# Robust Shortest Path with Incremental Information Revelation

Edwin Meriaux, Aditya Mahajan

***Abstract*— In this paper, we consider the problem of finding the shortest path in a graph when there is aleatoric uncertainty about the presence and/or cost of certain edges. We investigate hybrid path planning, in which an agent observes the uncertain information as it traverses the graph and may adapt to the new information. We model this problem as a robust partially observable Markov decision process (robust POMDP) and identify an information state for dynamic programming decomposition. We propose a series of pruning steps, which truncate the state space based on the relationship between the cost and the uncertainty. We then show how to adapt the Neural Monte Carlo Tree Search (Neural MCTS) algorithm to obtain an approximate solution. Finally, we present a numerical study to illustrate the effectiveness of the proposed approach.**

## I. Introduction

Consider the problem of dispatching first responders (e.g., ambulances, fire trucks, etc.) during a natural disaster. It is critical to determine routes to ensure that first responders can arrive as quickly as possible, but the road network may be damaged, and real-time information from cameras and sensors might not be available. However, the dispatcher may have some information on which roads could potentially be damaged based on historical data on road conditions during past disasters, such as floods [1], [2]. As the first responders travel from the source to the destination, new information about road accessibility may become available, and they must update their routes based on this new information.

Shortest path problems arise in a variety of applications including robotics, communication networks, and search and rescue, and others. For deterministic models, exact solutions may be computed using Dijkstra's algorithm [3] or its approximations such as the A* search [4]. These algorithms can be generalized to stochastic shortest path problems as well [5]. However, all these classical algorithms assume that there is no aleatoric uncertainty about the model.

In many models there is aleatoric uncertainty about the model, either about the existence of the edges, or the cost of the edges, or the transition probabilities. Such problems are formulated as robust shortest path (RSP) problems [6] where the uncertainty across different options could be modeled as either independent [6], [7] or correlated [8].

RSP problems are often modeled as an adversarial zero-sum game against nature, where the agent seeks to identify the shortest cost path and nature selects a realization of the uncertainty that maximizes the agent's cost. The solution approaches can be categorized based on when the solution is computed as follows: (i) *Offline path planning*, which seeks to identify the path with the worst case cost prior to the agent acting in the environment [9]. Such problems can be posed as robust Markov decision process (RMDP) [10] but RSPs do not usually satisfy the $(s, a)$-rectangularity assumptions needed for RMDPs to have efficient solutions [11], [12]. Some iterative heuristic algorithms for offline RSPs have been proposed in the literature [13], but in general the solution is NP-hard [14]. (ii) *Online path planning*, which does not assume any prior knowledge on the uncertainty [15]. The proposed solution approaches predict the edge costs and iteratively improve these predictions as the edge costs are revealed [16]. (iii) *Hybrid path planning*, generates a general solution before the agent acts in the environment and then update the solution as additional information is revealed [17]. A commonly used algorithm is D* [17], which is a variant of A* search that optimistically assumes the best case scenario and selects a path according to that. When new information becomes available, the agent re-plans according to the new optimistic best case scenario. Various extensions of D* algorithm exist in the literature, including Focused D* [18], D* Lite [19], Lifelong Planning A* [20], which all follow the same general framework as D* but use different heuristics to speed up computations.

Motivated by dispatch of first-responders during natural disasters, we are interested in hybrid path planning for RDP with worst case performance guarantees. Heuristics such as D* do not guarantee that the proposed solution provides the optimal (or close to optimal) worst case performance. Our main contributions are as follows. We show that hybrid path planning can be modeled as a robust partially observable MDP (robust POMDP) [10], [21]. Exact solution approaches for robust POMDPs suffer from the curse of dimensionality. To circumvent these computational limitations, we propose a series of pruning steps which truncate the state spaces based on the relationship between cost and uncertainty. In addition, we show how to adapt Neural Monte Carlo Tree Search (MCTS) methods [22] to efficiently solve hybrid path planning in RSP. We illustrate the effectiveness of the proposed solution approach via a numerical study of different environments of increasing complexity.

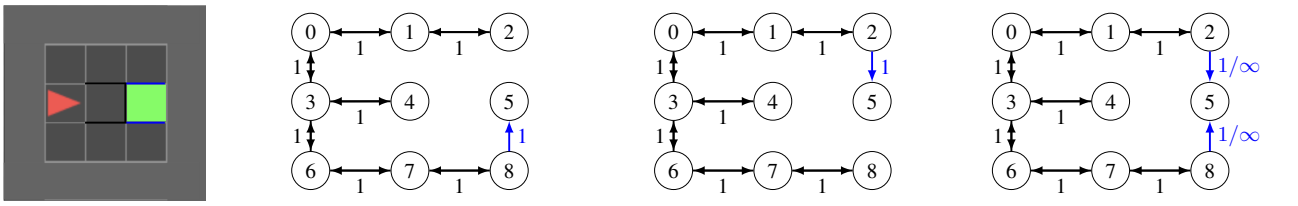
## II. Preliminaries on Graphs

We start with some terminology and notation from graph theory that are used in this paper.

- A **weighted directed graph** $\mathcal{G} = \{\mathcal{N}, w\}$ is given by a finite set of vertices $\mathcal{N}$ and a weight function $w : \mathcal{N} \times \mathcal{N} \mapsto \mathbb{R} \cup \{+\infty\}$.

