The Robust Canadian Traveler Problem and Its Application to Robot Routing

by

Hengwei Guo

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
University of Toronto Institute for Aerospace Studies
University of Toronto

Abstract

The Robust Canadian Traveler Problem and Its Application to Robot Routing

Hengwei Guo

Master of Applied Science

University of Toronto Institute for Aerospace Studies

University of Toronto

2018

The stochastic Canadian Traveler Problem (CTP), which finds application in robot route selection under uncertainty, aims to find the traversal policy with the minimum expected cost. This thesis extends the stochastic CTP to what we call the Robust Canadian Traveler Problem (RCTP), in which the variability of the policy cost is also considered. An optimal algorithm and an approximate algorithm are then proposed to compute the policy that has a good balance of both mean and variation of the traversal cost.

The benefit of the proposed framework versus traditional approaches is shown by doing simulations in randomly generated worlds as well as on a map of 5 km of paths built from robot field trials. Specifically, the RCTP framework is able to search for policy alternatives with significant lower worst-case cost and less computational time compared against the optimal CTP policy, but with little sacrifice on the expected cost.

Acknowledgements

This thesis was completed with the support of my supervisor, colleagues, family, and friends. I would like to take this opportunity to express my appreciation for their generous support.

First and foremost, I would like to thank my supervisor Prof. Timothy D. Barfoot for his guidance through my work. He has made a lasting influence on me, particularly in how I look at field robotics and the importance of being rigorous on details. He gave me the freedom to explore whatever research directions that intrigued me and was an irreplaceable support in guiding me on the right track. He has been a motivating mentor, and to whom I offer my deepest thanks.

Many thanks to my colleagues and friends in the Autonomous Space Robotics Lab for always being helpful and consistently showing me how things should be done.

I would also like to thank Prof. Sheila McIlraith for leading me to the world of academic research and always being resourceful. Her enthusiasm for both research and science communication is contagious.

Finally, I'm very grateful to my parents, my sister, and my friends for always being supportive to me so that I can chase my dreams.

Hengwei Guo

Contents

List of Figures

Notation

General Notation

a	:	Lower-case variables in this font are scalars.
X	:	Upper-case variables in this font are sets or spaces. (e.g., the search space of the planning algorithm).
$\mathbf{a}$	:	Lower-case variables in this font are column vectors.
$\mathbf{A}$	:	Upper-case variables in this font are matrices.

Specific Symbols

V	:	A set of vertices in a graph.
E	:	A set of edges in a graph, $E \subseteq (V \times V)$.
s	:	The starting vertex of the path planning problem, $s \in V$
t	:	The destination vertex of the path planning problem, $t \in V$

Specific Functions & Operators

$P(\cdot)$	:	The probability of an event.
$S(\cdot)$	:	The successor states of a given state.
$\mathbb{E}[\cdot]$	:	The expectation operator.
$\Sigma[\cdot]$	:	The summation operator.
$\exp(\cdot)$	:	The exponential operator.
$\lvert\cdot\rvert$	:	The cardinality of a set or a graph.

Cost Functions

$h(\mathbf{x})$	:	An heuristic estimate of the cost-to-go to the goal from a state, $\mathbf{x} \in X$.
$f(\mathbf{x})$	:	An estimate of the cost of a path from the start to the goal constrained to pass through a state, $\mathbf{x} \in X$.
$f^*(\mathbf{x})$	:	The optimal cost of a path from the start to the goal constrained to pass through a state, $\mathbf{x} \in X$.

Acrynoms

VT&R Visual Teach and Repeat

CTP Canadian Traveler Problem

RCTP Robust Canadian Traveler Problem

MDP Markov Decision Process

CMDP Constrained Markov Decision Processes

POMDP Partially-Observed Markov Decision Process

STPG Spatio-Temporal Pose Graph

MEL Multi-Experience Localization

VI Value Iteration

VO Visual Odometry

LPA* Lifelong Planning A*

ARA* Anytime Repairing A*

RRT Rapidly-exploring Random Trees

GTP Gaussian Traveler Problem

GP Gaussian Process

Chapter 1

Introduction

1.1 Motivation

Robots that navigate in real-world scenarios often face the problem of choosing the best route under uncertainty (e.g., due to incomplete knowledge of the world and noisy observation of the environment). Such uncertainties can come from factors such as sensor limitations and dynamic changes in the environment (lighting, season, terrain modification, etc.). For example, Fig. 1.1 shows a map of an extended field test that we conducted with a vision-based robot at an old gravel pit in Sudbury, Canada [43]. The map was created using the Visual Teach and Repeat (VT&R) algorithm ([18], [41], [30], [8]) by manually piloting the robot through the path network once, and then autonomous path repeating was done by localizing over visual features stored in the map. Navigating in such an unstructured environment for an extended period of time meant the robot encountered many challenges such as shifting sand with little visual texture, fast-growing vegetation that changes visual appearance, and unexpected obstacles which were not present during mapping. All these challenges can result in failures of traversing certain paths. To make more robust plans for driving through such a network of paths, we propose to gather historical data on the success of driving on individual path segments and use this in route selection.

Routing scenarios of this kind can be formalized as a stochastic version of the Canadian Traveler Problem (CTP) [39]: an agent must find the best traversal policy between two vertices in a given graph where some edges may be blocked with some known probabilities; we call

Figure 1.1: Orthomosaic imagery of the 5 km network of paths at the Ethier Sand and Gravel in Sudbury, Ontario, Canada. The map shows challenging environmental uncertainties for robust route planning: lighting change, shifting sand with little visual texture, and dense vegetation with changing appearance.

such edges *stochastic*. A stochastic edge's binary traversability status can only be revealed upon reaching one of its endpoints.

For a given routing problem, there are a number of possibilities, or realizations, of how the hidden graph may look. The solution to the CTP is a policy rather than a path; for example, first attempt path A, if path A is not traversable, then attempt path B. Most prior work on stochastic CTPs are focused on finding the policy that minimizes the expected traversal cost. However, the cost variability of attempting a stochastic policy can also be important if we are worried about the worst case over several realizations.

For instance, consider the situation in which the robot can choose from the following two policies to travel to the charging station:

- Policy 1: Travel to the charging station with 0.9 probability of traveling 1100 meters and 0.1 probability of traveling 1900 meters.

- Policy 2: Travel to the charging station with 0.2 probability of traveling 1100 meters and 0.8 probability of traveling 1200 meters.

Note that both policies have the same expected traveling distance of 1180 meters, thus would be treated equally for algorithms that only minimize the expected cost. An adventurous decision maker might choose Policy 1 since the probability of traveling the least distance is higher. However, what if the remaining battery of the robot can only let it travel for up to 1500 meters? A more risk-averse decision is to choose Policy 2, due to its lower variability around the mean.

The focus of this thesis is thus to investigate path planning algorithms that can solve the stochastic CTP in a more risk-sensitive manner - being able to find traversal policies that have a good balance between cost expectation and variability.

1.2 Contributions

The primary goal of the thesis is to present risk-sensitive algorithms that can find robust policies to travel on a network of paths with uncertainties in road blockage. Specifically, the novel contributions of this thesis are as follows:

1. Propose the Robust Canadian Traveler Problem (RCTP) that tries to find the traversal policy over uncertain maps with a good balance of both mean and variation of the traversal cost.

2. Analyze different risk-quantifying methods to evaluate policy risks.

3. Provide an offline and an online algorithm to solve the RCTP.

4. Experimental validation of the algorithms on simulated maps, as well as a map that was collected from a robot field trial.

1.3 Thesis Overview

The remainder of this thesis is organized as follows. Chapter 2 presents further background and related work. Chapter 3 formally defines the RCTP and analyses the complexity of the

problem through its Markov Decision Process (MDP) definition and AND/OR tree representation. Chapter 4 provides two modified versions of the AO* algorithm [44] to solve the RCTP, one for offline optimality and one for fast online execution. Implementation tricks will also be discussed. Chapter 5 will explain the experimental environment, and then discuss the results of the proposed algorithms versus baseline approaches. Chapter 6 will discuss the methods to integrate the proposed planner into VT&R - a vision-based robot navigation framework. Finally, Chapter 7 summarizes the conclusions and future research directions.

Chapter 2

Background and Related Work

This chapter first reviews some popular path planning algorithms in mobile robotics that aim to find a feasible, or optimal path between a starting position and a goal position in a given map. Depending on different evaluation criteria, finding the optimal solution can refer to different objectives such as finding the path that has the shortest distance, the fastest travel time, or the most likelihood of being successfully traversable.

The Canadian Traveler Problem (CTP) is introduced next. It is a generalization of the shortest path problem to graphs that are partially observable. For CTPs, the existence of some edges in the graph can only be revealed when they are being explored. Thus instead of finding the optimal path, the goal of solving the CTP is to find the optimal strategy (or policy) to travel the map, namely the agent should know how to take the best action to adjust its plan when new observations are made.

The work presented in this thesis is motivated by solving the path planning challenges in the VT&R framework, which is a vision-based robot navigation system. Thus a brief introduction about the VT&R framework and the map structure being used are also introduced. The challenge that we are trying to solve for VT&R can be expressed as an instance of the CTP. However, the goal of the classic CTP is trying to find the optimal policy that minimizes the expected traversal cost, whereas we are interested in finding more robust policies that are also sensitive to the possible cost variability. We aim to find a good balance between optimality and robustness.

2.1 Path Planning for Mobile Robots

The robotics community has committed an enormous amount of time to solving the path planning problem. There are many algorithms that solve for optimal or approximate optimal paths; each method has its own strengths and weaknesses depending on the application domains and the complexity of the problems. In this section, we will review some typical categories of planning algorithms that are related to our work.

2.1.1 Path Planning in Known Environments

In a known or fully observable environment, the robot knows the entire map of the environment before it starts traveling. Thus an optimal solution can often be found offline before the robot actually starts to move. The path planning techniques for fully observable and known environments are relatively mature and many classic algorithms were proposed. For example, Bruce-force search methods such as the classic Breadth-first search, Depth-first search, the Floyd-Warshall algorithm [15], [53], the Bellman-Ford-Moore algorithm [4], [16], [34] and many others. Though being proposed around the 1960s, Dijkstra's algorithm [11] and the A* algorithm [20] are still being widely used today for optimal pathfinding in known environments.

Dijkstra's algorithm [11] traverses the graph from a starting vertex and then expands to other vertices in the order of the vertex cost-to-come (the cost of the vertex to the start). Thus the algorithm can be used to find the shortest path between the start vertex and every other. It can also be used to find the shortest path from the start to a single destination by stopping the algorithm once the shortest path to the destination has been found. In this case, the algorithm only visits vertices that have a lower cost-to-come than the destination vertex.

Dijkstra's algorithm can be implemented by using a priority queue to store the vertices to be visited. Initially, all vertices are marked as unvisited and the start vertex is put into the queue. Then at every step, the vertex with the lowest cost-to-come is extracted from the priority queue and marked as visited. All of its unvisited neighbors are then put into the priority queue and their cost-to-comes are calculated. This process is repeated until the destination vertex is marked as visited, in which case all vertices that have a lower cost-to-come are already visited,

thus optimality is guaranteed. Any vertex that has a higher cost-to-come will not be visited, thus the algorithm does not necessarily traverse the whole graph.

The A* algorithm was later proposed as an improvement over the Dijkstra's algorithm for higher efficiency in finding the optimal path. A* also traverses the graph from a starting vertex and then gradually expands the visited vertices set until the destination is being visited. But instead of ordering vertices by their cost-to-come, A* orders vertices by a combination of cost-to-come and heuristic cost-to-go (an estimation of the cost from the vertex to the destination). The inclusion of the heuristic cost-to-go gives the planner the extra information to evaluate the quality of the candidates for the next node to be visited. For Dijkstra's algorithm, the planner only considers how far a candidate is to the start; whereas for A*, the planner also considers how close a candidate is to the destination. This will allow the planner to prune bad candidates that are only close to the start but far to the goal, thus fewer vertices are likely to be expanded when the goal is visited. Figure 2.1 illustrates a comparison of the node expansions between A* and the Dijkstra's algorithm, where A* uses Manhattan distance to evaluate heuristic cost-to-go. In this instance, A* is able to find the optimal path by visiting far fewer vertices than the Dijkstra's algorithm.

Figure 2.1: A performance comparison of the A* algorithm (left) and the Dijkstra's algorithm (right) on a random world. The free space is marked in grey; obstacles are marked in black; the start vertex is marked in green; the destination is marked in red; the optimal path is marked in yellow; the visited vertices are marked in blue. The A* algorithm uses Manhattan distance to evaluate heuristic cost-to-go. This figure is generated by PathFinder [52].

It can be proved that if the heuristic being used is admissible (the true cost-to-go is never over-estimated), then the solution found is optimal. It has also been proved that the A* algo-

rithm is optimally efficient [20], which means any other algorithm that uses the same heuristic will expand at least as many vertices as A*. Due to its performance and accuracy, A* still enjoys widespread use today in robotics.

2.1.2 Path Planning in Partially-Known Environments

In real-world applications, the environment is often partially known or partially observable. Thus robot navigation is more difficult than that in a known environment. This is due to the uncertainty of the information in the environment and the robot sensor; or the environment itself can change during the robot navigation. In such scenarios, it is often more difficult to obtain an optimal solution. The robot has to evaluate the uncertainties to make more robust decisions, or frequently use local information to adjust its plan on the way.

Risk-Aware Path Planning

One way to tackle the uncertainties in the environment and the robot sensor is to compute risk-aware or risk-sensitive plans that are more likely to succeed, or can be bounded by a limited risk.

For example, Feyzabadi et al. [14] proposed an algorithmic framework that first uses a risk function to mask the vertices in the map, then a Constrained Markov Decision Processes (CMDP) will be constructed and solved to produce a path that has a bounded risk.

By taking robot localization uncertainty into consideration, some works [51], [45] etc. suggest to find and follow the path with the lowest accumulated uncertainty for robot navigation, i.e., execute the most reliable path to the goal. An illustration is demonstrated in Figure 2.2. The figure shows that though possibly longer than the shortest path, the most reliable path will have the highest likelihood of being successful. Whereas due to sensor limitations, the robot could get lost at some point when trying to execute the shortest path, which could cause drastic failures in practice.

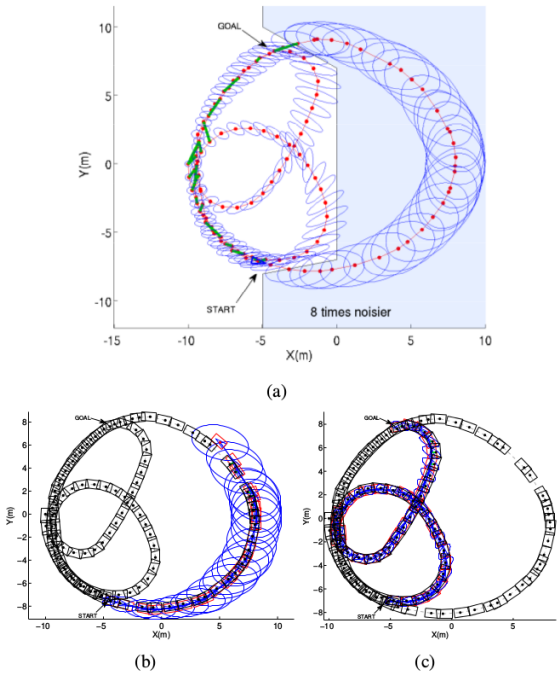


Figure 2.2: Comparison of the shortest path and the most reliable path between two vertices in a Pose SLAM graph. (a) A map generated by Pose SLAM [22], with red dots and lines representing the trajectory and ellipses representing pose uncertainties. (b) An attempt to traverse the shortest path to the goal, but at one point the sensor registration fails and robot gets lost. (c) A successful attempt of traversing the least uncertain path to the goal. The least uncertain path has the highest likelihood of being successful. Figure reproduced from [51].

Incremental Planning

Another way to adapt to the uncertain environment is to use incremental planning techniques that first follow an initial plan, and then replan when the previous plan is no longer valid. Many incremental search algorithms reuse information from previous searches to speed up the current search and can solve search problems potentially much faster than solving them repeatedly from scratch.

Lifelong Planning A* (LPA*) [25], D* [49] and its variants Focused D* [50], D* Lite [24] are among the most popular incremental search techniques. LPA* uses the A* algorithm to generate an initial path to follow. But instead of restarting the A* algorithm from scratch when inconsistent observations (obstacles, change in edge weights etc.) are encountered, LPA* will first start with the inconsistent vertex and identify all of its connected inconsistent neighbors. LPA* will then only resolve the inconsistent vertices to generate new plans. A vertex is in-

consistent if its cost-to-come is different from the best value that could be calculated from its immediate neighbors.

Like the A* algorithm, LPA* uses heuristic to prune the search space and priority queues are often used in implementation for higher efficiency. But LPA* is more efficient than running the A* algorithm multiple times by reusing the unchanged information from history.

However, for optimal pathfinding problems, replanning often indicates a better solution exists. The quality of the plan can be improved if more information can be incorporated into the planning step.

