^{u_0(a)}$ all we need is compute the respective probabilities p' and $\{p'_i\}_{i=0}^{u_0(a)}$.

Let $\bar{s}'_{\mathsf{rew}}(v) = \mathcal{B}^{p',r}$ be the Bernoulli distribution of the reward in vertex v in the resulting sate s'. Then, $p' := pp_0$, where p' is the probability to encounter the reward r at vertex v after the action $\mathsf{drop\text{-}off}(a, v)$. Similarly, the probabilities of the utility budget $\{p'_i\}_{i=0}^{u_0(a)}$ can be represented as

$$p'_i := \begin{cases} p_0 + p_1 p & \text{if } i = 0 \\ p_{u_0(a)}(1-p) & \text{if } i = u_0(a) \\ p_i(1-p) + p_{i+1}p & \text{otherwise,} \end{cases}$$

where p'_i is the probability of the agent a having i drones left after a makes a drop-off at v.

It is quite obvious that the state spaces of DRONDEL and DRONDELDP problems are different. Yet, since we are interested in planning over the state spaces we will compare only the states that are reachable in both problems from the initial state s^0. By $\bar{s}^0$ we denote the initial state of DRONDELDP that differs from s_0 only syntactically, cf. $u_0(a)$ and $\{(1, u_0(a))\} \cup \{(0, i)\}_{i=0}^{u_0(a)-1}$. Given this initial state, we aim to show that the distribution of the realizations of state s in DRONDEL given a path π that leads from s^0 to s coincides with the distributions in $\bar{s}$ – the state we achieved in DRONDELDP by applying π to $\bar{s}^0$.

Theorem 2. *Let π be a path from the initial state s^0 to some state s in* DRONDEL. *Let $\bar{s}^0$ be the determinized counterpart of s^0. Then, π is applicable from $\bar{s}^0$ and leads to $\bar{s}$ s.t.*

1. *$s_{\mathsf{T}}(a) = \bar{s}_{\mathsf{T}}(a)$ and $s_{\mathsf{loc}}(a) = \bar{s}_{\mathsf{loc}}(a)$ for each $a \in A$;*
2. *for each $v \in V$ and $\bar{s}_{\mathsf{rew}}(v) = \mathcal{B}^{p,r}$ it holds:*

$$\mathbb{P}(s_{\mathsf{rew}}(v) = r \mid \pi) = p \text{ and } \mathbb{P}(s_{\mathsf{rew}}(v) = 0 \mid \pi) = 1 - p;$$

3. *for each $a \in A$, $\bar{s}_{\mathsf{utl}}(a) = \{(p_i, i)\}_{i=0}^{u_0(a)}$, and each i:*

$$\mathbb{P}(s_{\mathsf{utl}}(a) = i \mid \pi) = p_i.$$

Proof. Let π be a sequence of actions leading from s^0 to s. Since by definition $s^0_{\mathsf{T}} = \bar{s}^0_{\mathsf{T}}$ and $s_{\mathsf{loc}} = \bar{s}_{\mathsf{loc}}$, and the actions in both settings act the same on these parts of the states, we have 1. by induction. Using the same induction we want to prove that conditions 2. and 3. hold after any step of π.

Base: Recognize that rewards and budget utilities are altered solely by the drop-off action. Let a_{df} be the first agent to execute a drop-off, located at v_{df}. Let π_0 be the prefix of π that ends with the first drop-off action and $\mathcal{R}(v_{\mathsf{df}}) = \mathcal{B}^{p,r}$. Examine the distributions of utilities and rewards at s after this drop-off action.

$\bar{s}_{\mathsf{utl}}(a_{\mathsf{df}}) = \{(u_0(a_{\mathsf{df}}), 1-p), (u_0(a_{\mathsf{df}}) - 1, p), \ldots, (0, 0)\}$,
$\bar{s}_{\mathsf{rew}}(v_{\mathsf{df}}) = \mathcal{B}^{0,r} = 0$. And at the same time.
$\mathbb{P}(s_{\mathsf{utl}}(a_{\mathsf{df}}) = u_0(a_{\mathsf{df}}) \mid \pi_0) = 1 - p$,
$\mathbb{P}(s_{\mathsf{utl}}(a_{\mathsf{df}}) = u_0(a_{\mathsf{df}}) - 1 \mid \pi_0) = p$, and
$\mathbb{P}(s_{\mathsf{rew}}(v) = 0 \mid \pi_0) = 1$.