The authors are with the department of Electrical and Computer Enigneering, McGill University, Montreal, Canada. Emails: edwin.meriaux@mail.mcgill.ca, aditya.mahajan@mcgill.ca

This research was supported in part by NSERC Alliance Grant. In addition, EM was supported in part by MITACS Globalink Award and McGill Graduate Mobility Award.




(a) A gridworld with uncertain edges

(b) Uncertain graph $\mathcal{G}_1$.

(c) Uncertain graph $\mathcal{G}_2$.

(d) Compact representation of $\{\mathcal{G}_1, \mathcal{G}_2\}$

Fig. 1: Uncertain graphs for Example 1.

- The weight function $w$ implicitly encodes the edges of the graph. In particular, for $n, m \in \mathcal{N}$, if $w(n,m) < \infty$ then $(n,m)$ is an edge of the graph, otherwise it is not. For this reason, we do not explicitly model the set of edges in the graph.
- A weighted graph $\mathcal{G} = (\mathcal{N}, w)$ is said to be **undirected** if for all $n, m \in \mathcal{N}$, $w(n,m) = w(m,n)$.
- Given a graph $(\mathcal{N}, w)$, the set of **out-neighbors** of a vertex $n \in \mathcal{N}$ is defined as:

$$\mathcal{N}^+(n) := \{m \in \mathcal{N} : w(n,m) < \infty\}. \tag{1}$$

- Given two vertices $n, m \in \mathcal{N}$, we say that there is a **path** between $n$ and $m$ if there exists an integer $k$ and vertices $n_1, \dots, n_k \in \mathcal{N}$ such that $n_1 = n$, $n_k = m$ and for all $i \in \{1, \dots, k-1\}$, $n_{i+1} \in \mathcal{N}^+(n_i)$.
- A path is said to be **loopless** (or without cycles) if all the vertices in the path are unique.

## III. Problem Formulation

As mentioned in the Introduction, we are interested in finding the shortest path from a source to a destination vertex when there is uncertainty about the weight function of the graph.

We model this uncertainty as follows. The agent knows that the true graph lies in an uncertain set of $L$ weighted directed graphs, denoted by $\mathcal{G}_1, \dots, \mathcal{G}_L$, where for $\ell \in \mathcal{L} := \{1, \dots, L\}$, $\mathcal{G}_\ell := (\mathcal{N}, w_\ell)$, $w_\ell : \mathcal{N} \times \mathcal{N} \to \mathcal{C}$, where $\mathcal{C}$ is a finite subset of $\mathbb{R}_{>0} \cup \{+\infty\}$ that contains $+\infty$. The true graph is denoted by $G_{\ell^*}$, $\ell^* \in \mathcal{L}$. The agent knows the uncertain set $\mathcal{L}$ but does not know the index $\ell^*$ of the true graph.

A source vertex $n^s \in \mathcal{N}$ and a destination node $n^d \in \mathcal{N}$ are given and it is assumed that the following assumption holds.

**Assumption 1** *For each graph $\mathcal{G}_\ell$, $\ell \in \mathcal{L}$, there exists a path from $n^s$ to $n^d$.*

The agent starts with the knowledge of just $\mathcal{G}_1, \dots, \mathcal{G}_L$. Additional information about the local neighborhood becomes available incrementally to the agent as it traverses the graph. In particular, the agent knows where the vertex of the graph is at, and the local neighborhood of that node (in the true graph $\mathcal{G}_{\ell^*}$). This can be modeled by assuming that at each time $t$, the agent knows the node $n_t$ and obtains an observation $o_t = \mathcal{O}(\ell^*, n_t)$ where

$$\mathcal{O}(\ell, n) := \bigcup_{m \in \mathcal{N}_\ell^+(n)} \{(m, w_\ell(n,m))\}, \quad \ell \in \mathcal{L}, n \in \mathcal{N}. \tag{2}$$

We use $\Theta = \{\mathcal{O}(\ell, n) : \ell \in L, n \in \mathcal{N}\}$ to denote the set of all possible observations.

The agent can construct the out-neighborhood of $n_t$ from the observation $o_t$ as follows:

$$\mathcal{A}(o_t) := \{m \in \mathcal{N} : (m, c) \in o_t \text{ and } c < \infty\} = \mathcal{N}_{\ell^*}^+(n_t). \tag{3}$$

Based on the history of these observations, the agent chooses an action $a_t \in \mathcal{A}(o_t) = \mathcal{N}_{\ell^*}^+(n_t)$ to decide where to go next. The actions are chosen according to a policy $\boldsymbol{\pi} = (\pi_1, \pi_2, \dots, \pi_t, \dots)$ where:

$$\pi_t : (n_{1:t}, o_{1:t}, a_{1:t-1}) \mapsto a_t, \text{such that } a_t \in \mathcal{A}(o_t). \tag{4}$$

Let $\boldsymbol{\Pi}$ denote the set of all such policies.

Given policy $\boldsymbol{\pi} \in \boldsymbol{\Pi}$ and $\ell \in \mathcal{L}$, let $T_\ell^{\boldsymbol{\pi}}$ denote the first time when an agent starting at vertex $n^s$ in graph $\mathcal{G}^\ell$ and following policy $\boldsymbol{\pi}$ reaches the destination $n^d$.