2.1.3 Offline versus Online Path Planning

Offline planners compute the entire path or trajectory to the destination before the robot starts to move. For example, A* will only return a solution when the optimal path is found. However, in practice, many robot systems will only allow a limited amount of computational time to find a solution. Thus it is often desired for the system to have an online planner that can come up with a feasible solution quickly, even if the solution is not optimal. The robot will take some actions first, and then the system can later improve upon the sub-optimal solution when more computational time is available.

Anytime Repairing A* (ARA*) [28] is an extension of the A* algorithm that achieves the online performance. ARA* is inspired by the fact that A*'s optimality can be sacrificed to obtain quicker execution time by inflating the heuristic. ARA* first finds a possibly highly sub-optimal solution very fast by using an inflated heuristic, then iteratively reduces the degree of inflation until an optimal solution is found, or the allocated time expires. Like LPA*, ARA* is able to reuse effort from previous searches to improve efficiency.

Sampling methods are also popular choices for online (or anytime) path planning. Some typical examples are RRT* [23], and its improved variants such as Informed RRT* [19]. RRT* is an improvement on Rapidly-exploring Random Trees (RRT) [27]. RRT* solves the path planning problem by incrementally growing a tree through collision-free space. At every step, a random sample within a parametric distance is drawn from the search space to perform tree expansion, which incrementally grows the tree towards unexplored regions. Similar to the

concept of Dijkstra's algorithm, the new vertex is connected to a nearby vertex that minimizes the new vertex's cost-to-come. It then finds any nearby vertices whose cost-to-come can be improved by connecting to the new vertex. These vertices will be rewired as descendants of the new vertex. A feasible solution is found once the goal is added to the tree. The solution can then be improved over time (and eventually becomes optimal asymptotically) through repeated tree expansions and rewiring.

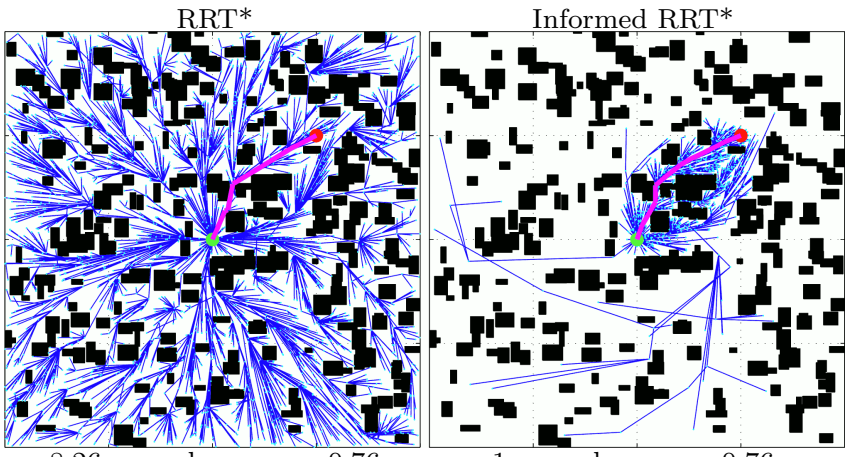


Figure 2.3: A performance comparison of the RRT* algorithm and Informed RRT* algorithm on a random world. The free space is marked in white; obstacles are marked in black; the start vertex is marked in green; the destination is marked in red; the expansion tree is marked in blue; the current best path is marked in pink. Informed RRT* only spent 1 second to find an equivalent-cost solution that RRT* spent 8.26 seconds to find. Figure reproduced from [19].

Informed RRT* is a further improvement on RRT* that, similar to A* algorithm, uses admissible heuristics on the upper bound to guide new vertex sampling once a feasible solution is found. Informed RRT* uses heuristics to avoids regions that will not improve the existing solution, and thus can improve the solution at a much faster speed than RRT*. Figure 2.3 shows a comparison of the RRT* and Informed RRT* algorithm, in which Informed RRT* only spent 1 second to find an equivalent-cost solution that RRT* spent 8.26 seconds to find.

2.2 The Canadian Traveler Problem (CTP)

All the algorithms being discussed in the previous section are designed to find a feasible or optimal path to the destination if one exists. However, for partially observable environments where replanning is often necessary, a more general approach is to compute a policy (or strategy) that tells the agent what path to take in every possible realization of the environment. For example, as in the Canadian Traveler Problem (CTP).

The Canadian Traveler Problem was first proposed by Papadimitriou et al. [39]. The name of the problem originates from the difficulties that Canadian drivers have in winter: heavy snow can randomly block some routes in a path network. Most related studies focus on the stochastic version of the CTP in which for a given graph, some edges are associated with an independent probability of being blocked; these edges are called *stochastic* edges. When the agent is at an endpoint of a stochastic edge, it has the option to disambiguate its state (i.e., observing if the edge is traversable or not). The true traversability status of each edge is static and will not change during the route traversal. The goal of solving the CTP is to find the optimal policy to traverse from a starting position to the goal with the minimum expected cost. The optimal policy is different from the shortest path: for a given instance of the problem, there are a number of possible realizations of how the actual graph may look, a policy describes a deterministic walk of the graph for every realization. A formal definition of the stochastic version of the CTP is given in Definition 2.1.

Definition 2.1 ((Stochastic) Canadian Traveler Problem)**.** *The stochastic CTP can be described using a tuple* $\tau =< G = (V, E), C, P, s, t >$ *where:*

- $G = (V, E)$ *is the given graph with* V *as the set of vertices and* E *as the set of undirected edges connecting the vertices,*
- $C : e \in E \rightarrow \mathbb{R}$ *is the function assigning edge costs,*
- $P : e \in E \rightarrow [0, 1]$ *is the function assigning edge blocking probabilities. An edge with 0 blocking probability is called a* ***deterministic*** *edge, an edge with 1 blocking probability will be removed from the graph, otherwise the edge is called a* ***stochastic*** *edge,*
- $s \in V$ *is the starting vertex of the agent,*

- $t \in V$ *is the destination vertex in* G*,*
- *The goal is to find the minimum-expected-cost policy* $\phi^* = \text{argmin}_{\phi \in \Phi}\{\mathbb{E}(\sigma(\phi))\}$*, where* Φ *is the set denoting all possible policies,* $\sigma(\phi)$ *returns the random variable representing the cost of policy* ϕ*.*

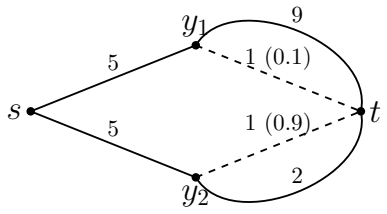


Figure 2.4: A simple CTP instance. Dashed lines are stochastic edges. Solid lines are deterministic edges. The cost of each edge is given above the edge. The blocking probabilities of stochastic edges are shown in parentheses. The goal of solving the CTP is to find the minimum-expected-cost policy to traverse from s to t.

As an example, Figure 2.4 illustrates a simple CTP instance with two stochastic edges. In the figure, solid lines denote deterministic edges and dashed ones denote stochastic edges. Numbers above each edge denote the edge's cost and blocking probabilities of stochastic edges are shown in parentheses next to the edge cost.

CTPs can be converted to Markov Decision Processes (MDPs) and Deterministic Partially Observable Markov Decision Processes (Deterministic POMDPs) [2]. However, Papadimitriou et al. [39] showed that the problem is #P-hard and Fried et al. [17] further proved that CTP is PSPACE-complete, suggesting that the MDP and POMDP formulations will have exponentially many states. Thus, although CTPs can be solved optimally in MDP and POMDP formulations in theory, most studies use approximate methods [29], sampling methods [12], or graph search with heuristic pruning [9], [2] to find sub-optimal policies.

The AO* algorithm [44] and some of its more sophisticated variants [2], [13], [1] have also been used to solve the CTP. AO* uses branch-and-bound with heuristics to prune the search space to find an optimal solution as efficiently as possible, making it a typically efficient option (though the worst-case complexity does not change) to solve the CTP.

There are many variants of the Canadian Traveler Problem that offer extensions in different aspects:

- In the *k-Canadian Traveler Problem*, an upper bound k is imposed on the number of blocked edges in the graph. A randomized online algorithm is proposed in [5], which is even optimal for some special configurations of the graph.

- In the *Recoverable Canadian Traveler Problem*, each blocked stochastic path is associated with a recovery time to reopen. Noy et al. [3] proposed a polynomial-time travel strategy that guarantees the smallest worst-case travel time, in the case where an upper bound on the number of blockages is known in advance, and the recovery times are not long relative to the travel times.

- *Canadian Traveler Problem with Remote Sensing* is a generalization to the CTP where remote sensing actions are allowed at some cost towards a distant location. The goal of solving the problem is thus minimizing the sum of the travel cost and the sensing cost. Bnaya et al. [9] proposed a framework that utilizes heuristics to determine when and where to sense the environment, and is optimal in some special cases.

This thesis is interested in solving the Canadian Traveler Problem by finding solutions that have a good balance of both policy cost expectation and variability. To the best of our knowledge, there are no CTP frameworks other than what is being proposed in this thesis, that explicitly evaluate policy variability and take that into account in policy computation.

2.3 Visual Teach and Repeat

The topics of this thesis were initially inspired by the path planning challenges that the VT&R framework faces: the VT&R algorithm builds a network of reusable paths [47] [48] for autonomous repeats of the previously demonstrated paths, but some segment of the paths may not always be traversable during the actual traversal due to various reasons such as prone to random obstacles or prone to localization failures. This section will introduce the VT&R algorithm and the map structure being used.

Furgale and Barfoot [18] proposed the Visual Teach and Repeat (VT&R) algorithm that enables mobile robots to autonomously repeat previously demonstrated path by only using visual data from a stereo camera. The VT&R algorithm consists of two phases. During the teach

pass, a human will drive the robot to demonstrate desired routes, while the mapping system will build a local topologically connected map. Then during the repeat pass, the robot will localize against the previously built map to repeat the demonstrated route autonomously. The VT&R algorithm was tested in different settings, such as urban, indoor-to-outdoor, vegetation-free terrain etc. Lots of attempts have been made to improve and extend the VT&R algorithm. McManus et al. [33] used a laser scanner and lidar intensity images to avoid lighting effects on the appearance change. Paton et al. [40] transformed images collected from a three-channel stereo camera into color-constant images that are resistant to changes in outdoor lighting conditions and improved the autonomy repeat rate to be above 99.9%. Berczi et al. [8] extended the VT&R algorithm to also have the ability to assess the terrain and predict the traversibility of a new terrain by comparing against past experiences. The initial VT&R algorithm only localizes against one previous experience (the teach pass), which is vulnerable to environmental appearance changes over time. Recently, researchers have developed the Multi-Experience Localization (MEL) algorithm [41] that addresses this limitation of single experience VT&R with a overarching enhancement: the ability to continuously estimate, with uncertainty, the localization between the live experience and a privileged (manual) experience, by using several other intermediate experiences simultaneously to bridge the appearance gap, while only one teach pass is required.

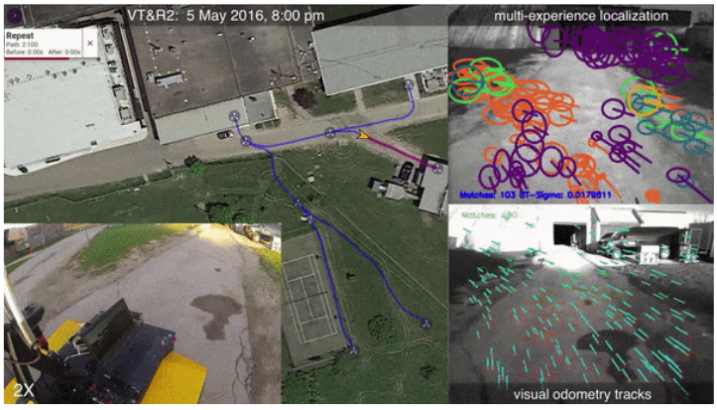


Figure 2.5: Live demonstration of the VT&R algorithm. The left panel shows the privileged (teach) paths layered on top of the satellite map. The bottom-left corner shows the image of the robot. The bottom-right corner shows the features extracted from the live image as well as the visual odometry track. The top-right corner shows the matched features from the live image to previous experiences over the path, with different colors indicating feature matches from different experiences.

Figure 2.5 shows an example from a live run of the VT&R algorithm with MEL. The left panel shows the privileged (teach) paths layered on top of the satellite map. The bottom-left corner shows the image of the robot. The bottom-right corner shows the features extracted from the live image as well as the visual odometry track. The top-right corner shows the matched features from the live image to previous experiences over the path, with different colors indicating feature matches from different experiences. The MEL algorithm uses a Spatio-Temporal Pose Graph (STPG) to support the multi-experience bridging. An example of how MEL using STPG for multi-experience bridging is shown in Figure 2.6.

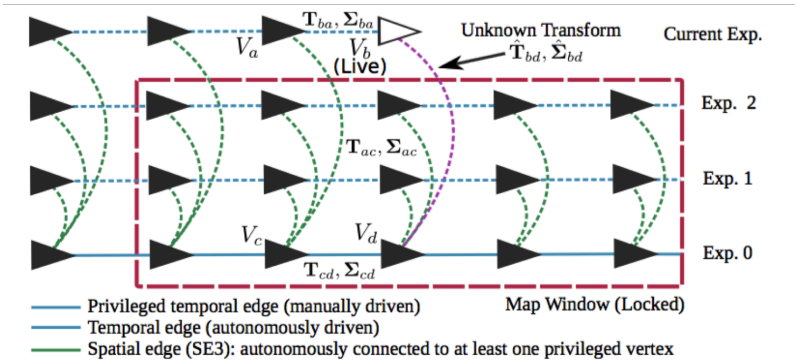


Figure 2.6: Overview of the multi-experience localization (MEL) problem and the spatial-temporal pose graph (STPG) data structure. The aim is to estimate the unknown transform and uncertainty, $\{\hat{\mathbf{T}}_{bd}, \hat{\boldsymbol{\Sigma}}_{bd}\}$ (dashed, purple line), between the live vertex, V_b, and the target vertex, V_d, in the privileged path (solid blue line). This is achieved by matching all landmarks in V_b to landmarks observed in the map window (dashed, red rectangle), transformed into the coordinate frame of V_d. This setup allows for outlier rejection and a simple optimization of $\{\hat{\mathbf{T}}_{bd}, \hat{\boldsymbol{\Sigma}}_{bd}\}$ against a map of locked landmarks with uncertainty.

The STPG data structure is an undirected graph, $G = \{V, E_s, E_t\}$, where V is a set of vertices, E_t is a set of temporal edges, and E_s is a set of spatial edges. Vertices, each with an associated reference frame, F, store raw sensor observations and triangulated 3D landmarks with associated covariances and descriptors. An edge in the graph links vertices metrically with a relative $SE(3)$ transformation with uncertainty. Temporal edges (blue lines) link vertices that are temporally adjacent, while spatial edges (green lines) link vertices that are temporally distant yet spatially close. Temporal edges can furthermore be denoted as privileged if they were collected while the robot was being taught a route or autonomous if the robot was repeating a route; this distinction is illustrated in Figure 2.6 as solid and dashed lines, respectively. An

experience is a collection of vertices linked by temporal edges.

The STPG thus naturally forms a network of reusable paths [47] [48] with rich historical data, which includes robot poses with uncertainties, landmarks observed, vehicle velocities, obstacle detection results, localization results, and manual interventions etc., all associated with timestamps. Thus a variety of options are available for generating statistics to predict the success rate of a proposing route. For example, the temporal difference between a previous experience and the current experience can act as an indicator of environmental change; the localization result (uncertainty, accuracy etc.) along an edge tells information about the probability of successful localization; terrain assessment result and manual intervention history can also indicate the traversability of the paths.

With each path being assigned a probability of successful traversal, this is exactly the stochastic Canadian Traveler Problem that was discussed in the previous section. However, as illustrated by the motivational example in Chapter 1.1, two policies that have similar expected costs can have quite different variability around the mean. In this thesis, we are more interested in finding robust policies that, while keeping the expected cost low, are also stable in terms of possible policy cost variability.

2.4 Discussion

In this chapter, we presented some typical planning algorithms for pathfinding in mobile robotics. Depending on different contexts of the problems, each algorithm can have its own strength and weakness.

For fully observable environments where a complete knowledge of the map is given, offline-optimal and search-based algorithms such as A* are often used to find an optimal path before the robot taking any actions. Whereas for partially observable environments, it is often difficult for the agent to find the optimal path due to the missing of information. The agent can only make a best guess based on its current knowledge and adjusts the plan when new information is available. Incremental search algorithms such as LPA* can then be applied to make replanning efficient by utilizing the information from previous searches.

In some settings of partially observable environments, though the observations can only

be made upon robot execution, the agent can still have a knowledge about all the possible observation outcomes. For example, the traversability of a path can only be traversable or untraversable. Thus though finding an optimal path may be hard, the agent might still be able to compute an optimal policy that tells the robot what is the best action to take in any possible situation. The Canadian Traveler Problem is one such problem that aims to find the traversal policy with the minimum expected cost. Existing graph search algorithms such as AO* may be used to compute the optimal policy.