Step. We assume that for each prefix π_i where $i < n$ our theorem holds. Now lets assume that n-th drop-off action is happening in vertex v^n_{df} by agent a^n_{df}. Once again we denote:

$\bar{s}_{\mathsf{utl}}(a_{\mathsf{df}}) = \{(u_0(a_{\mathsf{df}}), 1-p), (u_0(a_{\mathsf{df}}) - 1, p), \ldots, (0, 0)\}$,
$\bar{s}_{\mathsf{rew}}(v_{\mathsf{df}}) = \mathcal{B}^{p,r} = pr$. And at the same time.
$\mathbb{P}(s_{\mathsf{utl}}(a_{\mathsf{df}}) = u_{n-1}(a_{\mathsf{df}}) \mid \pi_n) = 1 - p$,
$\mathbb{P}(s_{\mathsf{utl}}(a_{\mathsf{df}}) = 0 \mid \pi_n) = \mathbb{P}(s_{\mathsf{utl}}(a_{\mathsf{df}}) =$
$= 0 \mid \pi_{n-1}) + \mathbb{P}(s_{\mathsf{utl}}(a_{\mathsf{df}}) = 1 \mid \pi_{n-1}) * p$
$\mathbb{P}(s_{\mathsf{utl}}(a_{\mathsf{df}}) = i \mid \pi_n) = \mathbb{P}(s_{\mathsf{utl}}(a_{\mathsf{df}}) =$
$= i \mid \pi_{n-1}) * (1-p) + \mathbb{P}(s_{\mathsf{utl}}(a_{\mathsf{df}}) = i + 1 \mid \pi_{n-1}) * p$, and
$\mathbb{P}(s_{\mathsf{rew}}(v) = 0 \mid \pi_n) = \mathbb{P}(s_{\mathsf{utl}}(a_{\mathsf{df}}) > 0 \mid \pi_{n-1}) * p$.

□

Corollary 1. *Let π be a plan for* DRONDEL *that ends at s. Then, $z(\pi) = \sum_{v \in V}(\mathbb{E}[\bar{s}^0_{\mathsf{rew}}(v)] - \mathbb{E}[\bar{s}_{\mathsf{rew}}(v)])$.*

NP-Membership of DRONDEL

We aim to leverage the determinized problem to establish NP-membership for the DRONDEL decision problem. This involves demonstrating that its determinized version, DRONDELDP, has a polynomial-length solution.

Start with some basic assumptions. Let n represent the input size of the DRONDELDP problem in bits, assuming standard input representation with each number as bitwise, and each graph vertex requiring at least one bit. Thus, $u_0(a) = |V| \leq n$. The computation of the time discretization of the problem results in the waiting time δ_{W}. This computation can be done in $O(n^2)$, where n is the number of bits in the input of the problem.

We also note that transforming DRONDEL into DRONDELDP including initial sate and action application is polynomial-time in the size of the input. Thus, we may solve the problem in its deterministic form – DRONDELDP. To establish NP-membership, we employ a non-deterministic Turing machine (NTM) that "guesses" the order of locations for drop-offs for each agent, with the number of guesses not exceeding the number of vertices in the graph, denoted as $|V|$. The machine then computes the shortest path between each pair of locations using Dijkstra's algorithm, ensuring each path is no longer than $|V|$. This process is repeated for each agent in A, resulting in $O(|A||V|^2)$ actions. For the combination of individual plans to adhere to the communal time schedule, we need to synchronize these plans.

To complete the proof of NP-membership, we show that achieving synchronization among agents is feasible in polynomial time. This involves a slight modification to the plan representation. Considering that waiting from time 0 to 2^n with $\delta_{\mathsf{W}} = 1$ requires an exponential number of wait actions, we replace the standard $\mathsf{wait}(a, v)$ action with $M \cdot \mathsf{wait}(a, v)$, where $M \in \mathbb{N}_+$. This extended wait action ensures synchronization and requires a polynomial number of steps, as the NTM "guesses" the number of times it should be applied in at most n steps. Notably, synchronization is only needed at drop-offs, allowing at most $O(|A||V|^2)$ extended wait actions per agent. This yields the following theorem.

Theorem 3. DRONDEL *problem lies in NP.*

This line of arguments leads us to the following conclusion, that will help us reduce the size of the graphs we are working with, pruning some of the unnecessary vertices.