**Definition 1 (Proper Policy)** A policy $\boldsymbol{\pi} \in \boldsymbol{\Pi}$ is called **proper** if for every $\ell \in \mathcal{L}$, $T_\ell^{\boldsymbol{\pi}} < \infty$.

**Assumption 2** *There exists a policy $\boldsymbol{\pi} \in \boldsymbol{\Pi}$ that is proper.*

**Remark 1** Suppose each graph $\mathcal{G}_\ell$, $\ell \in \mathcal{L}$ has the property that for every $n, m \in \mathcal{N}$, $m \neq n^d$, such that $w_\ell(n,m) < \infty$, we have $w_\ell(m,n) < \infty$. Then, Assumption 1 implies Assumption 2.

The cost of a policy $\boldsymbol{\pi} \in \boldsymbol{\Pi}$ in graph $\mathcal{G}_\ell$, $\ell \in \mathcal{L}$, when the agent starts at $n^s$ and ends at $n^d$ is given by:

$$J_\ell^{\boldsymbol{\pi}} := \sum_{t=1}^{T_\ell^{\boldsymbol{\pi}}} w_\ell(n_t, a_t). \tag{5}$$

The worst-case cost of a policy $\boldsymbol{\pi} \in \boldsymbol{\Pi}$ when the agent starts at $n^s$ and ends at $n^d$ is given by:

$$J^{\boldsymbol{\pi}} := \max_{\ell \in \mathcal{L}} J_\ell^{\boldsymbol{\pi}} \tag{6}$$

We are interested in the following optimization problem.

**Problem 1** *Given $\mathcal{N}$, $n^s, n^d \in \mathcal{N}$, and uncertain graphs $\mathcal{G}_1, \dots, \mathcal{G}_L$, find a policy $\boldsymbol{\pi} \in \boldsymbol{\Pi}$ of the form in (4) to*

*minimize $J^{\boldsymbol{\pi}}$ given by* (6)*, i.e., solve*

$$\min_{\boldsymbol{\pi}\in\boldsymbol{\Pi}} \max_{\ell\in\mathcal{L}} J_\ell^{\boldsymbol{\pi}}.$$

We now illustrate the model via a simple example.

**Example 1** Consider the grid-world shown in Fig. 1a. The objective is to find a path for an agent (denoted by the red triangle) to reach the destination (shown by the green square). The gray walls are open, the black walls are blocked, and the blue walls are uncertain: they may be either open or closed, but the agent does not know their status..

We model this uncertain shortest path problem with two uncertain graphs $\{\mathcal{G}_1, \mathcal{G}_2\}$, as shown in Figs. 1b and 1c. We will later use the more compact representation of Fig. 1d to compactly represent graphs $\{\mathcal{G}_1, \mathcal{G}_2\}$.

For this example, it is easy to verify that Assumption 1 holds. Moreover, both $\{\mathcal{G}_1, \mathcal{G}_2\}$ satisfy the condition of Remark 1. Therefore, Assumption 2 also holds.

## IV. Exact solution using dynamic programming

### A. Information State

The problem outlined in Problem 1 can be modeled as an uncertain dynamical system with a partially observed state $s_t = (\ell^*, n_t)$. At $t = 1$, the system starts in the initial state $s_1 = (\ell^*, n^s)$. For $t > 1$, the state evolves as:

$$s_{t+1} = f(s_t, a_t) \tag{7}$$

where the update function $f$ is given by

$$f((\ell, n), a) = (\ell, a), \quad \ell \in \mathcal{L}, n \in \mathcal{N}, a \in \mathcal{N}_\ell^+(n). \tag{8}$$

At each $t$, the agents gets an observation $o_t = \mathcal{O}(s_t)$ given by (2) and chooses an action $a_t$ accordingly to a policy $\boldsymbol{\pi} = (\pi_1, \pi_2, \dots)$ of the form (4). The performance of the policy is given by (6).

Following [23], we can convert the above partially observable uncertain system to a fully observable uncertain system using an information state. The information state is given by $x_t := (\mathcal{F}_t, n_t, \mathcal{A}(o_t))$ where $\mathcal{F}_t$ is the set of feasible $\ell \in \mathcal{L}$ consistent with the history of observations and actions up to time $t$. The feasible set $\mathcal{F}_t$ evolves as follows:

$$\mathcal{F}_t = \phi(\mathcal{F}_{t-1}, n_t, o_t) \tag{9}$$

where the update function $\phi$ is given by

$$\phi(\mathcal{F}, n, o) = \{\ell \in \mathcal{F} : \mathcal{O}(\ell, n) = o\}, \quad \forall \mathcal{F} \in 2^{\mathcal{L}}, n \in \mathcal{N}, o \in \Theta \tag{10}$$

Where, $2^{\mathcal{L}}$ denotes the power set of $\mathcal{L}$.