In practice, many mobile robot systems only allow a limited amount of time to find a solution. To achieve the real-time performance, incremental search methods such as ARA* and sampling methods such as RRT* will first sacrifice optimality to find a feasible solution quickly, and then gradually adjust its plan towards the optimal solution when more time is available.

There are some other related works that also try to seek for the optimal policy in partially observable environments. For example, the PAO* algorithm for planning with hidden state introduces the idea of pinch points [13]. It identifies areas of uncertain traversability that may have a significant impact on reaching the goal, and then tries to compute an optimal policy with the minimum expected cost while considering those pinch points. However, not much literature considers the possible cost variation of following the same policy multiple times in stochastic environments. In this thesis, we are interested in solving the Canadian Traveler Problem by finding policies that have low expected costs as well as low variations on the possible costs. We attempt to find offline solutions that guarantee optimality, as well as online solutions that can satisfy the needs of real-time operations.

Chapter 3

Problem Formulation

This thesis is devoted to finding robust, or risk-sensitive policies for the Canadian Traveler Problem, such that both policy cost expectation and variability will be considered during policy selection. In the previous chapter, we have introduced related literature on the CTP and some related planning techniques. In this chapter, in order to formalize the Robust Canadian Traveler Problem, we will first introduce the AND/OR tree data structure that can be used to represent (and visualize) the CTP problem as well as its policies. Then different approaches to quantify policy risks will be discussed from the perspective of viewing the policy cost as a random variable that follows a discrete distribution. Finally, by choosing the exponential risk as the policy risk measure, a formal definition of the Robust Canadian Traveler Problem will be proposed. An MDP definition of the RCTP will also be provided.

3.1 Representing CTPs as AND/OR Trees

The AND/OR tree is a data structure often used for problems that can be decomposed into subproblems, and if an action can lead to more than one possible state. The AND/OR tree contains two types of nodes: AND nodes (chance nodes) that combine solutions from all of their children; and OR nodes (choice nodes) that choose a single solution from their children[37].

Figure 3.1 shows an example AND/OR tree, in which the root state (an OR node) has two edges leaving it by taking different actions. Every action leads to an AND node with two successor OR nodes, each representing a possible outcome by taking the corresponding action.

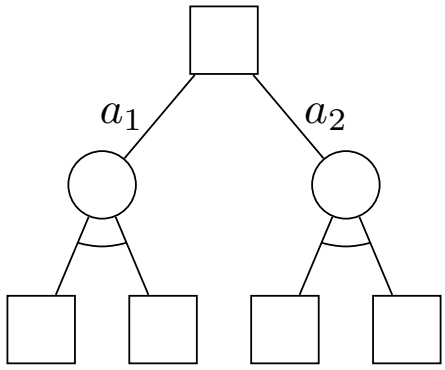


Figure 3.1: An example AND/OR tree: by convention, OR nodes (choice nodes) are represented as squares, AND nodes (chance nodes) are represented as circles. In the example, the root OR has two edges leaving it, one for action a_1 and one for action a_2. Each action leads to an AND node with two successor OR nodes, each representing a possible outcome of taking the corresponding action.

In AND/OR tree search, a solution is a sub-tree defined as follows:

- the initial state (root node in the full AND/OR tree) belongs to the solution sub-tree,
- every OR node in the solution tree selects **exactly one** edge (choose one action) leaving it from the corresponding connected edges in the full tree,
- every AND node in the solution tree copies **all of the children** (possible outcomes from a chosen action) from the corresponding AND node in the full tree,
- every directed path in the solution tree terminates at a goal state.

To find the minimum-cost solution sub-tree, the cost of each sub-tree associated with a node must be evaluated, which can be calculated recursively by the function $f^*(v)$:

- if v is a goal state, $f^*(v) = 0$,
- if v is a terminal state other than the goal state, $f^*(v) = \infty$,
- if v is an OR node, $f^*(v) = \min_{v' \in S(v)}[c(v, v') + f^*(v')]$,
- if v is an AND node, $f^*(v) = \sum_{v' \in S(v)}[p(v, v') * (c(v, v') + f^*(v'))]$,

where $f^*(v)$ denotes the cost of an optimal solution for state vertex v. $S(v)$ represents the set of children vertices of v in the tree, $c(v, v')$ is the cost of the edge connecting v and v', and $p(v, v')$ is the probability that taking the previously chosen action in state v results in state v'.

As suggested by [2] and [13], given an instance of the CTP, an AND/OR tree representation can be constructed as follows:

1. Each node in the AND/OR tree is a state $\delta = (v, L)$ where $v \in V$ is the position of the agent. $L = \{\text{“}A\text{”}, \text{“}T\text{”}, \text{“}U\text{”}\}^k$ is an information vector representing the agent's knowledge about the k stochastic edges in the graph, in which “A” stands for ambiguous, “T” stands for traversable and “U” stands for untraversable. The root of the AND/OR tree is an OR node with state $(s, [\text{“}A\text{”}, ..., \text{“}A\text{”}])$.

2. Children of OR nodes are always AND nodes. These AND nodes are either:

   (a) termination states (agent at goal t, or no solution)

   (b) states $\delta' = (v', L')$ in which v' is an end vertex of an undisambiguated stochastic edge and is reachable via a deterministic path from its parent OR node, L' is copied from its parent. If v' connects to more than one undisambiguated edges, copies will be made for each of them.

   The edge cost of an AND node to its parent OR node is the shortest deterministic distance connecting the two states, assuming all undisambiguated stochastic edges are blocked. These AND nodes represent the available actions that the parent OR node can take (go to an endpoint of a stochastic edge to perform disambiguation, or go to a terminal state).

3. Children of AND nodes are always OR nodes, corresponding to the possible disambiguation results from its parent (traversable or untraversable). The edge cost from an OR node to its parent AND node is the cost of observation (we assume the observation cost is 0 in this thesis).

3.1.1 A Simple Example

Figure 2.4 illustrates a simple CTP instance with two stochastic edges. The complete AND/OR tree corresponding to this instance is shown in Figure 3.2, in which the optimal policy with the

least expected cost is colored in green. In the figure, OR nodes (corresponding to the actions the agent can take) are depicted by squares and AND nodes (corresponding to probabilistic outcomes of the agent's actions) are depicted by circles. The letters "U" and "T" next to arcs emanating from AND nodes denote untraversable and traversable disambiguation outcomes, respectively. The fractional number below each AND node is the blocking probability of the corresponding stochastic edge. The numbers next to arcs emanating from OR nodes denote the action costs. The expected cost of the optimal policy is thus $f(s,(A,A)) = 5 + 0.1 \times 9 + 0.9 \times 1 = 6.8$, with a best-case cost of 6 and a worst-case cost of 14.

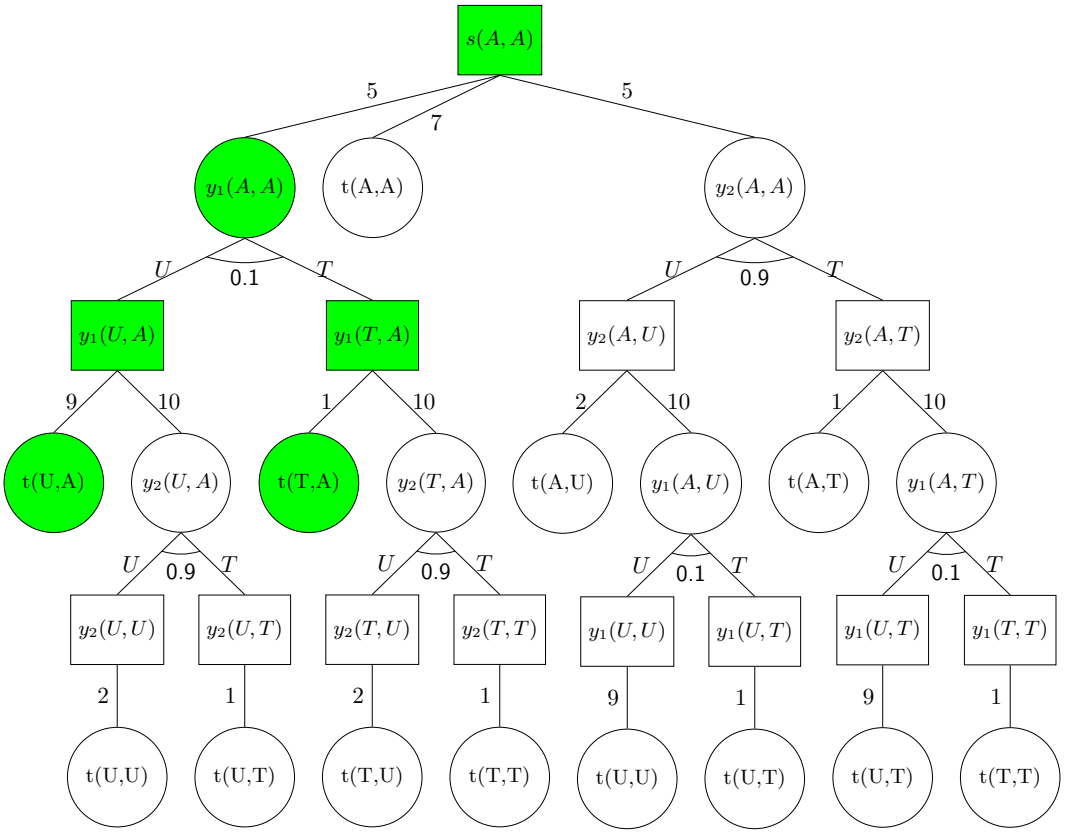


Figure 3.2: The complete AO Tree corresponding to the CTP Instance in Figure 2.4. OR nodes are depicted by squares and AND nodes by circles. Numbers below AND nodes are blocking probabilities of the corresponding edges to be disambiguated. Numbers beside edges are action costs. The letters "U" and "T" next to arcs emanating from AND nodes denote untraversable and traversable disambiguation outcomes, respectively. The policy tree with the lowest expected cost is in green.

The benefit of representing CTPs as AND/OR trees is that we can use existing graph search algorithms such as AO* [44] to search for the optimal policy efficiently. AO* is a heuristic

search algorithm that finds an optimal policy in a given state-space search problem that can be represented using an AND/OR tree. AO* uses heuristics to prune the search space and will not require a complete AND/OR tree representation of the problem as it can be implicitly suggested by the start state and the successor function. AO* builds a solution tree that initially only contains the start state. The solution tree is grown until a complete policy is found. The tree expansion is done by repeatedly using a heuristic to select the most promising path to expand, and then back-propagate to update costs. If the heuristic is admissible (never over-estimates the actual costs), then the solution found by AO* is optimal. More details about the AO* algorithm will be discussed in Chapter 3.

3.2 Quantifying Policy Risk

Our motivation to further extend the CTP to the Robust Canadian Traveler Problem comes from the observation that in some CTP instances, there exist policies that have very close or even equivalent expected costs to the optimal policy, but the uncertainties over the policies' costs can vary significantly. For example, the marked policy in Figure 3.2 has the minimum expected cost of 6.8, but it also has a large worst case of 14. However, consider another policy as shown in Figure 3.3, It first disambiguates the stochastic edge at y_2, then traverses to t either with a cost of 1 or 2 depending on the disambiguation result. The expected cost of this policy is $5+0.9\times 2+0.1\times 1=6.9$, which is slightly higher than the optimal policy. But the worst-case cost is only 7, which is much lower. For risk sensitive tasks (for example, when the robot has a limited remaining battery), we often want to choose such risk-averse policies over the optimal policy. To find such risk-averse policies, we first need to choose our tool to quantify policy risks.

For a CTP instance with a policy represented as an AND/OR tree, each traversal of the policy will reach one of the leaves with some probability. Thus we can view the cost of traversing a policy as a discrete distribution, with each leaf in the tree as a possible outcome. There are numerous ways to measure risk here, for example by the worst-case cost, the range of the distribution, the number of categories, etc.

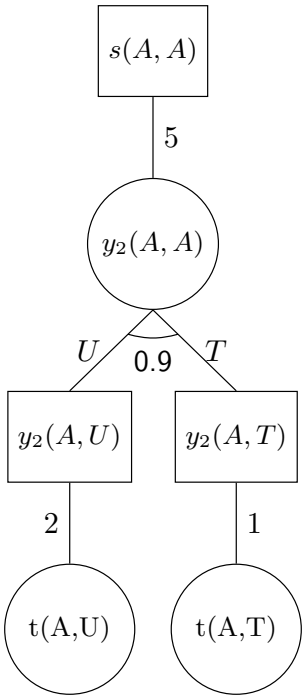


Figure 3.3: A risk-adverse policy for the CTP Instance in Figure 2.4. OR nodes are depicted by squares and AND nodes by circles. Numbers below AND nodes are blocking probabilities of the corresponding edges to be disambiguated. Numbers beside edges are action costs. The letters "U" and "T" next to arcs emanating from AND nodes denote untraversable and traversable disambiguation outcomes, respectively. This policy has an expected cost of 6.9, a best-case cost of 6 and a worst-case cost of 7.

3.2.1 Quantifying Policy Risk by Variance

One very popular metric to measure risk is to use variance. By viewing the policy traversal cost as a random variable following a discrete distribution, the variance of a policy cost can be calculated as $var(\pi) = \mathbb{E}(\pi^2) - \mathbb{E}(\pi)^2 = \sum p_i c_i^2 - (\sum p_i c_i)^2$, where p_i, c_i are the associated probability and cost of each leaf i. For example, variance of the marked optimal policy in Figure 2.4 is $[(5+9)^2 \times 0.1 + (5+1)^2 \times 0.9] - [(5+9) \times 0.1 + (5+1) \times 0.9]^2 = 5.76$, the variance of the risk-adverse policy in Figure 3.3 is $[(5+1)^2 \times 0.1 + (5+2)^2 \times 0.9] - [(5+1) \times 0.1 + (5+2) \times 0.9]^2 = 0.09$. It has been demonstrated in the literature that it is possible to solve the mean-variance adjusted MDP [31], but there are some major drawbacks:

- First of all, using variance as part of the evaluation criteria is not friendly to heuristic graph search algorithms such as AO*. AO* incrementally build its search tree through

expansions from the tree that just contains the root node. It relies on good-quality heuristic to prune the search space and relies on the admissibility of a heuristic to ensure optimality; that is, the heuristic value of a node during expansion should never exceed its optimal value. However, it is not trivial to find a good admissible heuristic for partial policy variance except that we know it is non-negative. Without a good heuristic, the search speed of AO* algorithm can decrease significantly as not much pruning can be done.

- Secondly, similar to mean-variance optimization in MDPs, it can produce counterintuitive policies due to the violation of Bellman's principle of optimality [32]. In order to keep the total variance small, a decision maker receiving small rewards at the early stages may try to avoid large rewards at later stages. Similarly, if the untraversable branch of an AND node is estimated to have a large cost, then in order to keep the policy variance small, the returned policy could prefer a large-cost sub-tree over low-cost subtrees in the traversable branch. Then during the actual course of traversal, even if the agent observes the stochastic edge to be traversable, it may not take the lowest-cost action based on the traversable observation, because what the agent really wants is to keep the total variance small if the next time this stochastic edge become untraversable, which will yield a large cost.

- Thirdly, variance penalizes deviation below the mean as much as above the mean, while deviation below the mean is actually desired in terms of the policy cost.

3.2.2 Quantifying Policy Risk by Exponential Risk Measure

To tackle the above-mentioned problems, we propose to use the exponential risk [21] as our risk measure.

Definition 3.1 (Exponential Risk)**.** *Given a random variable π and a weight parameter $w > 0$, the exponential risk is given by the function*

$$\gamma(\pi, w) = \frac{1}{w} \ln(\mathbb{E}(\exp(w\pi))).$$

A mathematical definition of the exponential risk measure is shown in Definition 3.1. It is widely used in the financial industry to make risk-aware investment decisions. For example, for pricing financial securities in incomplete markets [36], and pricing insurance products in dynamic markets [54]. The exponential risk measure has some nice properties:

- Approximate mean-variance adjustment: by taking a second-order Taylor expansion on the formula, we can see that

$$\begin{aligned}
\gamma(\pi, w) &= \frac{1}{w}\ln\left(\mathbb{E}(\exp(w\pi))\right) \\
&= \frac{1}{w}\ln\left(1 + w\mathbb{E}(\pi) + \frac{w^2}{2}\mathbb{E}(\pi^2) + \cdots\right) \\
&= \frac{1}{w}\left(w\mathbb{E}(\pi) + \frac{w^2}{2}\mathbb{E}(\pi^2) - \frac{1}{2}\left(w\mathbb{E}(\pi) + \frac{w^2}{2}\mathbb{E}(\pi^2)\right)^2 + \cdots\right) \\
&\approx \mathbb{E}(\pi) + \frac{w}{2}\text{var}(\pi)
\end{aligned}$$

  so minimizing exponential risk is approximately minimizing mean plus variance with weight $w/2$.