Corollary 2. *Let $G = (V, E, \mathsf{cost})$ be a graph associated with an instance of* DRONDEL *problem. Let $W \subseteq V$ be the set of vertices that are either the initial location of some agent or constitutes a location with a positive initial reward, i.e., for each $v \in W$ either $l_0(a)$ for some $a \in A$ or $\mathbb{E}[u_0(v)] > 0$. Then, we can eliminate any vertex $v' \in V$ that, upon removal, does not result in an increased distance between the vertices in W.*

Solving DRONDEL Using Search

Having described the DRONDEL problem, and an equivalent deterministic search problem DRONDELDP, we now describe the algorithms we employ to solve these problems.

Search Algorithms

UCT As DRONDEL is a stochastic planning problem, the most natural baseline would be to apply MCTS to DRONDEL directly. Specifically, we use a variant of UCT (Upper Confidence Bounds for Trees) (Kocsis and Szepesvári 2006) called MaxUCT, as detailed by Helmert and Keller (2013). The MaxUCT algorithm differs from UCT only in its use of Max-Monte-Carlo backups, where decision nodes are updated based on the value of their best child. The sole modification we introduce here is that in cases where not all children of a decision node were evaluated, we average the values of the children expanded so far.

UCT$^{\mathbf{D}}$ Although UCT and MaxUCT can solve DRONDEL, we can also use the same algorithm to solve DRONDELDP. In the deterministic problem, there is no need to sample an outcome for each action, making the search much more efficient, as we demonstrate in the Experimental Evaluation.

BFS Another benefit of the deterministic version of the DRONDELDP formulation is that we can employ algorithms which are not designed for stochastic planning. We use BFS (Breadth First Search), and show that even this uninformed algorithm is competitive with our baseline (UCT). Note that since we have an optimization problem, BFS explores the whole search space layer by later until it is exhausted. If stopped early, it reports the best solution so far.

BnB We also employ an informed search algorithm on DRONDELDP, i.e., Branch and Bound (Land and Doig 1960), a search algorithm that systematically explores the state space, while maintaining upper and lower bounds on the best possible solution. It prunes unpromising branches that cannot improve upon the current best solution. The algorithm repetitively expands states in selected branches until it exhausts all possibilities proving that the current solution at hand is optimal. The upper and lower bounds we use are described below. In our implementation, we order the node expansion according to their lower bound heuristic plus the expectation on the reward collected so far, as in A^* search (Hart, Nilsson, and Raphael 1968).

Upper and Lower Bounds

We now describe the upper and lower bounds we use in our implementation. These provide fairly loose bounds, and we leave improving them as future work.

Upper Bound Given a state s in DRONDELDP, we calculate an upper bound on its utility as follows: for each agent a and a state $\bar{s}$, we order the vertices with positive expected reward that the agent can visit based on the remaining fuel budget, $t^{\max}(a) - \bar{s}_{\mathsf{T}}(a)$. Let $(v_1, \ldots, v_m)$ be the vector containing these vertices, ordered by their expected reward $\mathbb{E}[\bar{s}_{\mathsf{rew}}(v_i)]$. Let $\bar{s}_{\mathsf{rew}}(v_i)$ be a Bernoulli r.v. of getting the reward $r(v_i)$ with probability $p(v_i)$. To consider the distribution of remaining utility budget, we either assume the best rewards are always collected or assume the agent can collect all expected rewards. Thus, the upper bound on the reward an agent can collect is given by:

$$\min \left\{ \sum_{i=1}^{\lceil \bar{s}_{\mathsf{utl}} \rceil} r(v_i), \sum_{i=1}^{m} \mathbb{E}[\bar{s}_{\mathsf{rew}}(v_i)] = \sum_{i=1}^{m} r(v_i) \cdot p(v_i) \right\}.$$

To obtain the overall upper bound, we sum the individual upper bounds over all agents.

Lower Bound In the lower bound analysis, we calculate the rewards each agent collects in a greedy fashion, ensuring that each reward is assigned to at most one agent. Agents are ordered based on their current fuel budget, $t^{\max}(a) - \bar{s}_{\mathsf{T}}(a)$, and rewards are collected starting from the agent with the lowest fuel. Let this agent be a. Assuming the agent can collect rewards in locations $(v_1, \ldots, v_m)$ in increasing order of their reward expectation due to the greedy approach. Let $\bar{s}_{\mathsf{utl}}(a) = (p_i, i)_{i=0}^{u_0(a)}$ be the utility distribution of agent a. Without loss of generality, assume $m < u_0(a)$. Then, the lower bound on the reward a can collect is:

$$\sum_{i=1}^{m-1} p_i \sum_{j=1}^{i} \mathbb{E}[\bar{s}_{\mathsf{rew}}(v_i)] + \sum_{i=m}^{u_0(a)} p_i \sum_{j=1}^{m} \mathbb{E}[\bar{s}_{\mathsf{rew}}(v_i)].$$

Since each reward is collected at most once, the lower bound is the sum of the individual lower bounds of all agents.