At $t = 0$, the initial uncertainty is given by $\mathcal{F}_0 = \mathcal{L}$. At $t = 1$, the agent makes an observation $o_1$ and updates the uncertain set as $\mathcal{F}_1 = \phi(\mathcal{F}_0, n_1, o_1)$. Thus, the information state at time $t = 1$ is $x_1 = (\mathcal{F}_1, n_s, \mathcal{A}(o_1))$ and for $t > 1$, evolves as

$$x_{t+1} = (\phi(\mathcal{F}_t, n_{t+1}, o_{t+1}), n_{t+1}, \mathcal{A}(o_{t+1})).$$

**Remark 2** The cost of an outgoing edge at the current agent's vertex $n_t$ with action $a_t$ is $w_{\ell^*}(n_t, a_t)$. In (10), the sets $\mathcal{F}_t$ are constructed such that the out neigborhood of every $\ell \in \mathcal{F}_t$ are the same. Thus, for any feasible action $a_t \in \mathcal{A}(n_t)$, the agent knows the cost $w_{\ell^*}(n_t, a_t)$.

### B. Dynamic Programming Solution

**Theorem 1** *Suppose Assumptions 1 and 2 hold. Let $V$ be the unique bounded solution of the following fixed point equation, called the dynamic program:*

$$V(\mathcal{F}, n, \mathcal{A}(o)) = \min_{a\in\mathcal{A}(o)} \max_{\ell\in\mathcal{F}} \Big\{ w_\ell(n, a) + V(\phi(\mathcal{F}, a, \mathcal{O}(\ell, a)), a, \mathcal{A}(\mathcal{O}(\ell, a))) \Big\}. \tag{11}$$

*Define $\pi^*(\mathcal{F}, n, \mathcal{A}(o))$ to be the* $\arg\min$ *of the RHS of* (11)*. Then the time homogeneous policy $\boldsymbol{\pi}^* = (\pi^*, \pi^*, \dots)$, i.e., choosing $a_t = \pi^*(\mathcal{F}_t, n_t, \mathcal{A}(o_t))$, is optimal for Problem 1.*

PROOF The dynamic programming solution follows from [23]. The analysis in [23] was for finite horizon models. Following the arguments in [24] these finite horizon results can be extended to infinite time horizons for stochastic shortest path problem under Assumptions 1 and 2.

### C. Interpretation as a two-player zero-sum game

Problem 1 is a minimax optimization problem and as such can be viewed as a two-player zero-sum game where the agent is player one and is trying to minimize the cost of going from the source to the destination while nature is player two and is trying to maximize the cost. The zero-sum game has the following salient features: it is sequential since the agents play one-by-one; it is multi-stage since the play evolves over multiple rounds; and finally, it has one-sided asymmetric information since nature's actions are not perfectly observed by the agent but agent's actions are perfectly observed by nature. The dynamic program of Theorem 1 computes a minimax equilibrium of the game.

### D. Value Iteration

The dynamic program of Theorem 1 can be compactly written in terms of Bellman operators. In particular, for any value function $V$ define the Bellman operator

$$[\mathcal{B}V](\mathcal{F}, n, o) = \min_{a\in\mathcal{A}(o)} \max_{\ell\in\mathcal{F}} \Big\{ w_\ell(n, a) + V(\phi(\mathcal{F}, a, \mathcal{O}(\ell, a)), a, \mathcal{A}(\mathcal{O}(\ell, a)) \Big\} \tag{12}$$

and the greedy policy

$$[\Lambda V](\mathcal{F}, n, o) = \arg \min_{a\in\mathcal{A}(o)} \max_{\ell\in\mathcal{F}} \Big\{ w_\ell(n, a) + V(\phi(\mathcal{F}, a, \mathcal{O}(\ell, a)), a, \mathcal{A}(\mathcal{O}(\ell, a)) \Big\}. \tag{13}$$

Then, the dynamic program of Theorem 1 can be written as

$$V = \mathcal{B}V.$$

Value iterature (VI) is the simplest iterative algorithm to compute the fixed point of the above fixed-point equation. Given an accuracy level $\varepsilon > 0$, the VI algorithm proceeds as follows.

1) Initialize $k = 0$ and $V_0 \equiv 0$.
2) Compute $V_{k+1} = \mathcal{B}V_k$ and $\pi_k = \Lambda V_k$.
3) If $\|V_{k+1} - V_k\|_\infty \leq \varepsilon$, return $\pi_k$ and stop. Else set $k = k + 1$ and return to step 2.

### *E. Pruning of the uncertainty graphs*

The complexity of VI depends on the size of the set of all feasible information states. In this section, we propose a method to prune the graphs $\mathcal{G}_1, \dots, \mathcal{G}_L$, which reduces the number of feasible information states and thereby improves the computational complexity of the algorithm. We start with some definitions.

For any path $p$ in graph $\mathcal{G}_\ell$, $\ell \in \mathcal{L}$, let $C_\ell(p)$ denote the cost of the path. If the path contains an edge with infinite weight, then its cost is infinity. For any path $p$, define

$$C_{\max}(p) = \max_{\ell \in \mathcal{L}} C_\ell(p)$$

For any edge $(n, m)$, $n, m \in \mathcal{N}$, let $D_\ell(n, m)$ denote the cost of the shortest path from the source to the destination in graph $\mathcal{G}_\ell$ that contains the edge $(n, m)$. For any edge $(n, m)$, define:

$$D_{\min}(n, m) = \min_{\ell \in \mathcal{L}} D_\ell(n, m).$$

We consider three types of pruning:

1) **Pruning of $\mathcal{L}$.** For every $\ell_\circ \in \mathcal{L}$, remove the graph $\mathcal{G}_{\ell_\circ}$ if there exists a graph $G_{\ell'}$, $\ell' \in \mathcal{L}$, $\ell' \neq \ell_\circ$, such that $w_{\ell_\circ}(n, m) \leq w_{\ell'}(n, m)$, for all vertices $n, m \in \mathcal{N}$.
2) **Pruning of Edges in $\mathcal{G}_\ell, \ell \in \mathcal{L}$.** Take a vertex $n \in \mathcal{N}$ and a loopless path $p_\circ$ from $n$ to the destination such that $C_{\max}(p_\circ) < \infty$. Consider an edge $(n, m)$, which does not lie on $p_\circ$, such that

   $$D_{\min}(n, m) > C_{\max}(p_\circ).$$

   Then remove the edge $(n, m)$ from all graphs $\mathcal{G}_\ell$, $\ell \in \mathcal{L}$. Note that removing an edge is equivalent to setting its weight to $+\infty$.
3) **Pruning of Vertices in all $\mathcal{G}_\ell$, $\ell \in \mathcal{L}$.** Remove all vertices $n \in \mathcal{G}_\ell$ that do not belong to any finite-cost path from the source $n^s$ to the destination $n^d$ in any graph $\mathcal{G}_\ell$, $\ell \in \mathcal{L}$.
   Removing a vertex means that we remove it from the node set $\mathcal{N}$.

Note that the pruning steps can be repeated. So, when we start with a problem, we apply the pruning steps one by one, until they no longer lead to a simplification.

**Proposition 1** *The three pruning methods described above do not change the optimal solution.*

PROOF We will separately establish that each of the pruning methods do not change the optimal solution.

1) Fix an $\ell_\circ \in \mathcal{L}$. Suppose there exists an $\ell' \in \mathcal{L}$ such that $w_{\ell_\circ}(m, n) \leq w_{\ell'}(m, n)$ for all $m, n \in \mathcal{N}$. Then, for any policy $\boldsymbol{\pi}$, we have

   $$J^{\boldsymbol{\pi}}_{\ell_\circ} \leq J^{\boldsymbol{\pi}}_{\ell'}$$

   where we allow the right hand side to be infinity. Thus,

   $$\max_{\ell \in \mathcal{L}} J^{\boldsymbol{\pi}}_\ell = \max_{\ell \in \mathcal{L} \setminus \{\ell_\circ\}} J^{\boldsymbol{\pi}}_\ell$$

   Thus, nature can ignore $\ell_\circ$ while selecting its action.
2) Let $p_\circ$ and $(n, m)$ be as in the definition of pruning. Consider any information state $x = (\mathcal{F}, n, \mathcal{A}(o))$. Let $(n, m_\circ)$ be the first edge of $p_\circ$. Since $C_{\max}(p_\circ) < \infty$, we must have that $w_\ell(n, m_\circ) < \infty$ for all $\ell \in \mathcal{L}$. Therefore, $m_\circ \in \mathcal{A}(o)$ for all feasible observations $o$ at vertex $n$. Define

   $$Q(\mathcal{F}, n, \mathcal{A}(o), a) = \max_{\ell \in \mathcal{F}} \Big\{ w_\ell(n, a) + V(\phi(\mathcal{F}, a, \mathcal{O}(\ell, a)), a, \mathcal{A}(\mathcal{O}(\ell, a))) \Big\}. \quad (14)$$

   Note that $Q(x, m) \geq D_{\min}(n, m)$ (because no path for $n$ to the destination can cost less than $D_{\min}(n, m)$) and $Q(x, n) \leq C_{\max}(p_\circ)$ (because following the path $p_\circ$ is a feasible policy, hence the optimal policy must be at least as good as $p_\circ$). The fact that $D_{\min}(n, m) > C_{\max}(p_\circ)$ implies that

   $$Q(x, m) \geq D_{\min}(n, m) > C_{\max}(p_\circ) \geq Q(x, m_\circ).$$

   Hence, action $m$ is never optimal, and can therefore be eliminated
3) Under Assumption 2, an optimal policy will never visit a vertex that does not belong to any finite-cost path from the source to the destination. So, removing such a node, does not change the optimal policy.

## V. APPROXIMATE SOLUTION USING NEURAL MONTE-CARLO TREE SEARCH

It is well acknowledged that optimally solving dynamic programming equations suffers from the curse of dimensionality, even in the perfectly observed expected cost setting. This is exacerbated in the partially observed setting. Due to this fact, the dynamic programming solution of Theorem 1 can only be used to solve small toy-examples. In the case of larger models, some form of an approximate solution is needed.

In recent years, Monte-Carlo Tree Search (MCTS) has emerged as an efficient method to solve large scale dynamic programming problems [25], especially in the zero-sum game setting, with AlphaGo [22] being the most prominent example. For small-scale environments, MCTS builds a game tree and learns a value for each vertex of the game tree using a variant of upper confidence bounds (UCB) [26] called UCT (upper confidence bounds applied to trees). For larger models, it is not possible to explicitly construct a game tree. In such situations, a variant of MCTS called Neural MCTS is used, which only constructs a part of the game tree and

approximates the value of all vertices in the game tree using a deep neural network. We describe how to adapt such Neural MCTS approach to obtain an approximate solution.