- Monotonicity: if $a \leq b$, then $\gamma(a, w) \leq \gamma(b, w)$ for constant a, b.

- Equivariant to translation: $\gamma(a + \pi, w) = a + \gamma(\pi, w)$ for constant a:

$$\begin{aligned}
\gamma(a + \pi, w) &= \frac{1}{w}\ln\left(\mathbb{E}(\exp(w(a + \pi)))\right) \\
&= \frac{1}{w}\ln\left(\mathbb{E}(\exp(aw + w\pi))\right) \\
&= \frac{1}{w}\ln\left(\exp(aw) \times \mathbb{E}(\exp(w\pi))\right) \\
&= \frac{1}{w}\ln\left(\exp(aw)\right) + \frac{1}{w}\ln\left(\exp(w\pi)\right) \\
&= a + \frac{1}{w}\ln\left(\mathbb{E}(\exp(w\pi))\right) \\
&= a + \gamma(\pi, w)
\end{aligned}$$

- It penalizes deviations above the mean more than deviations below, due to the asymmetric shape of the exponential curve. Though it still penalizes improved paths below the

expected cost, any solver to the CTP actually already finds the solution with the minimum expected cost. Thus putting more penalization on deviation above the mean will make the solver to sacrifice as little mean as possible to avoid high-cost paths.

For example, with $w = 2$, the exponential risk of the marked optimal policy in Figure 2.4 can be calculated as $\frac{1}{2}\ln\{\exp[(5+9)\times 2]\times 0.1+\exp[(5+1)\times 2]\times 0.9\} = 12.85$, the exponential risk of the risk-adverse policy in Figure 3.3 can be calculated as $\frac{1}{2}\ln\{\exp[(5+2)\times 2]\times 0.9+\exp[(5+1)\times 2]\times 0.1\} = 6.95$. We can see that the risk-adverse policy has a lower risk than the optimal policy by the evaluation from the exponential risk measure. Moreover, with $w = 2$ the numbers obtained from the exponential risk evaluation are close to policy mean plus variance: the optimal policy in Figure 2.4 has a mean of 6.8 and a variance of 5.76, $6.8 + 5.76 = 12.56$ is close to its exponential risk 12.85, the risk-adverse policy in Figure 3.3 has a mean of 6.9 and a variance of 0.09, $6.9 + 0.09 = 6.99$ is close to its exponential risk 6.95.

3.3 The Robust Canadian Traveler Problem (RCTP)

Motivated by avoiding large extreme costs, we thus extend the CTP to what we call the Robust Canadian Traveler Problem (RCTP). In the RCTP, we replace the goal of the problem to searching for the policy with the minimum exponential risk instead of the minimum expected cost, as inspired by the properties of the exponential risk measure discussed in the previous section. A definition of the RCTP is given in Definition 3.2

Definition 3.2 (Robust Canadian Traveler Problem)**.** *The Robust Canadian Traveler Problem can be described using a tuple* $\tau =< G = (V, E), C, P, s, t, w >$ *where:*

- $G = (V, E)$ *is a graph with* V *as the set of vertices and* E *as the set of undirected edges,*
- $C : e \in E \to \mathbb{R}$ *is the function assigning edge costs,*
- $P : e \in E \to [0, 1]$ *is the function assigning edge blocking probabilities. An edge with 0 blocking probability is called a deterministic edge, an edge with 1 blocking probability will be removed from the graph, otherwise the edge is called a stochastic edge,*
- $s \in V$ *is the starting vertex of the agent,*

- *$t \in V$ is the destination vertex in G,*
- *w is the parameter that adjusts the weight of the exponential term in the exponential risk measure.*
- *The goal is to find the policy $\phi^* = \text{argmin}_{\phi \in \Phi}\{\frac{1}{w}\ln(\mathbb{E}(\exp(w \times \sigma(\phi))))\}$, where Φ is the set denoting all possible policies, $\sigma(\phi)$ returns the random variable representing the cost of policy ϕ.*

3.4 Assumptions

The following assumptions are made in our RCTP framework:

1. The blocking probabilities of the stochastic edges are independent from each other.
2. The cost of disambiguation (observing if a stochastic edge is traversable or not at its end vertices) is zero.
3. Once being disambiguated, the traversability status of the stochastic edges do not change during the course of traversal.

3.5 The MDP Formulation

As in CTP, the RCTP can also be formulated as an MDP described by the 5-tuple $< X, A, P_a, C_a, \gamma >$, where:

- X is a finite set of states. For RCTP, we can define the states the same way as we do for the AND/OR tree conversion. X is now the set of all possible (v, L) pairs where $v \in V$ denotes the position of the agent in the graph. $L = \{\text{“}A\text{”}, \text{“}T\text{”}, \text{“}U\text{”}\}^k$ is an information list maintaining the agent's knowledge about the traversability of the k stochastic edges in the graph.
- A is a finite set of actions, with A_δ being the finite set of actions available from state δ. For RCTP, A is now the set of all possible $< v, v', e >$ tuples, meaning to traverse from v to v', and then disambiguate the stochastic edge e at endpoint v'.

- $P_a(\delta, \delta')$ is the probability that taking action a in state δ will lead to state δ'. For RCTP, it is specified by the stochastic edge to be disambiguated.
- $C_a(\delta, \delta')$ is the immediate cost for transitioning from state δ to δ'. For RCTP, we let it be the shortest deterministic distance from δ to δ'.
- $\gamma \in [0, 1]$ is the discount factor, we simply let it be 1 for RCTP denoting no discount for future costs.

Value Iteration (VI) is one of the standard algorithms to solve MDPs by calculating the array V, such that $V(\delta)$ contains the estimated costs (on average) by following the optimal solution from state δ. For standard MDPs, V can be calculated recursively by:

$$V(\delta) = \min_{a \in A_\delta} \left\{ \sum_{\delta'} P_a(\delta, \delta')(R_a(\delta, \delta') + V(\delta')) \right\}.$$

Substituting in the exponential risk measure to minimize the exponential risk instead of the average, we have

$$V(\delta) = \min_{a \in A_\delta} \left\{ \frac{1}{w} \ln \sum_{\delta'} P_a(\delta, \delta') \exp(w(R_a(\delta, \delta') + V(\delta'))) \right\}.$$

This is actually one of the risk-sensitive MDPs that have been well-studied [26]. However, as the problem has an exponential state space of $O(|V|3^{|E|})$, it is not practical to use MDP with value iteration to solve large-scale RCTP instances. In the next chapter, we will demonstrate methods to adapt the AO* algorithm and use heuristics to speed up the policy computation.

3.6 Summary

In this chapter, we showed the approach to represent CTPs as AND/OR trees. The AND/OR tree can be used to give a nice intuitive visualization of the problem domain and policy structures. And more importantly, existing AND/OR graph search methods such as AO* can then be applied to find the optimal policy efficiently.

As illustrated by the AND/OR tree data structure, a CTP policy can be viewed as a discrete distribution with each leaf in the AND/OR tree as a possible outcome. The AND/OR tree also

gives access to compute the cost and probability of each category, which can then be utilized to compute different statistics of the policy that can be used to quantify policy risks. We explicitly discussed the drawbacks of using the most popular choice - variance, and then propose to use the exponential risk as a better choice to quantify policy risk.

We then give a formal definition of the Robust Canadian Traveler Problem, which aims to compute the policy with the minimum exponential risk. Finally, an MDP definition of the RCTP is also provided to give more insights into the complexity of the problem.

Chapter 4

Algorithms

This chapter describes the algorithms that we propose to solve the Robust Canadian Traveler Problem. We will first describe the AO* algorithm that can be directly used to solve classic CTPs represented by AND/OR trees. Then we will propose a modified version of the AO* algorithm, which we call Offline-RCTP-AO*, that can be used to solve RCTPs optimally offline. Finally, an online version of the algorithm, which we call Online-RCTP-AO*, will be introduced. The Online-RCTP-AO* algorithm provides approximate solutions to the Offline-RCTP-AO* algorithm, but with significantly less computational time.

4.1 The AO* Algorithm

In Chapter 3.1, we showed how to represent CTPs as AND/OR trees. In theory, the optimal policy to a problem that is represented as an AND/OR tree can be found by computing every node's minimum expected cost recursively from the bottom layer up to the root. However, CTPs have an exponential search space, which corresponds to an exponential number of nodes in the AND/OR tree representation. It is thus not practical to conduct such brute-force computations. Fortunately, we are only interested in the optimal cost of the root node, and we do not need to calculate every node's optimal cost to calculate this number. AO* [44] is a heuristic search algorithm that improves the brute force method by using heuristics to find the optimal solution while only visiting a small portion of the search space. Specifically, AO* calculates node cost in a top-down fashion and uses admissible lower bounds (lower bounds that are guar-

anteed not to overestimate the true cost of any node) to select the next layer of nodes to expand, such that only a small portion of the complete AND/OR tree is examined. A detailed description of the AO* algorithm is shown in Algorithm 1. We define the following notation: for a node (state) v, $h(v)$ returns v's heuristic cost-to-go, $f(v)$ is the estimated cost of the current best policy at v, $S(v)$ returns the successors (children) of v, $c(v, v')$ is the cost of going from state v to v'.

Algorithm 1 The AO* Algorithm

```
1:  function AO*(s, h(·), t)
2:      f(s) ← h(s); s.type = OR; T.root ← s
3:      while s.status ≠ SOLVED and f(s) ≠ inf do
4:          v ← SELECTNODE(T.root)
5:          EXPAND(v)
6:          for v' ∈ S(v) do
7:              f(v') = h(v')
8:              if v' ∈ t then v'.status = SOLVED
9:          BACKPROPAGATE(v, T)
10:     if T.root.val == inf then
11:         return NO SOLUTION
12:     return T
13:
14: function BACKPROPAGATE(v, T)
15:     while v ≠ T.root do
16:         if S(v) ≠ ∅ then
17:             if v.type == OR then
18:                 v' = argmin_{v'∈S(v)}(f(v') + c(v, v'))
19:                 f(v) = c(v, v') + f(v')
20:                 if v'.status == SOLVED then v.status = SOLVED
21:             if v.type == AND then
22:                 f(v) = Σ_{v'∈S(v)}{p(v, v') × [c(v, v') + f(v')]}
23:                 if v'.status == SOLVED ∀v' ∈ S(v) then
24:                 v.status = SOLVED
25:         v = v.parent
```

AO* builds a solution tree that initially only contains the root node (start state). The solution tree, which is a sub-tree of the complete AND/OR tree representing partial solutions of the optimal conditions, is gradually augmented until the root node is solved (complete solution found). The tree expansion is performed by two alternating steps, expansion and propagation:

- *expansion:* in this step, a non-terminal leaf node (a state representing neither goal state nor no-solution state) is chosen by traversing the most promising sub-tree from the root. Its successors are then added to the tree and assigned lower heuristic labels.

- *propagation:* in this step, the expanded node's cost will be recalculated using the heuristic costs of its newly-added successors. If the successors are terminal leaf nodes in the complete AND/OR tree, the costs are already optimal and the sub-problems associated with the nodes will be marked as solved, otherwise the costs are lower-bounded by the heuristic. Then every node's cost in the expanded-node-to-root path will be updated using the latest costs of its successors. The statuses (solved or not) of the sub-problems associated with the nodes are also updated accordingly during the propagation step: the sub-problem associated with an OR node is solved if its best child is solved; the sub-problem associated with an AND node is solved if both of its two children are solved.

Figure 4.1 shows the complete AND/OR tree associated with the instance in Figure 2.4. In the figure, the optimal policy sub-tree returned by AO* is colored in green. Extra expanded nodes are colored in yellow. Non-terminal leaves in the AO* expansion tree are colored in red. We can see that the AO* algorithm is able to find the optimal solution by only exploring a small portion of the complete AND/OR tree.

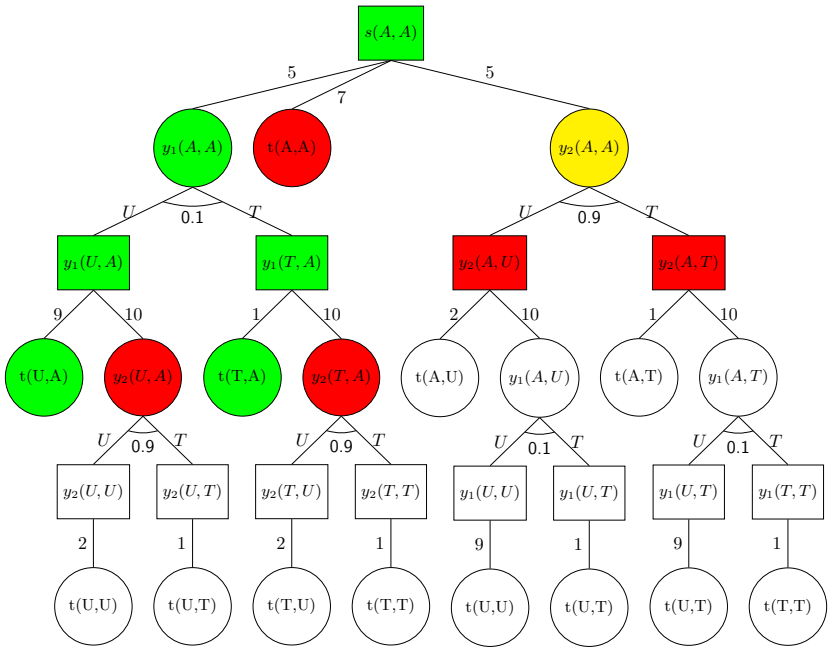


Figure 4.1: The complete AO Tree corresponding to the CTP Instance in Figure 2.4. OR nodes are depicted by squares and AND nodes by circles. Numbers below AND nodes are blocking probabilities of the corresponding edges to be disambiguated. Numbers beside edges are action costs. The policy tree returned by AO* is colored in green. Extra expanded nodes are colored in yellow. Non-terminal leaves in the AO* expansion tree are colored in red.

For CTPs, an intuitive admissible heuristic to calculate a node (state) v's cost-to-go is to calculate the cost of the naive shortest path from the agent position in that node (state) v to the goal. That is, during the shortest path calculation, all undisambiguated stochastic edges to the knowledge of v will be treated as traversable (the stochastic edges that are already disambiguated will still use their disambiguated true statuses). Since only in the best case scenario all the remaining undisambiguated stochastic edges would be revealed to be traversable, therefore this heuristic will never overestimate the true optimal cost of each node, and thus is admissible.

We also note that during the expansion phase, the best partial solution tree may have many non-terminal leaves. AO* works correctly no matter which of these leaves is chosen for expansion. However, the efficiency of AO* can be improved by using a good selection function to choose which non-terminal leaf of the best partial solution tree to expand next. Possibilities include always selecting the leaf with the least estimated cost, or selecting the leaf with the greatest probability of being reached.

4.1.1 Domain-Specific Pruning Techniques

Exploiting the properties of the CTP, we can introduce some domain-specific techniques into the AO* algorithm to make the search even faster. For example, Ferguson et al. [13] proposed to propagate a node cost not only upwards, but also to its neighbors in order to make the algorithm converge faster; Aksakalli [1] analyzed the upper bound and lower bound of each node's cost to prune unnecessary nodes; Aksakalli et al. [2] further introduced a caching mechanism to avoid duplicate expansions of different nodes with the same state. All of these techniques can be applied to our algorithmic framework to speed up the search as well. For the simplicity of demonstration, we excluded them from our algorithm description in Algorithm 1.

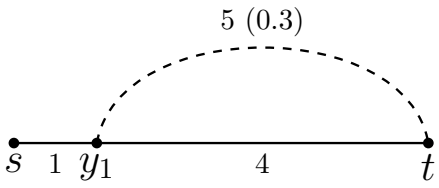
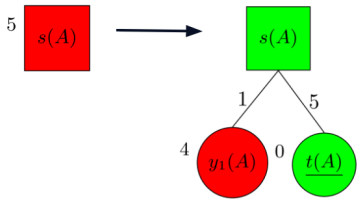
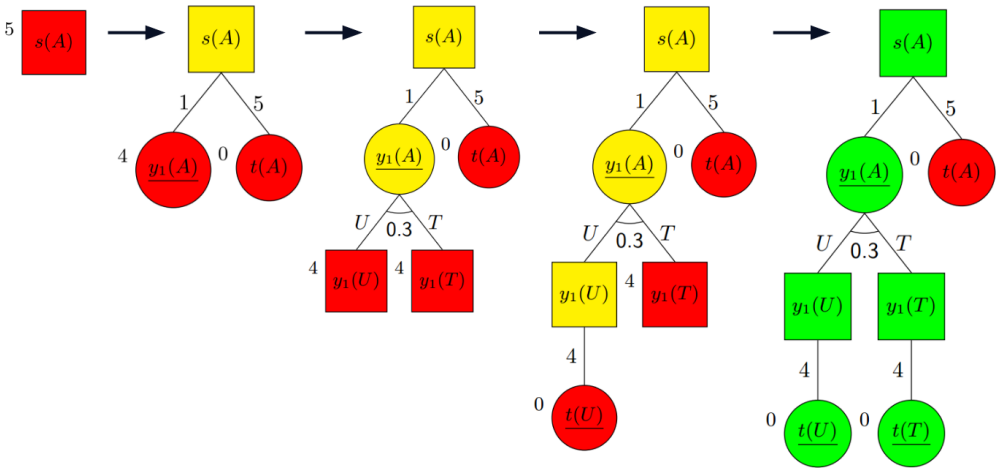


Figure 4.2: A CTP instance with one stochastic edge (denoted by dashed line) and two deterministic edges (denoted by solid lines). Cost of each edge is given above the edge. The blocking probabilities of stochastic edges are shown in parentheses.