Experimental Evaluation

We evaluated the algorithms on a set of DRONDEL instances we generated. The experimental setting is implemented in Python (code will be released upon paper acceptance). The experiments are run on a computer with 72 CPUs (one CPU per instance), an overall memory limit of 378 GB, and a time limit of 900 seconds per instance. The evaluation is done for both anytime and optimal modes. To strengthen the baseline (UCT), the rewards are modeled as Bernoulli distributions.

Benchmarks We created several different domains. Each domain exhibits similar characteristics, where we can scale the size of the map. The domains are based on the maps that appear in Moving AI Lab (Sturtevant 2012).

Mountain-Top (MT) Reward areas are dispersed across a map, each containing rewards with greater probabilities toward the center. These maps simulate a scenario where only the approximate area of a potential survivor is known.

Full Random (FR) The reward distributions are assigned randomly at each grid cell.

Sanity Check (SC) Square maps with deterministic rewards along the edges. Agents start at a corner, and in an optimal solution traverse the borders meeting in an opposite corner.

Anti-Greedy (AG) From a starting point, two paths diverge – one laden with deterministic rewards, and the other empty, except for the final square with a low probability of a high reward. On average, the second path is superior. This domain was constructed to hinder random greedy exploration.

Anytime In anytime mode, we evaluate the quality of the resulting plan as a function of the search algorithm's runtime. Here we use all of the abovementioned algorithms.

Optimal In optimal mode, we evaluate how quickly we can find a provably optimal solution. Since pure Monte Carlo Tree Search can never guarantee optimality without additional constraints that we are not applying to the problem, we evaluate only BFS and BnB (note that without the DRONDELDP formulation, guaranteeing optimality would never be possible). Thus, we evaluate how many problems are solved optimally for different planning times.

The instances for Anytime and Optimal differ. **Anytime** comprises 135 maps with sizes ranging from 30 to 345 vertices, accommodating 1 to 9 agents, and horizons ranging from 5 to 59. In turn, **Optimal** includes 72 maps with sizes ranging from 4 to 139, 1 to 3 agents, and horizons from 5 to 25, with exceptions in anti-greedy maps. In both sets of instances, rewards for each vertex range from 0 to 7.

Results Figure 4 (a–e) shows results for anytime planning. For simplicity the algorithms maximize the rewards collected by the agents along the plan π that ends in the state s, i.e., $\max_\pi \sum_{v \in V} \mathbb{E}[s^0_{\mathsf{rew}}(v)] - \mathbb{E}[s_{\mathsf{rew}}(v) \mid \pi]$. For each DRONDEL instance, we compute a score similarly to the IPC score: q^* denotes the utility of the best known solution. The normalized utility of an algorithm yielding a plan with utility q is q/q^*. Each plot shows the average normalized utility in the domain as a function of the time limit.

The results show a significant contrast, with the baseline UCT performing much worse than its application to the deterministic problem (UCT$^{\mathrm{D}}$). Surprisingly, deterministic search algorithms like BFS and BnB outperform plain UCT, and BnB even surpasses UCT$^{\mathrm{D}}$.

Figure 4(f–h) shows results for optimal planning. As mentioned above, this would be impossible with the baseline. Even so, we can see that using the heuristics in BnB improved over the uniformed BFS.

Conclusion

We have presented a stochastic planning problem called DRONDEL, which represents a realistic problem we need to solve as part of a larger project. We showed how to address this problem using deterministic search techniques, by reformulating it as DRONDELDP. Our empirical evaluation demonstrates the benefits of this approach. In future work, we will explore how to apply our approach to other stochastic planning problems.

References

Aggarwal, S.; Gupta, P.; Mahajan, N.; Balaji, S.; Singh, K. J.; Bhargava, B.; and Panda, S. 2023. Implementation of drone based delivery of medical supplies in North-East India: experiences, challenges and adopted strategies. *Frontiers in Public Health*, 11.

Alawad, W.; Halima, N. B.; and Aziz, L. 2023. An Unmanned Aerial Vehicle (UAV) System for Disaster and Crisis Management in Smart Cities. *Electronics*, 12(4).