**Algorithm 1** Neural MCTS

```
Initialize:
  Create root node t_0 representing initial state
  Initialize a DNN Ψ_θ with parameters θ
for each episode do
  Set t ← t_0.
  repeat
    if t is not fully expanded then
      Add a random unexpanded edge to tree
      break
    end if
    Selection action according to UCT
    Set t to be the next state.
    if t is destination then
      Set v to be cost incurred so far
      break
    else Action limit is reached
      Set v to be a large penalty
      break
    end if
  until break is called
  Backpropogate v along the nodes traversed
    and update values using temporal difference
  Update DNN weights to predict updated values.
end for
```

A Neural MCTS consists of a tree $\mathcal{T}$ and a deep neural network (DNN) that approximates the value; function of the dynamic program. The tree is a truncated approximation of the game tree corresponding to the zero-sum game between the agent and nature. In our case, the nodes of the tree at even depth (including depth 0) correspond to the moves of the agent, while the nodes at odd depth correspond to the moves of nature. The root node corresponds to the initial state of the system. The algorithm works in two steps: training and inference.

During training, a tree starting from the root node is built incrementally until a budget of a fixed number of nodes has been reached (this is called the *expansion limit*). A tree policy is used to traverse the tree and, at each node, either expand the tree by adding one or more new child nodes, or select an action by looking at the value of each of the available available child nodes and selecting one of them using UCT (upper confidence bounds for trees). The value is then backed up through the traversed path (using temporal difference learning) to update the values of all nodes along the path. The main feature of Neural MCTS is that a DNN is trained in parallel, which learns a mapping from the state of knowledge at that node to its value.

Training proceeds in multiple episodes. In each episode, the tree policy is followed either to expand the tree, or to take actions until the agent reaches the destination vertex or until a total budget on the number of action is reached (this budget is called the *action limit*). If the episode ends because of the agent reaching the destination vertex, the terminal node is given a value equal to the cost of path followed from the source until the destination (and this value is backed up as described above); if the episode ends on reaching the action limit, the terminal node is given a large value as penalty (and this value is backed up as described above). This process is repeated for large number of episodes.

During inference, the weights of the DNN are frozen and the agent chooses the greedy action at each node, according to the values learned by the DNN.

## VI. Numerical Examples

In this section, we present a simulation study to analyze the performance of the algorithms presented in this paper with a baseline algorithm.

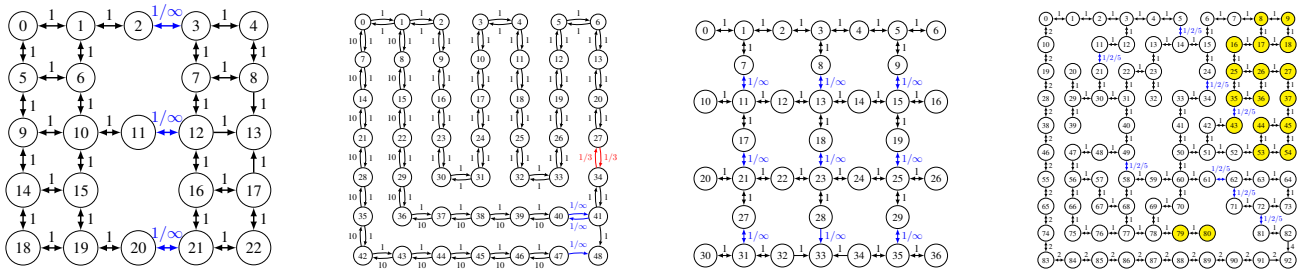
### A. Models Considered

We test the algorithms on four models with increasing complexity. The models are shown in Fig 2 using the compact graph notation, which was presented in Fig. 1d. The details of the models are as follows:

1) Model 1, as shown in Figure 2a, is a 23 vertex graph with 3 uncertain edges where at least one of these edges exist. Thus, there are $|\mathcal{L}| = 7$ possible true weights for the graph.
2) Model 2, as shown in in Figure 2b, is a 49 vertex graph with 3 uncertain edges. There is uncertainty in whether the blue edges exist or not and uncertainty in the cost of the red edges. If all the blue edges do not exist, the red edges are valued at 3, if one of the pairs of blue edges exist, then the red edges, are valued at 1. Thus, there are $|\mathcal{L}| = 4$ possible true weights for the graph.
3) Model 3, as seen in Figure 2c, is a 36 vertex graph with 9 uncertain edges where at least one edge in each row exists. Thus, there are $|\mathcal{L}| = 27$ possible true weights for the graph.
4) Model 4, as seen in Figure 2d, is a 92 vertex graph with 8 uncertain edges where there is uncertainty in the cost of the blue edges and the sum of all the weights is 30. Thus, there are $|\mathcal{L}| = 168$ possible true weights for the graph.

For each model, we run the graph pruning procedure described in Sec. IV-E.

1) For model 1, the only pruning which can be done is removing all graph weights which has more than 1 of the uncertain edges existing. This pruning comes from **Pruning of** $\mathcal{L}$. After the pruning, there are $|\mathcal{L}| = 3$ possible true weights for the graph.
2) For model 2, pruning does not simplify the graphs.
3) For model 3, pruning does not simplify the graphs.
4) In model 4, all edges highlighted in yellow in Figure 2d can be removed through a combination of **Pruning of Edges** and **Pruning of Vertices** methods. After the pruning, there are $|\mathcal{L}| = 21$ possible true weights for the graph.