(a) Policy A



(b) Policy B

Figure 4.3: Two possible ways that the AO* algorithm can take to compute the optimal policy on Figure 4.2, resulting in two policies with different structures but the same expected cost. OR nodes are depicted by squares and AND nodes by circles. Numbers below AND nodes are blocking probabilities of the corresponding edges to be disambiguated. Numbers beside edges are action costs. The policy tree returned by AO* is colored in green. Extra expanded nodes are colored in yellow. Non-terminal leaves in the AO* expansion tree are colored in red. The numbers next to the leaves are their heuristic cost-to-go calculated by the naive shortest distance from the node to the goal. The minimum child (combining cost-to-come and cost-to-go) that an OR node take is marked by the underline. For policy B, regardless of the disambiguation result at y_1, the agent will take the same path (with a cost of 4) to travel to t. Thus policy B is equivalent to policy A during traversal, but will take more steps and more time to compute.

However, besides the techniques in the above-mentioned works, there is another trick that we found can be utilized to speed up the algorithm by pruning the unnecessary disambiguations. Recall that in our AND-OR tree representation of the problem, for each OR node, every reachable end vertex of an undisambiguated stochastic edge will be added as an AND-node child to indicate a possible disambiguation. However, under this circumstance, disambiguating an edge does not necessarily mean trying to traverse that edge, which could cause wasting of

computational resources to expand trees for unnecessary disambiguations.

Now we consider a simple example in Figure 4.2 that only has one stochastic edge. Figure 4.3 shows two possible ways that the AO* algorithm can take to expand the solution tree - resulting in two policies that have different tree structures but are essentially the same in practice:

- Policy A: Travel from s to t directly with a cost of 5.
- Policy B: First travel from s to y_1, then disambiguate the stochastic edge y_1t (the upper edge), whether it is traversable or not, always take the deterministic edge y_1t (the lower edge) to t after the disambiguation. The total cost is also 5 since we assume disambiguation can be performed at zero cost.

Though policy A and B are essentially the same for the navigator, policy B would require more node expansions, more steps, and thus more time to compute as illustrated by Figure 4.3. Since the search space of CTPs grows exponentially, performing such unnecessary disambiguations could result in a massive waste of computation time, especially if it occurs at the early phase of tree expansions and if the implicit tree is deep. To prevent these unnecessary tree expansions, before we add in any new AND node to the expansion tree, we may first test if the stochastic edge to be disambiguated will actually be attempted or not.

Moreover, a node will never be expanded if its heuristic cost is greater than the cost of the risk-free deterministic path to the goal. To utilize these two properties to further speed up the AO* algorithm, we can modify the tree expansion step to only add "good" successors into the partial solution tree as follows:

- *selective expansion*: in this step, a non-terminal leaf node (a state representing neither goal state nor no-solution state) is chosen by traversing the most promising sub-tree from the root. Its successors are then added to the tree and assigned lower heuristic labels. Reject the successors whose node to be expanded is an OR node and any of the following condition is true:

    1. If the cost from the node-to-expand to the successor is larger than the risk-free cost (assuming all undisambiguated stochastic edges are untraversable) from the node-to-expand to goal;

2. If the successor (an AND node)'s two children have the same shortest risk-free path to the goal.

4.2 Adapting AO* Algorithm to Solve RCTP

This section will demonstrate how to adapt the AO* algorithm to solve RCTPs. The AO* algorithm will be adjusted to search for the minimum exponential risk with a given weight parameter w, instead of searching for the policy with the minimum expected cost. We provide two versions of the adapted AO* algorithm: an offline version that guarantees to find the optimal policy upon termination; and an online version with a limited search depth, but sequentially doing planning and execution to promptly interact with the environment.

4.2.1 The Offline-RCTP-AO* Algorithm

Benefiting from the properties of the exponential risk measure, the AO* algorithm can be easily adjusted to apply to RCTPs - we only need to change the cost estimation function for the back-propagation phase. We define the following notation: for a node (state) v, ϕ_v denotes the current best (partial) policy at v, $h(v)$ returns v's heuristic cost-to-go, $f(v)$ is the estimated cost (exponential risk) of the current best policy at v, $f^*(v)$ is the optimal cost of v, $S(v)$ returns the successors (children) of v, $c(v, v')$ is the cost of going from state v to v'. The adjusted algorithm is depicted in Algorithm 2. It also has two alternating steps, expansion and propagation:

- *expansion:* for the Offline-RCTP-AO* algorithm, the expansion step is still the same as the classic AO* algorithm, except that when selecting the next node to expand, the estimated exponential risk (updated in the propagation step) is used. Its successors are then added to the tree and assigned lower heuristic labels.

- *propagation:* the expanded node's cost (exponential risk) are now propagated recursively from bottom to root as follows:

  - if the node v is an OR node, its updated cost will be the minimum of its children's exponential risk plus the edge cost to its minimum child, which represents choosing

the best action from the sub-problems. Because OR node will only take deterministic actions and the exponential risk function is equivariant to translation as proved in Chapter 3.2.2. Specifically, we have

$$f(v) = \min_{v' \in S(v)} \{f(v') + c(v, v')\}$$

– if the node v is an AND node, $f(v)$ is now updated as follows:

$$\begin{aligned} f(v) &= \frac{1}{w} \ln(\mathbb{E}(\exp(w \times \sigma(\phi_v)))) \\ &= \frac{1}{w} \ln(\sum_{v' \in S(v)} p(v, v') \mathbb{E}(\exp(w(\sigma(\phi'_v) + c(v, v'))))) \\ &= \frac{1}{w} \ln(\sum_{v' \in S(v)} p(v, v') \mathbb{E}(\exp(w\sigma(\phi'_v)) \times \exp(w \times c(v, v')))) \\ &= \frac{1}{w} \ln(\sum_{v' \in S(v)} p(v, v') \exp(wf(v')) \times \exp(w \times c(v, v'))) \\ &= \frac{1}{w} \ln(\sum_{v' \in S(v)} p(v, v') \exp(w(f(v')) + c(v, v'))) \end{aligned}$$

The form of the equation shows that the estimated exponential risk $f(v)$ can be calculated recursively in a bottom-up fashion easily without storing any extra information.

The only difference between the Offline-RCTP-AO* algorithm and the AO* algorithm is the state evaluation function, where Offline-RCTP-AO* evaluates a state's exponential risk instead of expected cost. Since both functions are monotonic for constants, the Offline-RCTP-AO* thus also has the property that if the heuristic being used is admissible, the returned policy is optimal. Moreover, when the algorithm returns, it either returns the solution tree or indicates that there does not exist a policy that guarantees to reach the goal. Thus the algorithm is also complete.

The returned policy of Offline-RCTP-AO* also maintains Bellman's principle of optimality, which states that each subpolicy of an optimal policy must itself be optimal regarding its initial state and terminal states. Recall that in our AND/OR tree representation, only OR

Algorithm 2 The Offline-RCTP-AO* Algorithm

```
1:  function OFFLINE-RCTP-AO*(s, h(·), t)
2:      f(s) ← h(s); s.type = OR; T.root ← s
3:      while s.status ≠ SOLVED and f(s) ≠ inf do
4:          v ← SELECTNODE(T.root)
5:          EXPAND(v)
6:          for v' ∈ S(v) do
7:              f(v') = h(v')
8:              if v' ∈ t then v'.status = SOLVED
9:          RCTP_BACKPROPAGATE(v, T)
10:     if T.root.val == inf then
11:         return NO SOLUTION
12:     return T
13:
14: function RCTP_BACKPROPAGATE(v, T)
15:     while v ≠ T.root do
16:         if S(v) ≠ ∅ then
17:             if v.type == OR then
18:                 v' = argmin_{v'∈S(v)}{f(v') + c(v, v')}
19:                 f(v) = c(v, v') + f(v')
20:                 if v'.status == SOLVED then v.status = SOLVED
21:             if v.type == AND then
22:                 f(v) = (1/w) ln Σ_{v'∈S(v)}{p(v, v') × exp(w[c(v, v') + f(v')])}
23:                 if v'.status == SOLVED ∀v' ∈ S(v) then v.status = SOLVED
24:         v = v.parent
```

nodes take actions. Our OR node cost update equation (line 23-24) just restates the Bellman optimality equation.

We can use the same admissible heuristic as we do for CTPs to assign the initial heuristic values for any newly added nodes during expansion, since in the deterministic case, the exponential risk has the same value as the deterministic cost. Thus when substituting into the exponential risk function, our estimated node costs are also admissible.

The offline algorithm is suitable for the following scenarios:

1. There is a known deterministic path (that can be very expensive) from the agent's position to the goal. Otherwise the algorithm will simply expand the whole implicit AND/OR tree and return infinity cost.

2. The agent has enough time to compute the offline optimal policy before starting to traverse the map.

As a demonstration, Figure 4.4 shows the result of applying the Offline-RCTP-AO* using the exponential risk with $w = 2$ to the same graph instance as in Figure 2.4. The returned

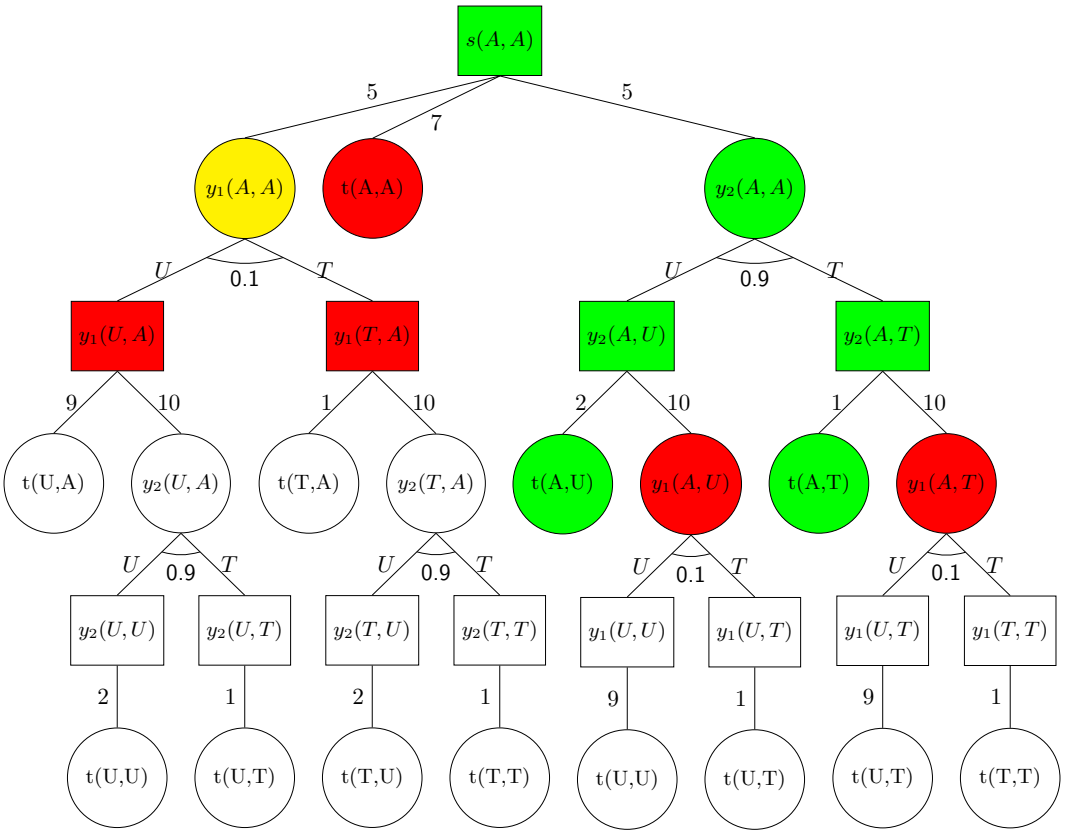


Figure 4.4: The complete AND/OR Tree corresponding to the RCTP Instance in Figure 2.4. OR nodes are depicted by squares and AND nodes by circles. Numbers below AND nodes are blocking probabilities of the corresponding edges to be disambiguated. Numbers beside edges are action costs. The policy tree returned by Offline-RCTP-AO* using exponential risk measure with $w = 2$ is colored in green. Extra expanded nodes are colored in yellow. Non-terminal leaves in the AO* expansion tree are colored in red.

policy is colored in green. It is actually the more robust policy in Figure 3.3. It has an expected cost of 6.9, a best-case cost of 6, and a worst-case cost of 7. The variance of this policy is 0.09 and the evaluated exponential risk (with $w = 2$) is 6.95. Like the classic AO* algorithm, the Offline-RCTP-AO* algorithm is able to find the desired policy by only visiting a small portion of the complete AND/OR tree when a good heuristic is being used.

Similar to AO*, the efficiency of Offline-RCTP-AO* largely depends on the quality of the heuristic being used. The worst case happens when the heuristic assigns the same value to every node and all the edges have the same cost. In that case, the algorithm will fully expand every internal layer of the complete AND/OR tree, and thus has a worst-case runtime of $O(|V|3^{|E|})$.

4.2.2 The Log-Sum-Exp Trick

Recall that the exponential risk with parameter w is given by $\gamma(\pi, w) = \frac{1}{w}\ln(\mathbb{E}(\exp(w\pi)))$, which requires computations of the following form:

$$z = \ln \sum_{n=1}^{N} \exp(x_n).$$

Depending on the scale of the x_n, computing this equation directly can cause potential numeric overflow or underflow problems. For instance, consider $\mathbf{x} = \begin{bmatrix} 0 & 1 \end{bmatrix}^T$, we can easily get $z = \ln(1 + e) \approx 1.31$. However consider another example where we have $\mathbf{x} = \begin{bmatrix} 1000 & 1001 \end{bmatrix}^T$, our computer will actually return $z = \inf$ due to the overflows when trying to compute $\exp(1000)$ and $\exp(1001)$. Similarly, we will get $z = \ln(0) = -\inf$ for $\mathbf{x} = \begin{bmatrix} -1000 & -1001 \end{bmatrix}^T$ as will have $\exp(-1000) = 0$ and $\exp(-1001) = 0$ due to the numeric underflows.

Fortunately, we can use the *log-sum-exp* trick [35] in the implementation step to get around the potential numeric overflow or underflow issue. The trick to resolve the problem exploits the following identity:

$$\ln \sum_{n=1}^{N} \exp(x_n) = a + \ln \sum_{n=1}^{N} \exp(x_n - a)$$

for any scalar a. This means we can freely shift the exponentiated variates up and down with exactly equivalent results. A common choice is thus to factor out the largest term, namely let

$$a = \max\{x_1, x_2, ..., x_n\}.$$

This means that the largest value of the exponential terms would be $\exp(0) = 1$, which ensures that no numeric overflow would happen. Moreover, if underflow happens to any of the exponential terms, the result would be still reasonable. Now if we look at the previous examples again, for $\mathbf{x} = \begin{bmatrix} 1000 & 1001 \end{bmatrix}^T$, we can now compute it as $z = 1001 + \ln(e^{-1} + 1) \approx 1001.31$ without any overflow issues. Similarly, for $\mathbf{x} = \begin{bmatrix} -1000 & -1001 \end{bmatrix}^T$, we can now compute it as $z = -1000 + \ln(1 + e^{-1}) \approx -999.69$ without worrying about underflows.

4.2.3 The Online-RCTP-AO* Algorithm

Though the Offline-RCTP-AO* algorithm is able to find the desired policy efficiently by using heuristics to prune the state space, it does not get rid of the curse of dimensionality and can still take a long time to compute the policy for large-scale problems. For live operation in real-world scenarios, it is often desired to come up with a traversal plan quickly, even if it is not optimal. Moreover, the offline algorithm assumes the existence of a deterministic path to the goal, otherwise the AO* algorithm would simply return infinite expected cost, though there are chances that the true underlying graph does have a traversable path to the goal.

In this section, we modify the offline algorithm to an online version by introducing a limit on the search depth to the AO* algorithm. When the search depth is reached, the algorithm will simply return the best partial tree. The agent will take the first best action from the partial solution tree and observe the outcomes. Note that in our problem formulation, an action means traveling to an end vertex of a stochastic edge and performing disambiguation, or traveling to the goal.

The agent will then keep replanning, executing the new plan, and making new observations until it reaches the goal or figures out that the goal cannot be reached. The Online-RCTP-AO* algorithm is depicted in Algorithm 3.