Andreadakis, E.; Diakakis, M.; Vassilakis, E.; Deligiannakis, G.; Antoniadis, A.; Andriopoulos, P.; Spyrou, N. I.; and Nikolopoulos, E. I. 2020. Unmanned Aerial Systems-Aided Post-Flood Peak Discharge Estimation in Ephemeral Streams. *Remote Sensing*, 12(24).

Bellman, R. 1957. A Markovian decision process. *Journal of Mathematics and Mechanics*, 6(5): 679–684.

Boccardo, P.; Chiabrando, F.; Dutto, F.; Tonolo, F. G.; and Lingua, A. 2015. UAV Deployment Exercise for Mapping Purposes: Evaluation of Emergency Response Applications. *Sensors*, 15(7): 15717–15737.

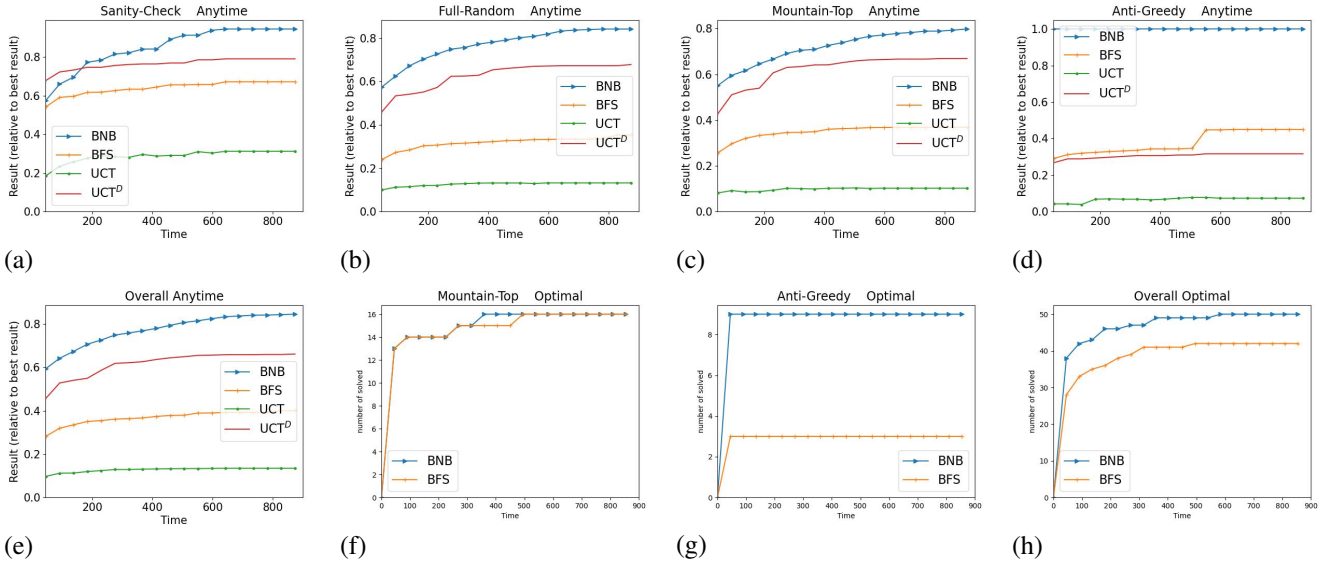

Figure 4: Plots (a) to (d) show results of anytime planning on different domain types; (e) shows average anytime result on all maps; (g) and (f) show optimal results on mountain-top and anti-greedy domains; (h) shows average of all optimal results.

Carlson, J.; and Murphy, R. R. 2005. How UGVs physically fail in the field. *IEEE Transactions on Robotics*, 21: 423–437.

Chang, K.-J.; Tseng, C.-W.; Tseng, C.-M.; Liao, T.-C.; and Yang, C.-J. 2020. Application of Unmanned Aerial Vehicle (UAV)-Acquired Topography for Quantifying Typhoon-Driven Landslide Volume and Its Potential Topographic Impact on Rivers in Mountainous Catchments. *Applied Sciences*, 10(17).

Cimatti, A.; and Roveri, M. 1999. Conformant Planning via Model Checking. In Biundo, S.; and Fox, M., eds., *ECP*, volume 1809 of *Lecture Notes in Computer Science*, 21–34. Springer.