(a) Model 1 (b) Model 2 (c) Model 3 (d) Model 4

Fig. 2: The four models considered in the numerical experiments.

We denote model 1 pruned by model 1p and model 4 pruned by model 4p.

### B. Algorithms Evaluated

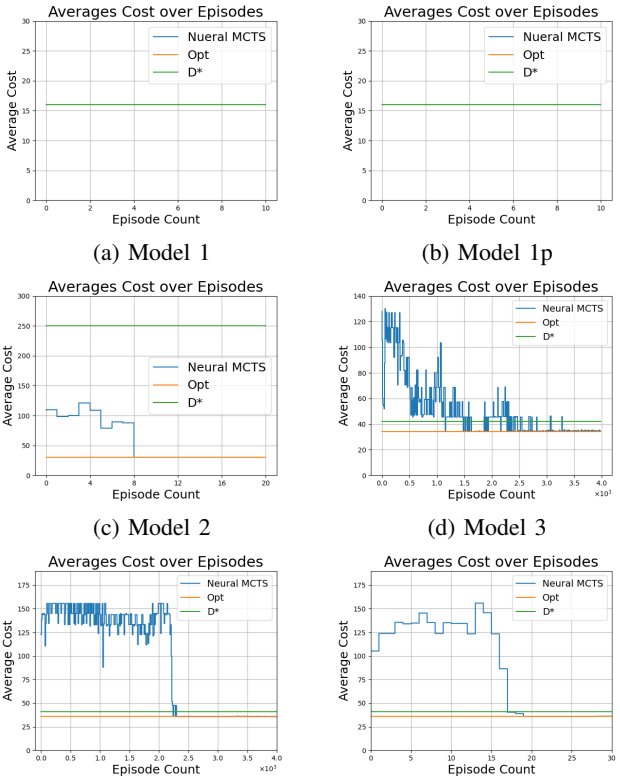
We consider the performance of the following algorithms:

1) The **Value Iteration (VI)** algorithm, as presented in Sec. IV. The size of the set of feasible states for each model is as follows:
   - Model 1 has 7 states
   - Model 1p has 3 states
   - Model 2 has 4 states
   - Model 3 has approximately $4.83 \times 10^9$ states.
   - Model 4 has approximately $3.44 \times 10^{52}$ states.
   - Model 4p has approximately $1.55 \times 10^8$ states.

   Because of the size of the state space for models 3, 4 and 4p, we cannot evaluate Value Iteration on these models.
2) The **Neural MCTS** algorithm, as presented in Sec. V. This algorithm has several hyperparameters, that we discuss below. The DNN had four layers, where the first three layers were linear layers with Relu activation function and the last layer was a softplus layer. The size of the first three layers was equal to the number of possible edges in the network (to capture the state of knowledge of each edge) plus the number of vertices (to capture the current location of the agent); the output of the final layer was a real number approximating the value of the vertex. We performed a hyperparameter search to select the learning rate of $10^{-5}$. For the MCTS component, we set the expansion depth, action limit, and exploration depth to be 50, 50 and 500, respectively. We performed a hyperparameter search to select the exploration parameter (in UCT) to 1.
3) The **D* algorithm** [17], which was explained in the Introduction.

### C. Results

We run all algorithms for all models and evaluate the worst case performance of the resultant policy, which is evaluated according to (6). The results are shown in Table I. The training of curves Neural MCTS is shown in Fig. 3 relative to D* and the optimal value which is computed via VI in Fig. 3a, 3b, 3c and via a hand-crafted selective search in Fig. 3d, 3e, 3f. The computational time for each algorithm is shown in Table II.




(a) Model 1 (b) Model 1p (c) Model 2 (d) Model 3 (e) Model 4 (f) Model 4p

Fig. 3: Neural MCTS training curves.

For the smaller models (models 1, 1p, and 2), all algorithms compute a solution. Moveover, the solution obtained by Neural MCTS is the same as the optimal solution obtained via VI. For model 2, D* obtains a worse than optimal solution; for models 1 and 1p, it obtains the same solution as VI. For the larger models (models 3, 4, and 4p), VI is infeasible Because of the size of the state space. The other algorithms do converge, with Neural MCTS outperforming D*, but this improved performance comes at the cost of increased computation time. It is worth highlighting that most of the computational time for Neural MCTS is spent during offline training (and pruning). Once trained, the inference time for

TABLE I: Worst case performance of the policy learned by each algorithm

| Model | VI | Neural MCTS | D* |
|---|---|---|---|
| 1 | 16 | $16 \pm 0$ | 16 |
| 1p | 16 | $16 \pm 0$ | 16 |
| 2 | 30 | $30 \pm 0$ | 250 |
| 3 | N/A | $34 \pm 0$ | 42 |
| 4 | N/A | $36 \pm 0$ | 41 |
| 4p | N/A | $36 \pm 0$ | 41 |

TABLE II: Computation time (in seconds) taken by each algorithm

| Model | VI | Neural MCTS | D* |
|---|---|---|---|
| 1 | 13.11 | 12.03 | 1.11 |
| 1p | 2.28 | 13.73 | 1.34 |
| 2 | 4.20 | 215.02 | 1.61 |
| 3 | N/A | 2,495.56 | 69.37 |
| 4 | N/A | 15,410.43 | 303.75 |
| 4p | N/A | 755.02 | 117.86 |

Neural MCTS is small. Thus, Neural MCTS is a scalable and effective algorithm for approximately solving large-scale robust shortest path problems.