Due to the limited search depth, the Online-RCTP-AO* algorithm has to replan at every step of plan execution. Each replanning will take in more information to the planner, either from new observations, or deeper search depths in the complete AND/OR tree. Another way to think of the Online-RCTP-AO* algorithm is that at every step, the planner will pretend the stochastic edges that are out of reach of the search depth are always traversable. The online planner will thus simulate what the offline planner will do, but with a limited foresight. Since the online planner continuously adjusts its plan from online observations and will only take one action at every step, we now lose the side product that the offline algorithm produces - an AND/OR solution tree that can be used to calculate the statistics of the policy. However, the online algorithm in many cases, will produce competitive or even equivalent results to the offline algorithm results. Since the number of nodes in the AND/OR tree grows exponentially to the tree depth, the Online-RCTP-AO* algorithm will respond exponentially faster than the

Offline-RCTP-AO* algorithm due to the limit on the search tree depth.

Though not being optimal, the Online-RCTP-AO* algorithm is still complete. At every execution step, the algorithm will disambiguate one stochastic edge. In the worst case, it will eventually disambiguate all the stochastic edges and obtain full knowledge of the map, and thus will either find it way to the goal or return no solution.

Algorithm 3 The Online RCTP-AO* Algorithm

function ONLINE-RCTP-AO*($s, h(\cdot), t, d$)
  while $s \neq t$ **do**
    $T \leftarrow$ DEPTH-AO*($s, h(\cdot), t, d$)
    if $T.root.val == inf$ **then**
      return NO SOLUTION
    $v \leftarrow \text{argmin}_{v \in S(T.root)}(f(v) + c(T.root, v))$
    $s \leftarrow$ TRAVERSEANDOBSERVE(s, v)

function DEPTH-AO*($s, h(\cdot), t, d$)
  $s.val \leftarrow h(s)$; $s.type =$ OR; $T.root \leftarrow s$
  while $s.status \neq$ SOLVED **and** $f(s) \neq inf$ **or** $T.depth < d$ **do**
    $v \leftarrow$ SELECTNODE($T.root$)
    EXPAND(v)
    for $v' \in S(v)$ **do**
      $f(v') = h(v')$
      if $v' \in t$ **then** $v'.status =$ SOLVED
    RCTP_BACKPROPAGATE(v, T)
  return T

function RCTP_BACKPROPAGATE(v, T)
  while $v \neq T.root$ **do**
    if $S(v) \neq \emptyset$ **then**
      if $v.type ==$ OR **then**
        ▷ c(v,v') is the cost from v to v'
        $v' = \text{argmin}_{v' \in S(v)}\{f(v') + c(v, v')\}$
        $f(v) = c(v, v') + f(v')$
        if $v'.status ==$ SOLVED **then** $v.status =$ SOLVED
      if $v.type ==$ AND **then**
        $f(v) = \frac{1}{w} \ln \sum_{v' \in S(v)}\{p(v, v') \times \exp(w[c(v, v') + f(v')])\}$
        if $v'.status ==$ SOLVED $\forall v' \in S(v)$ **then** $v.status =$ SOLVED
    $v = v.parent$

4.3 Summary

In this chapter, we presented three algorithms: the original AO* algorithm that can be directly used to solve classic CTPs; the Offline-RCTP-AO* algorithm, which changes the AO* algo-

rithm to return the policy with the minimum exponential risk instead of the minimum expected cost; and finally the Online-RCTP-AO* algorithm that imposes a search depth on the Offline-RCTP-AO* algorithm, and constantly replans after each action and observation.

The Offline-RCTP-AO* algorithm is complete and optimal, which means it will always find the optimal solution if one exists. It is suitable when there exists a deterministic path from the starting position to that goal. Otherwise, if the goal cannot be reached without attempting at least one stochastic edge, both the AO* and Offline-RCTP-AO* algorithm will try to expand the complete AND/OR tree and return infinite cost. Like AO*, the Offline-RCTP-AO* algorithm is able to find the optimal solution by only visiting a small portion of the complete search space.

However, the Offline-RCTP-AO* algorithm still suffers from the curse of dimensionality as the search space does increase exponentially. As the name suggests, the Offline-RCTP-AO* algorithm may not return a solution promptly for large-scale problems. Thus we also proposed the Online-RCTP-AO* algorithm. It approximates the offline version of the algorithm with a limited search depth, but replans at every step when new observations are available. Since the depth of tree expansion is limited during plan computation, the Online-RCTP-AO* algorithm will respond exponentially faster than the Offline-RCTP-AO* algorithm for some reasonable range of search depths. In the worst case, the Online-RCTP-AO* algorithm will disambiguate all the stochastic edges and plan with full knowledge of the map at the last step. The Online-RCTP-AO* algorithm is thus complete, though not optimal.

Chapter 5

Experiments

To evaluate the proposed algorithms, experiments were conducted in both simulated environments as well as on a map of 5 km of paths built from robot field trials. To evaluate the Offline-RCTP-AO* algorithm, we compared the policies found by solving RCTP with a grid search over different risk weights against the policies found by solving CTP. We also compared the policies with a naive policy that continuously uses Dijkstra's algorithm to find the shortest path to the goal and replans when being blocked. Finally, all policies were also compared with the optimal policy: shortest path to the goal when the true status of the graph is known (can only be known by an oracle). To evaluate the Online-RCTP-AO* algorithm, besides the same set of experiments as mentioned above, the performances of running the online algorithm with different search depths were also compared.

5.1 The Simulated Environment

The simulated environment consists of a series of graphs generated by randomly placing 98 distinct vertices in a 100×100 coordinate grid, plus the two vertices at (0,0) and (99, 99); a set of edges were added by computing a spanning tree to ensure the vertices are connected. Additional edges were added by randomly selecting the edges from the Delaunay triangulation of the vertices until the graph got a total of 150 distinct edges. Edge costs were represented as the Euclidean distance between the end vertices. To set up the graph instance for the CTP and RCTP experiments, the vertex (0, 0) was always chosen as the start vertex, and (99, 99) was

always chosen as the goal. Each edge had a 0.2 probability of being stochastic and the blocking probability was uniformly sampled from the interval 0 to 1. Figure 5.1 shows an example of a randomly generated graph, with deterministic edges marked in black and stochastic edges shown in red (stronger lines having greater chances of being blocked). There are 150 edges in this map, 22 of them are stochastic edges. The starting position is marked in green and the goal is marked in red.

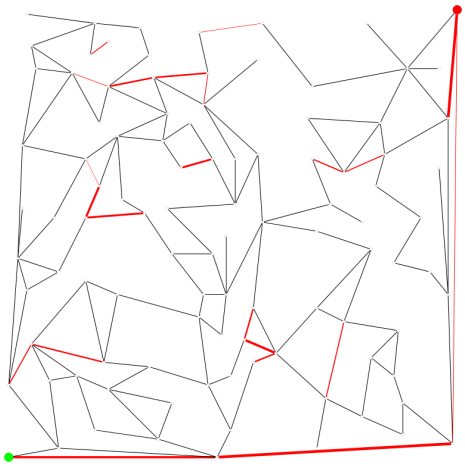


Figure 5.1: A simulated environment example, with start vertex marked in green and goal marked in red. Edge costs are the Euclidean distance between the end vertices. Deterministic edges are shown in black. Stochastic edges are shown in red, with stronger lines having higher probabilities of being blocked. There are 150 edges in this map, 22 of them are stochastic edges.

5.1.1 The Offline-RCTP-AO* Algorithm Experiments

For the Offline-RCTP-AO* algorithm experiments, to avoid infinite expected policy cost, we only chose graphs that had a deterministic path to the goal (assuming all stochastic edges are blocked). To avoid all methods trivially choosing the deterministic path, we only chose graphs where the shortest path to the goal contained at least one stochastic edge.

In Figure 5.2 and Figure 5.3 we show the results of 1000 trials over each of the policies acquired on the example graph in Figure 5.1. For each trial, the true traversability status of each stochastic edge was drawn according to its blocking probability. The RCTP policies were acquired by doing a grid search over increasing weights. For simplicity, we only plotted the results for the weights that caused a policy change, where “rctp-w” means the policy acquired

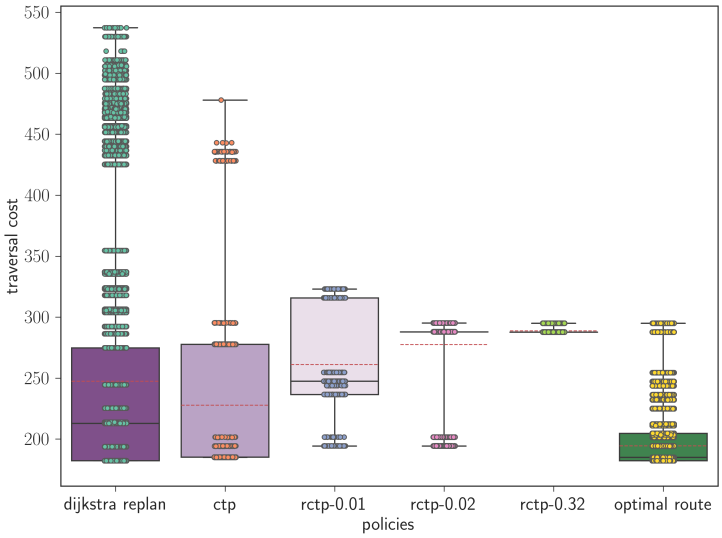
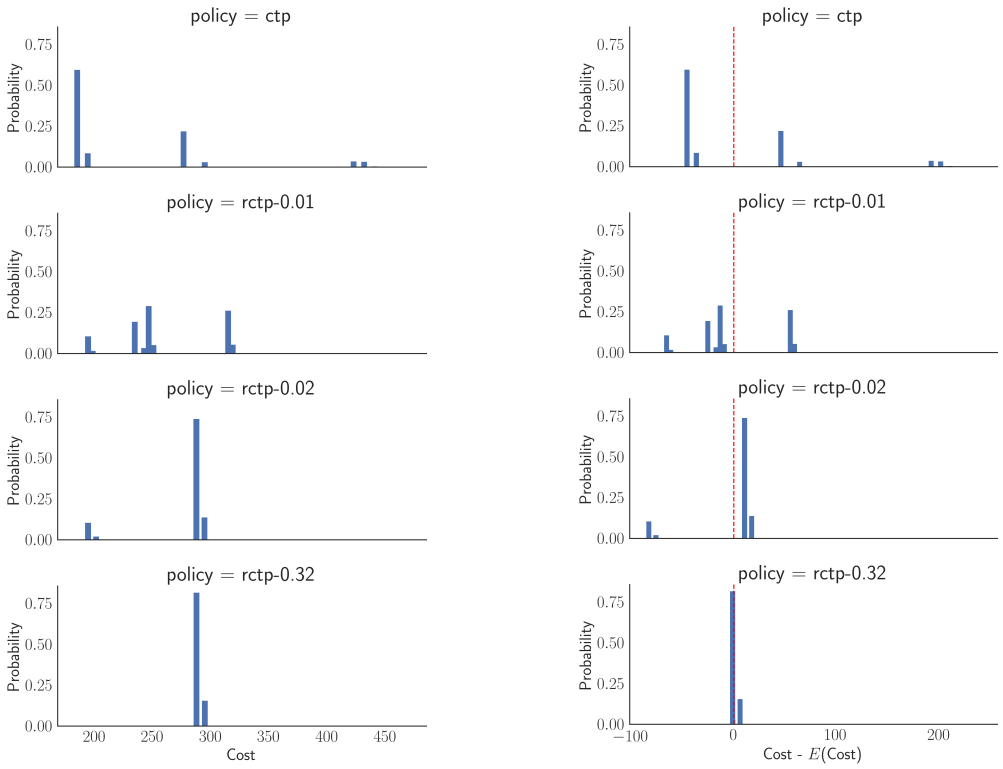


Figure 5.2: Boxplots of simulating 1000 traversal costs for different policies on Figure 5.1, with mean represented by the red dashed line and scatter points approximating the possible traversal outcomes. "rctp-w" represents the policy acquired by solving RCTP with weight w. Note that the true optimal path cost can only be calculated in hindsight.

by solving RCTP with weight w.

Figure 5.2 shows the boxplots of the actual traversal costs with mean represented by the red dashed line and scatter points approximating the possible traversal outcomes. We can see that doing Dijkstra's search with replanning yields the most unstable policy – it has the most spread out possible outcomes and the largest worst-case cost.

Figure 5.3 (a) shows the distribution of the actual traversal cost by following each of the different policies. Figure 5.3 (b) shows the distribution of the actual traversal cost minus the expected cost of the corresponding policy. By comparison, we can see that the non-robust policy (acquired by solving CTP) obtains the lowest expected cost, but also has a long tail as evidenced by the shape of the histogram. As we increase the weights, the obtained policy becomes more and more risk-averse as desired, resulting in smaller worst-case costs and more centered distributions. Importantly, as evidenced by Figure 5.3 (b), when policy changes as the weight increases, the exponential risk measure puts more penalty on deviations above the mean than deviations below. Moreover, when solving the problem with classic methods such as Dijkstra's algorithm with replanning, or a classic MDP solver using policy iteration or value iteration, we only obtain a traversal policy without much additional information. But by using the AND/OR tree representation and viewing the policy cost as a categorical distribution,



(a) (b)

Figure 5.3: Results of 1000 trials over each of the policies acquired by solving the problem on Figure 5.1. "rctp-w" means the policy acquired by solving RCTP with weight w. (a) shows the distribution of each policy's traversal cost. (b) shows the distribution of each policy's traversal cost minus its expected cost.

not only do we obtain a policy minimizing the weighted exponential risk measure, but also a statistical description of the costs as a byproduct: the probability of each possible outcome is computed as well. By comparing the policy cost distributions from different weights, the user has information on how to tune the parameter and find the most appropriate policy to trade off the mean and worst-case cost.

Figure 5.4 shows the plot of the returned policy's exponential risk versus different choices of the weight w by solving the RCTP on Figure 5.1. We can see that the returned policy's exponential risk is always increasing as we increase the weight, though the returned policy might not change. This is because the exponential risk function is monotonic and we keep increasing the weight on the exponential term.

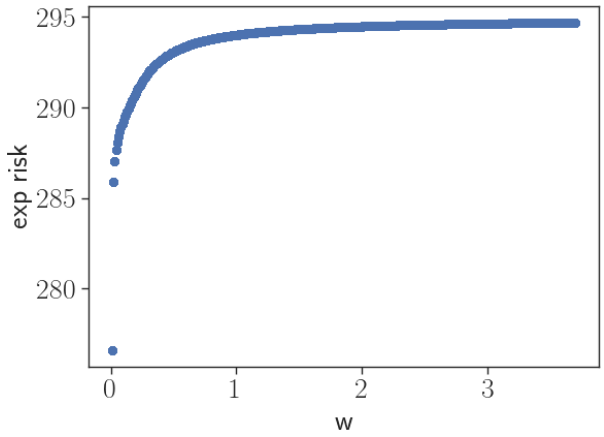
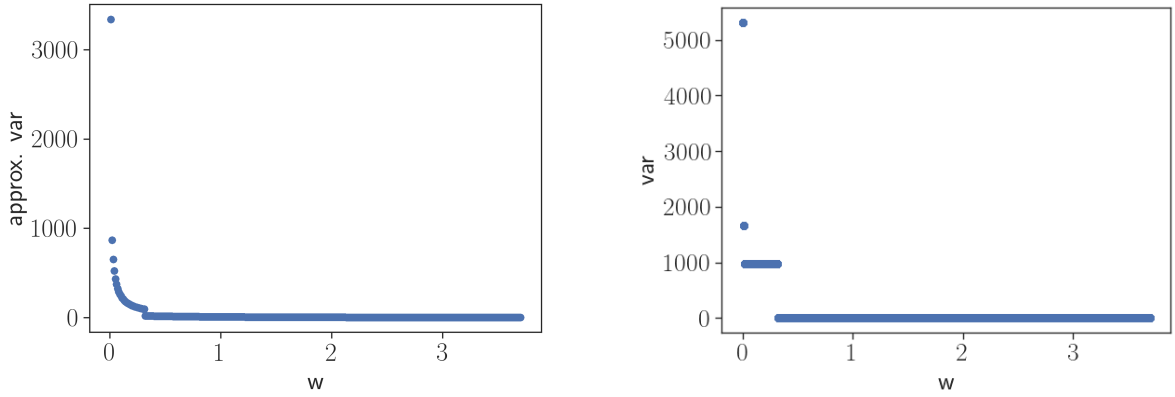


Figure 5.4: The plot of the returned policy's exponential risk versus different choices of the weight w by solving the RCTP on Figure 5.1.



(a) (b)

Figure 5.5: (a) The plot of the returned policy's approximated variance (calculated using the policy's exponential risk) versus different choices of the weight w by solving the RCTP on Figure 5.1. (b) The plot of the returned policy's variance (calculated through the policy's categorical distribution) versus different choices of the weight w by solving the RCTP on Figure 5.1.