Connor, D. T.; Wood, K.; Martin, P. G.; Goren, S.; Megson-Smith, D.; Verbelen, Y.; Chyzhevskyi, I.; Kirieiev, S.; Smith, N. T.; Richardson, T.; and Scott, T. B. 2020. Radiological Mapping of Post-Disaster Nuclear Environments Using Fixed-Wing Unmanned Aerial Systems: A Study From Chornobyl. *Frontiers in Robotics and AI*, 6.

Dovrat, D.; and Bruckstein, A. M. 2017. On Gathering and Control of Unicycle A(ge)nts with Crude Sensing Capabilities. *IEEE Intell. Syst.*, 32(6): 40–46.

Giordan, D.; Manconi, A.; Facello, A.; Baldo, M.; dell'Anese, F.; Allasia, P.; and Dutto, F. 2015. Brief Communication: The use of an unmanned aerial vehicle in a rockfall emergency scenario. *Natural Hazards and Earth System Sciences*, 15(1): 163–169.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. Syst. Sci. Cybern.*, 4(2): 100–107.

Howard, R. A. 1960. *Dynamic Programming and Markov Processes*. Cambridge, MA: MIT Press.

Jiménez-Jiménez, S. I.; Ojeda-Bustamante, W.; Ontiveros-Capurata, R. E.; and de Jesús Marcial-Pablo, M. 2020. Rapid urban flood damage assessment using high resolution remote sensing data and an object-based approach. *Geomatics, Natural Hazards and Risk*, 11(1): 906–927.

Keller, T.; and Helmert, M. 2013. Trial-Based Heuristic Tree Search for Finite Horizon MDPs. In Borrajo, D.; Kambhampati, S.; Oddi, A.; and Fratini, S., eds., *ICAPS*. AAAI.

Kocsis, L.; and Szepesvári, C. 2006. Bandit Based Monte-Carlo Planning. In Fürnkranz, J.; Scheffer, T.; and Spiliopoulou, M., eds., *ECML*, volume 4212 of *Lecture Notes in Computer Science*, 282–293. Springer.

Land, A. H.; and Doig, A. G. 1960. An Automatic Method of Solving Discrete Programming Problems. *Econometrica*, 28(3): 497–520.

Lei, T.; Zhang, Y.; Jingxuan, L.; Pang, Z.; Fu, J.; Kan, G.; Qu, W.; and Yang, W. 2018. The application of UAV remote sensing in mapping of damaged buildings after earthquakes. In *ICDIP*, volume 10806, 1080651–. SPIE.

Marfai, M. A.; Fatchurohman, H.; and Cahyadi, A. 2019. An Evaluation of Tsunami Hazard Modeling in Gunungkidul Coastal Area using UAV Photogrammetry and GIS. Case Study: Drini Coastal Area. *E3S Web of Conferences*.

Rottondi, C.; Malandrino, F.; Bianco, A.; Chiasserini, C. F.; and Stavrakakis, I. 2021. Scheduling of emergency tasks for multiservice UAVs in post-disaster scenarios. *Computer Networks*, 184: 107644.

Schaefer, M.; Teeuw, R.; Day, S.; Zekkos, D.; Weber, P.; Meredith, T.; and van Westen, C. 2020. Low-cost UAV surveys of hurricane damage in Dominica: automated processing with co-registration of pre-hurricane imagery for change analysis. *Natural hazards*, 101(3): 755–784.

Sturtevant, N. 2012. Benchmarks for Grid-Based Pathfinding. *Transactions on Computational Intelligence and AI in Games*, 4(2): 144 – 148.

Tarígan, A. P. M.; Suwardhi, D.; Fajri, M. N.; and Fahmi, F. 2017. Mapping a Volcano Hazard Area of Mount Sinabung Using Drone: Preliminary Results. *IOP Conference Series: Materials Science and Engineering*, 180(1): 012277.

Tran, D. Q.; Park, M.; Jung, D.; and Park, S. 2020. Damage-Map Estimation Using UAV Images and Deep Learning Algorithms for Disaster Management System. *Remote Sensing*, 12(24).

Watson, C. S.; Kargel, J. S.; and Tiruwa, B. 2019. UAV-Derived Himalayan Topography: Hazard Assessments and Comparison with Global DEM Products. *Drones*, 3(1).

Yakushiji, K.; Fujita, H.; Murata, M.; Hiroi, N.; Hamabe, Y.; and Yakushiji, F. 2020. Short-Range Transportation Using Unmanned Aerial Vehicles (UAVs) during Disasters in Japan. *Drones*, 4(4).