Note that the computational time reported in Table II includes the time taken to prune the models. For small models such as model 1, pruning does not provide computational savings for Neural MCTS because the pruning time is a significant portion of the training time. However, for larger models such as model 4, pruning reduces the overall computational time by almost a factor of 20. This suggests that it is worthwhile to run the model pruning step for larger models.

## VII. Conclusion

In this paper, we formulate the robust shortest path (RSP) problem as a robust POMDP and present a dynamic programming decomposition to identify the hybrid policy with the best worst-case guarantees. We present a series of graph pruning steps that truncate the state space based on the relationship between cost and uncertainty. We then show how to adapt Neural MCTS algorithms to efficiently provide an approximate solution to the dynamic program. Our numerical experiments show that Neural MCTS effectively scales to environment with large state spaces.

The results of our paper offer both theoretical guarantees and practical methods for real-world deployment. Future work can explore extensions of this approach to other forms of uncertainty and further refine the proposed methods for broader application in uncertain dynamical systems.

## Acknowledgments

The authors are grateful to Dr. S.I. Niculescu (CentraleSupélec, CNRS) for helpful discussions and feedback.

## References

[1] D. Shiri *et al.*, "Online algorithms for ambulance routing in disaster response with time-varying victim conditions," *OR Spectrum*, pp. 1–35, 2024.

[2] L. Talarico *et al.*, "Ambulance routing for disaster response with patient groups," *Computers & operations research*, vol. 56, pp. 120–133, 2015.

[3] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[4] P. E. Hart *et al.*, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, 1968.

[5] D. P. Bertsekas and J. N. Tsitsiklis, "An analysis of stochastic shortest path problems," *Math. Oper. Res.*, vol. 16, no. 3, pp. 580–595, 1991.

[6] G. Yu and J. Yang, "On the robust shortest path problem," *Computers & Operations Research*, vol. 25, no. 6, pp. 457–468, 1998.

[7] R. Montemanni and L. M. Gambardella, "An exact algorithm for the robust shortest path problem with interval data," *Computers & Operations Research*, vol. 31, no. 10, pp. 1667–1680, 2004.

[8] M. C. Resende, *The Robust Shortest Path Problem with Discrete Data*. Ph.D. thesis, Universidade de Coimbra, 2015.

[9] P. Kamkarian and H. Hexmoor, "A novel offline path planning method," in *International Conference on Artificial Intelligence (ICAI)*, p. 10, 2015.

[10] W. Wiesemann *et al.*, "Robust Markov decision processes," *Mathematics of Operations Research*, vol. 38, no. 1, pp. 153–183, 2013.

[11] S. S. Ketkov *et al.*, "An approach to the distributionally robust shortest path problem," *Computers & Operations Research*, vol. 130, p. 105212, 2021.

[12] Y. Zhang, S. Song, Z.-J. M. Shen, and C. Wu, "Robust shortest path problem with distributional uncertainty," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 4, pp. 1080–1090, 2017.

[13] M. M. Pascoal and M. Resende, "The minmax regret robust shortest path problem in a finite multi-scenario model," *Applied Mathematics and Computation*, vol. 241, pp. 88–111, 2014.

[14] P. Zieliński, "The computational complexity of the relative robust shortest path problem with interval data," *European Journal of Operational Research*, vol. 158, no. 3, pp. 570–576, 2004.

[15] T.-W. Zhang *et al.*, "A new hybrid algorithm for path planning of mobile robot," *The Journal of Supercomputing*, vol. 78, no. 3, pp. 4158–4181, 2022.

[16] W. Gao *et al.*, "Ratio-based distortion and network distance," 2024.

[17] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *Proc. IEEE Int. Conf. Robot. Autom.*, pp. 3310–3317, IEEE, 1994.

[18] A. Stentz *et al.*, "The focussed D* algorithm for real-time replanning," in *IJCAI*, vol. 95, pp. 1652–1659, 1995.

[19] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *IEEE Trans. Robot.*, vol. 21, no. 3, pp. 354–363, 2005.

[20] S. Koenig and M. Likhachev, "Incremental A*," *Advances in neural information processing systems*, vol. 14, 2001.

[21] T. Osogami, "Robust partially observable Markov decision process," in *Intl. Conf. Machine Learning*, pp. 106–115, PMLR, 2015.

[22] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[23] D. Bertsekas and I. Rhodes, "Sufficiently informative functions and the minimax feedback control of uncertain dynamic systems," *IEEE Trans. Autom. Control*, vol. 18, no. 2, pp. 117–124, 1973.

[24] D. Bertsekas, *Reinforcement learning and optimal control*, vol. 1. Athena Scientific, 2019.

[25] C. B. Browne *et al.*, "A survey of monte carlo tree search methods," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 1, pp. 1–43, 2012.

[26] P. Auer *et al.*, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, pp. 235–256, 2002.