We showed in Chapter 3.2.2 that the exponential risk measure can be thought of as approximating policy cost mean plus weighted variance. We can thus use a policy's returned exponential risk to calculate an approximate value of the policy's variance. In Figure 5.5, (a) shows the plot of the returned policy's approximated variance (calculated using the policy's exponential risk) versus different choices of the weight w by solving the RCTP on Figure 5.1. (b) shows the returned policy's variance (calculated through the policy's categorical distribution) versus the same choice of the weight w. We can see that the two plots have similar shapes,

which is consistent with our claim. As the weight increases, the algorithm is (approximately) looking for policies that have smaller variances.

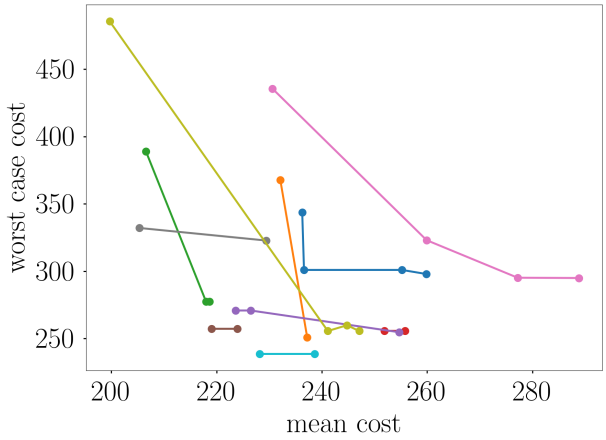


Figure 5.6: Policy worst-case cost versus mean cost for policies on 10 different randomly simulated graphs. Each line shows how the worst-case cost changes as the policy mean increases for the series of policies obtained from a given graph.

To analyze the trade-off between policy mean and worst-case costs, experiments were conducted on 10 different simulated graphs. For each graph, the CTP policy was computed and a series of different weights were applied to obtain the set of RCTP policies, and for each policy, 1000 trials were simulated. The result is shown in Figure 5.6, where each line shows how the worst-case cost changes as the policy mean increases for the series of policies obtained from a given graph. We can see from the graph that the lines generally follow a decreasing trend as the means increase. In many cases, increasing the mean cost by a small amount at the early phase can result in a big decrease in the worst-case cost.

5.1.2 The Online-RCTP-AO* Algorithm Experiments

To analyze the performance of the Online-RCTP-AO* algorithm and compare with the Offline-RCTP-AO* algorithm, experiments were conducted on the same graph instance in Figure 5.1. The Online-RCTP-AO* algorithm was run with different exponential risk weights and search depths. When weight w equals 0, the algorithm will try to minimize the expected policy cost instead of the exponential risk, reverting back to solving classic CTPs. For each experiment setting, 100 trials were conducted. For each trial, the true traversability status of each stochastic

edge was drawn according to its blocking probability. For simplicity, we only selected the weights that caused a policy change in the Offline-RCTP-AO* algorithm.

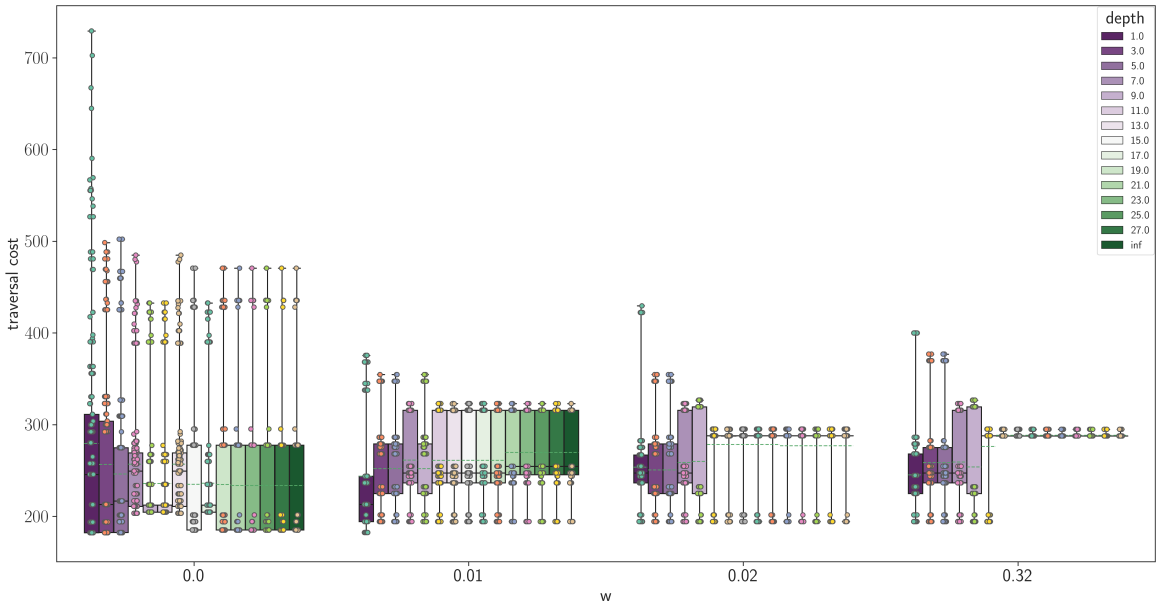


Figure 5.7: Boxplots of simulating 100 traversal costs for solving the RCTP on Figure 5.1 using the Online-RCTP-AO* algorithm with different exponential risk weights w and search depths. In each box, mean is represented by the dashed line and scatter points are approximating the possible traversal outcomes.

Figure 5.7 shows the boxplots of simulating 100 traversal costs for solving the RCTP on Figure 5.1 using the Online-RCTP-AO* algorithm with different exponential risk weights w and search depths. When weight w equals 0, the algorithm will try to minimize the expected policy cost instead of the exponential risk, reverting back to solving classic CTPs. When depth equals infinity, it is equivalent to solving the problem using the Offline-RCTP-AO* algorithm. In each box, mean is represented by the dashed line and scatter points are approximating the possible traversal outcomes. From the boxplots, we can see that when weight w is fixed, the online algorithm gradually converges to the offline algorithm as the search depth increases. Moreover, the online algorithm converges at smaller depths when larger weights are used, due to the use of stronger heuristics.

Figure 5.8 (a) shows the plot of the mean costs versus search depths of simulating 100 traversals for solving the RCTP on Figure 5.1 using the Online-RCTP-AO* algorithm with different exponential risk weights w and depths. The dashed line represents the result of running

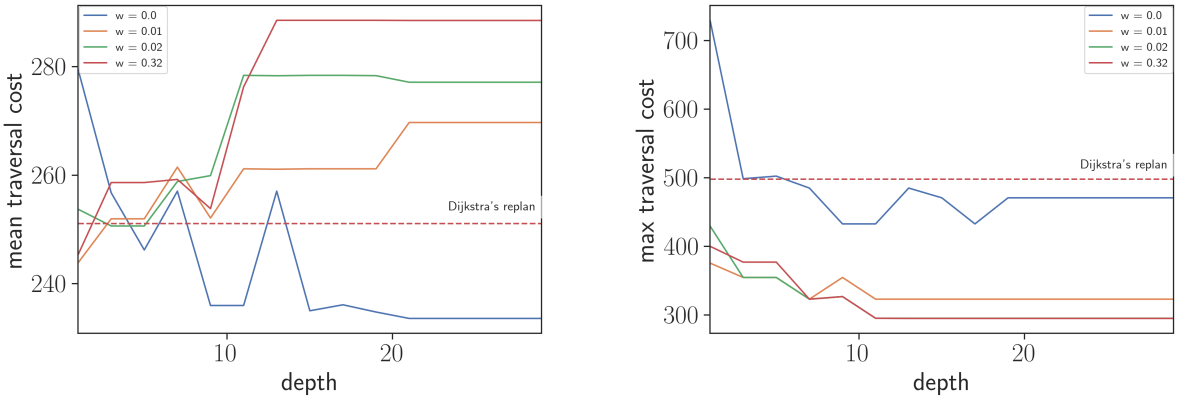


Figure 5.8: (a) The plot of the mean costs versus search depths of simulating 100 traversals for solving the RCTP on Figure 5.1 using the Online-RCTP-AO* algorithm with different exponential risk weights w and depths. The dashed line represents the results of running the Dijkstra's algorithm with replanning for the same experiments. (b) The plot of the worst-case costs versus search depths of simulating 100 traversals for solving the RCTP on Figure 5.1 using the Online-RCTP-AO* algorithm with different exponential risk weights w and depths. The dashed line represents the results of running the Dijkstra's algorithm with replanning for the same experiments.

the Dijkstra's algorithm with replanning for the same experiments. We can see that most of the time, solving the classic CTP ($w = 0$) returns the minimum mean cost. Except when the search depth is small where the agent will take action when not much information is available. This is consistent with our claim that the goal of solving CTPs is trying to minimize the expected cost of the policy, whereas the goal of solving RCTPs is not. Then as w increases, in most cases the mean cost increases as well. But at every search depth, the mean costs of the policies with different weights are not far from each other.

Figure 5.8 (b) shows the plot of the worst-case costs versus search depths for the same experiments. We can see that Dijkstra's algorithm with re-planning has the largest worst-case cost most of the time. This is consistent with our offline algorithm results that Dijkstra's algorithm has the most unstable performance. In most cases, the trying to solve the classic CTP yields the second-largest worst-case cost. Then as w increases, the worst-case cost decreases. This again, is consistent with our claim that the RCTP-AO* algorithm will become more and more risk-averse as we put more weights on the exponential risk.

Combining Figure 5.7 and Figure 5.8 together, we can see that the online algorithm can

perform reasonably well without reaching the full search depth.

Figure 5.9 shows the runtime comparisons of different settings of the Online-RCTP-AO* algorithm versus the classic offline AO* algorithm. For each online experiment, the running time was accumulated for every step of replanning. The replanning usually happened 3 to 6 times. Then the average total planning time over 100 simulations was calculated for each experiment setting. We can see that in all cases, the running time increases approximately exponentially as the search depth increases, and then freezes when the online algorithm converges to the offline algorithm when certain search depth is reached. This is because the search space increases exponentially as the search depth increases. Solving the CTP ($w = 0$) is more expensive than solving the RCTP ($w > 0$) due to the use of weaker heuristics at every planning step. The total running time decreases when more weight is put on the exponential risks, which results in stronger heuristics. More importantly, the plot suggests that for complex graph domains (like in our experimental setting) where offline algorithms cannot come up with the optimal plan quickly, the online algorithms can be used to provide good approximate solutions promptly. Especially when the exponential risk measure is used to seek for robust plans, the accumulated total running time over all steps can still be much smaller than the time needed for a single pass of the offline algorithm trying to solve the classic CTP.

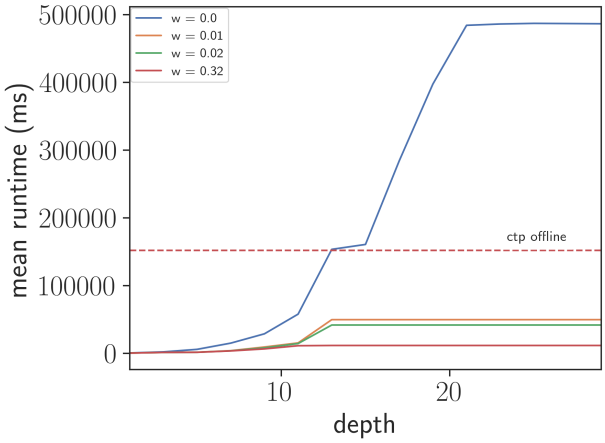


Figure 5.9: The plot of the average total planning time (accumulated over every step of replanning) versus search depths of simulating 100 traversals for solving the RCTP on Figure 5.1 using the Online-RCTP-AO* algorithm with different exponential risk weights w and search depths. The dashed line represents the result of running the offline classic AO* algorithm for the same experiments.

5.2 The Sudbury Map Environment

We also applied the algorithms to a real-world domain where pose graphs are used for robot navigation. In these experiments, we used a map dataset that was collected in Sudbury, Canada [43]. The dataset was generated as a byproduct for testing the VT&R algorithm ([18], [41]), which is a vision-based navigation system that enables a mobile robot to autonomously repeat previously demonstrated paths by only using visual data from a stereo camera. The algorithm computes and saves a rich history of information in its STPG map structure during the course of autonomous traversals, such as localization uncertainties, obstacles detected, and traversal success rates.

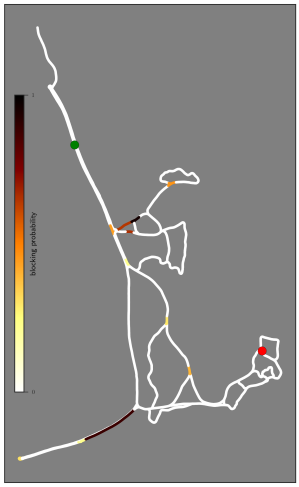


Figure 5.10: An example of the Sudbury experiment map, with darker edges having larger chances of being blocked. The starting position is marked in green, the goal is marked in red. The Sudbury map consists of 172 refactored paths, in this example 11 of them are stochastic.

Within the VT&R2 framework, the route planner works on the spatial-temporal pose graph (introduced in Chapter 2.3) constructed in the map building phase. The framework generates vertices with a roughly constant linear spacing of approximately 30 cm. Since the AO* algorithm has an exponential complexity in $|E|$, where $|E|$ is the number of stochastic edges, this can be a problem for large networks with thousands or tens of thousands of edges. However, noticing the fact that for most pose graphs, the path connecting two junctions (intersecting points of two or more paths) often consists of hundreds of edges chained together. Thus the original pose graphs can be reduced to graphs that only contain junctions and endpoints, with each edge chain connecting two junctions being wrapped as a single edge. After this simple

refactorization, the complexity of the problem is now reduced to be exponential in $|W|$ where $|W|$ is the number of junctions.

The Sudbury dataset was selected to see how the offline and online RCTP algorithms perform in practical map structures. We assume the robot is able to observe the true status of a stochastic edge at its endpoints using its sensors. We then used the same method as before to randomly assign stochastic edges and blocking probabilities. For the Sudbury map, edge costs were the actual distances between two vertices, in the unit of meters. The same start-goal pair was used for all experiments. Figure 5.10 shows an example of the experiment graph, with darker edges having a higher chance of being blocked. The starting position is marked in green and the goal is marked in red. The Sudbury map consists of 172 re-factored paths; in this example 11 of them are stochastic.

5.2.1 The Offline-RCTP-AO* Algorithm Experiments

The offline experiment settings were similar to what we did for the simulated environment: to avoid infinite expected policy cost, we only chose graphs that had a deterministic path to the goal (assuming all stochastic edges are blocked). To avoid all methods trivially choosing the deterministic path, we only chose graphs where the shortest path to the goal contained at least one stochastic edge.

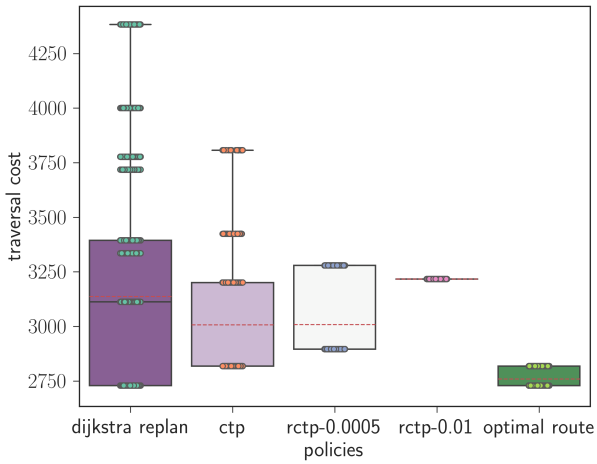


Figure 5.11: Boxplots of simulating 1000 traversal costs for different policies on Figure 5.10, with mean represented by the red dashed line and scatter points approximating the possible traversal outcomes. “rctp-w” represents the policy acquired by solving RCTP with weight w. Note that the true optimal path cost can only be calculated in hindsight.

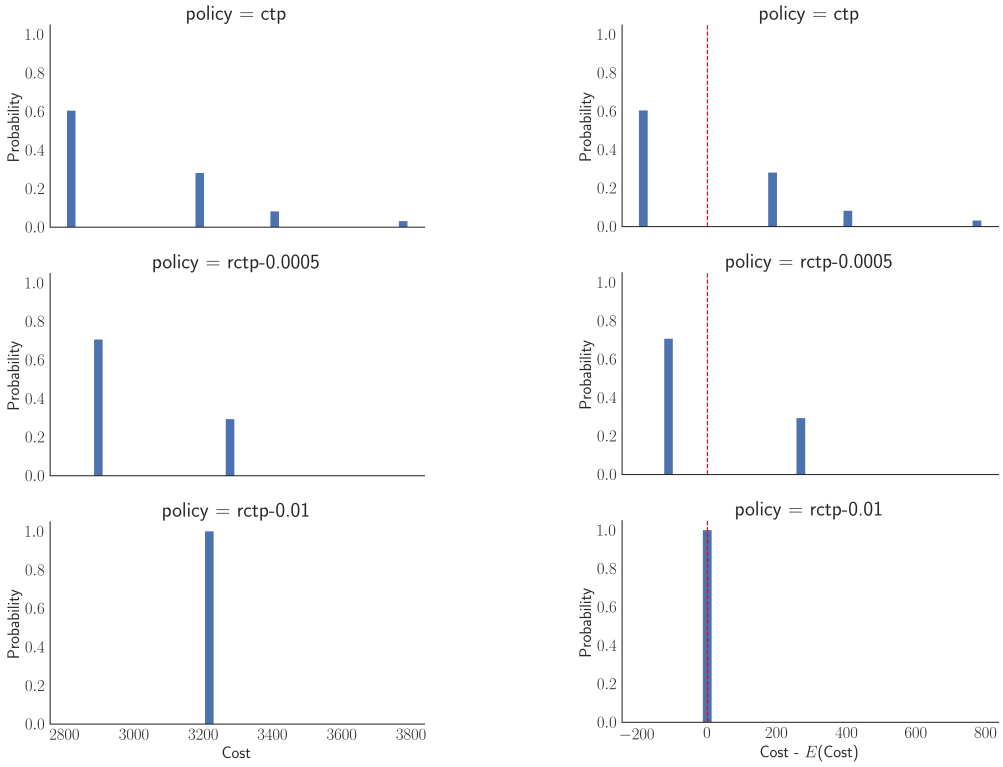


Figure 5.12: Results of 1000 trials over each of the policies acquired by solving the problem on Figure 5.10. "rctp-w" means the policy acquired by solving RCTP with weight w. (a) shows the distribution of each policy's traversal cost. (b) shows the distribution of each policy's traversal cost minus its expected cost.

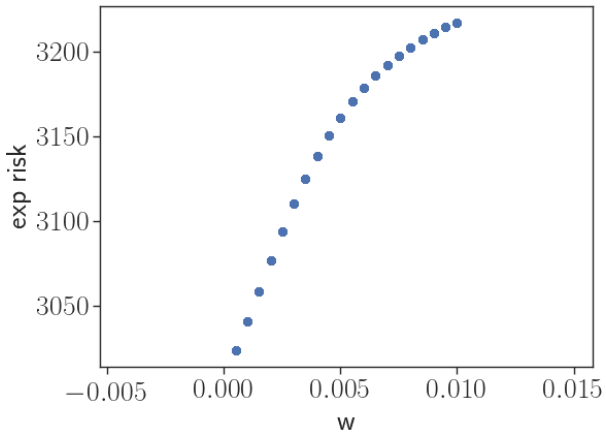
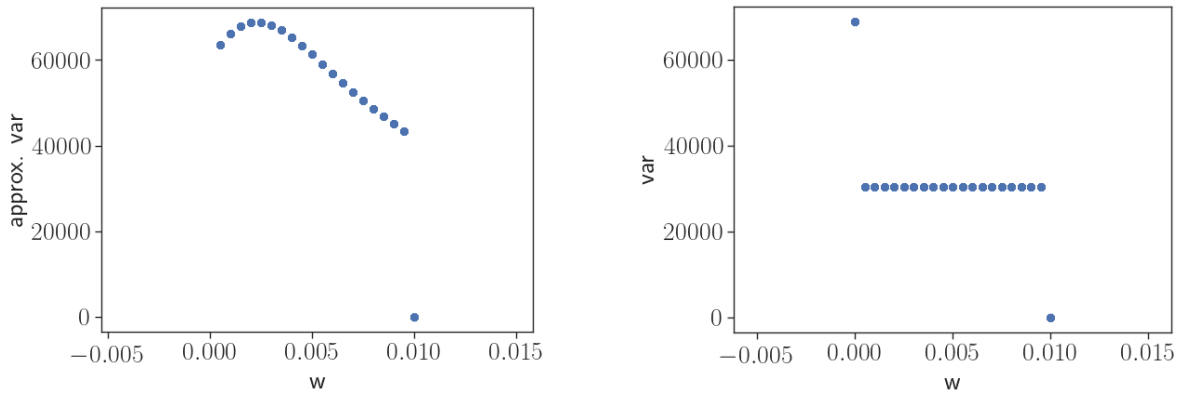


Figure 5.13: The plot of the returned policy's exponential risk versus different choices of the weight w by solving the RCTP on Figure 5.10



(a) (b)

Figure 5.14: (a) The plot of the returned policy's approximated variance (calculated using the policy's exponential risk) versus different choices of the weight w by solving the RCTP on Figure 5.10. (b) The plot of the returned policy's variance (calculated through the policy's categorical distribution) versus different choices of the weight w by solving the RCTP on Figure 5.10.

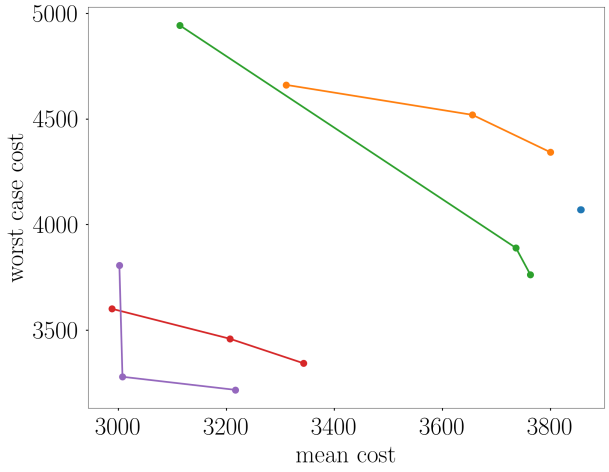


Figure 5.15: Policy worst-case cost versus mean cost for policies on 5 different randomly simulated graphs. Each line shows how the worst-case cost changes as the policy mean increases for the series of policies obtained from a given graph.

As in the other simulations, 1000 trials were conducted over each of the policies acquired by solving the RCTP problem with different weights on the example graph in Figure 5.10. The results are shown in Figure 5.11, Figure 5.12, Figure 5.13, Figure 5.14 and Figure 5.15 where Figure 5.15 in particular shows the mean versus worst-case trade-off by doing the same series of experiments as before over 5 different graphs instances. We can see that all results are consistent with the observations we made in the simulated environment experiments.

5.2.2 The Online-RCTP-AO* Algorithm Experiments

To analyze the performance of the Online-RCTP-AO* algorithm on the Sudbury data set and compare with the Offline-RCTP-AO* algorithm, experiments were conducted on the same graph instance in Figure 5.10 as well. Similar to what we did in the simulated environment, the Online-RCTP-AO* algorithm was run with different exponential risk weights and search depths. When weight w equals 0, the algorithm will try to minimize the expected policy cost instead of the exponential risk, reverting back to solving classic CTPs. For each experiment setting, 100 trials were conducted. For each trial, the true traversability status of each stochastic edge was drawn according to its blocking probability. For simplicity, we only selected the weights that caused a policy change in the Offline-RCTP-AO* algorithm.

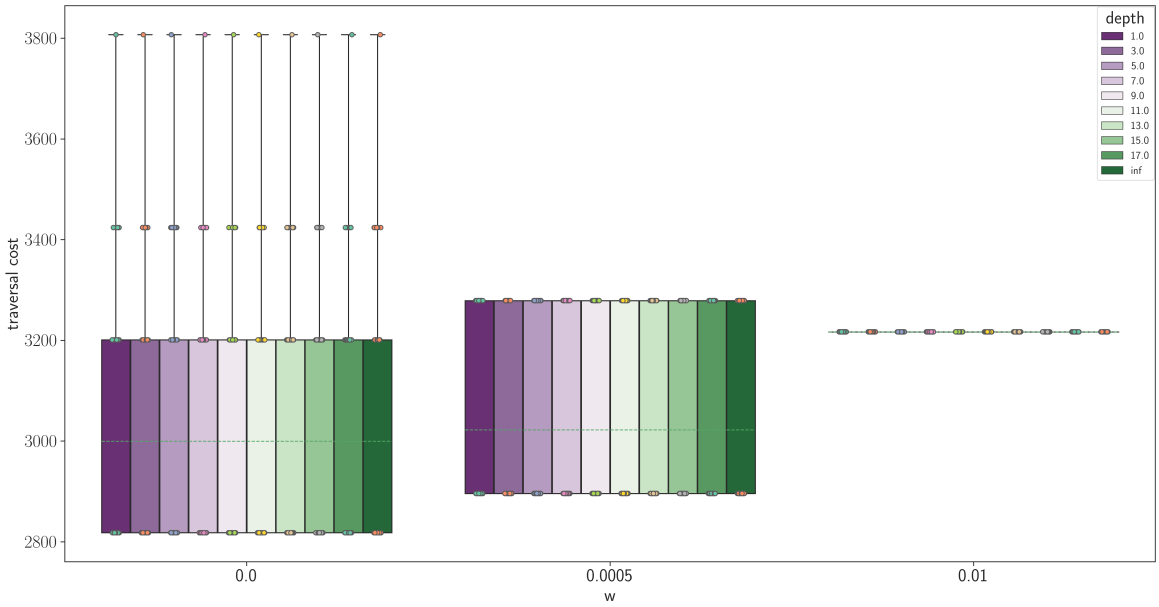


Figure 5.16: Boxplots of simulating 100 traversal costs for solving the RCTP on Figure 5.10 using the Online-RCTP-AO* algorithm with different exponential risk weights w and search depths. In each box, mean is represented by the dashed line and scatter points are approximating the possible traversal outcomes.

Figure 5.16 shows the boxplots of simulating 100 traversal costs for solving the RCTP on Figure 5.1 using the Online-RCTP-AO* algorithm with different exponential risk weights w and depths. When weight w equals 0, the algorithm will try to minimize the expected policy cost instead of the exponential risk, reverting back to solving classic CTPs. When depth equals infinity, it is equivalent to solving the problem using the Offline-RCTP-AO* algorithm. From

the boxplots, we can see that for this particular problem setting, the online algorithm would return the same policy as the offline algorithm even at search depth 1, suggesting that the heuristic being used was strong enough to take good (almost optimal) actions without foreseeing all the sequential outcomes.

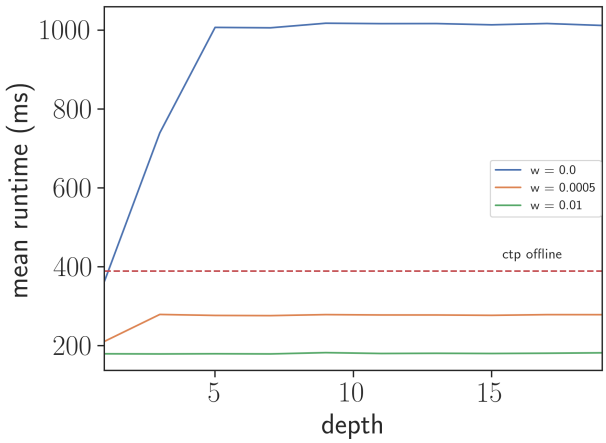


Figure 5.17: The plot of the average total planning time (accumulated over every step of replanning) versus search depths of simulating 100 traversals for solving the RCTP on Figure 5.10 using the Online-RCTP-AO* algorithm with different exponential risk weights w and search depths. The dashed line represents the results of running the offline classic AO* for the same experiments.

Figure 5.17 shows the runtime comparisons of different settings of the Online-RCTP-AO* algorithm versus the classic offline AO* algorithm. For each online experiment, the running time was accumulated for every step of replanning. The replanning usually happened 2 to 3 times. Then the average total planning time over 100 simulations was calculated for each experiment setting. We can see that the results are again consistent with the observations we made in the simulated environment experiments.

5.3 Summary

In this chapter, we showed the experimental results of running the Offline-RCTP-AO* algorithm and the Online-RCTP-AO* algorithm on both simulated environments and a 5 km map dataset that we collected in Sudbury, Ontario. The algorithms were also compared with a baseline method that always uses Dijkstra's algorithm to find the shortest path to the goal, and then replans when being blocked.

From the offline experiments, we can see that the baseline approach that uses Dijkstra's algorithm with replanning always yields the most unstable performance with the largest worst-case cost. The classic CTP solution will return the policy with the minimum expected cost, but also at the risk of ending up with large worst-case costs. Whereas for RCTP solutions, as we put more weight on the exponential risk, the returned policy will be more and more risk-adverse, with more centered cost distributions. Moreover, though the RCTP planner can be thought of as approximately minimizing variance-adjusted mean, it will penalize deviation above the mean more than deviation below. Thus as the exponential risk weight increases, the RCTP planner will (almost) always try to find policies that have lower worst-case costs first before it starts to sacrifice the best-case cost. The experimental results suggest that in many cases, the RCTP planner is able to find policies with significant lower worst-case cost, but with little sacrifice on the mean cost.

Experiments were also conducted to compare the performance of the Online-RCTP-AO* algorithm against the Offline-RCTP-AO* algorithm. The experimental results suggest that the online algorithm will gradually converge to the offline algorithm as the search depth increases. The online algorithm can perform reasonably well for some small search depths, while preserving the effect of reducing worst-case cost when exponential risk weight is increased. The experimental result also validates that the online algorithm is much faster than the offline algorithm, due to the limited search depth on the exponentially-growing search space. Moreover, the runtime comparison of solving CTPs and RCTPs on the same graph instances shows that the AO* algorithm will return faster when the exponential risk is being used, and thus deeper search depths can be tolerated for online performance.

Chapter 6

Ideas for Future Integration with VT&R

In order to solve the RCTP problem, the robots should first have an assumption over the stochastic paths in the map, and then have a mechanism to observe the traversability status of the terrain. In this chapter, we will thus discuss the approaches to integrate the RCTP planner into robot routing systems. Specifically, the necessary map preparation steps and system designs of the Visual Teach and Repeat (VT&R) framework will be presented as an illustration.

6.1 Assigning Stochastic Edges

The VT&R system uses the spatial-temporal pose graph (STPG) to structure the map environment and records auxiliary information of the driving history, such as vehicle velocities, landmarks observed, localization feature matches, manual interventions, and detection of obstacles. In the discussion of the Sudbury map experiments (Chapter 5.2), we have shown the benefit of refactoring the map to only contain vertices that are junctions and endpoints. In this section, we will talk about how to exploit the historical data to assign stochastic edges and their blocking probabilities.

During the repeat phase of VT&R, the robot localizes against the STPG to traverse the path. The localization is done by matching all landmarks (extracted by visual descriptors) in the live view to landmarks observed in a selected region of the STPG. Paton et al. [42] showed that the localization successful rate for an autonomous repeat is highly influenced by the illumination change, which is dominated by the time difference in day (0 to 12 hours) between the live task

and the experiences to be matched.

Figure 6.1 shows an analysis of the number of feature matches for the VT&R field trials spanning multiple seasons. The experiments were done by running the single-experience VT&R algorithm, which does not make use of bridging experiences to aid in localization. The figure shows that the number of landmark matches often drops below a critical threshold for robust localization if the time difference is greater than 4 to 6 hours, depending on the season and the landscape of the environment.

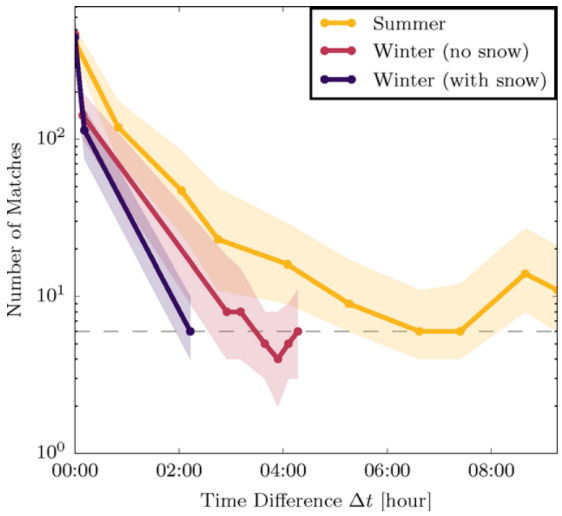


Figure 6.1: Evolution of the number of feature matches for the VT&R field trials spanning multiple seasons. The thick lines correspond to the median number through a full repeat path, and the shaded area defines the inter-quartile distance 25-75 %. The dashed line represents the critical threshold for robust localization. The x-axis represents the time difference, Δt, between the teach and the repeat path. Note the log scale on the y-axis. Figure reproduced from [42].

Thus during the first several attempts of the repeats where not much historical data is gathered, the time difference in day between the live task and the closest experience will play as the dominant indicator of whether the repeat will be successful or not. Some heuristic function can then be constructed to use the time difference in day between the live task and the closest experience to predict the probability of an edge being blocked. An example sigmoid function is given below:

$$p_{\text{block}} = \frac{1}{1 + e^{-(x-6)}}$$

The shape of the sigmoid function is shown in Figure 6.2. A path will be assigned a 0.5 probability of being blocked if it is 6 hours away from the closest experience. The blocking probability will gradually grow to 1 when the closest experience is 12 hours away, and will gradually decrease to 0 when the closest experience is within the same hour.

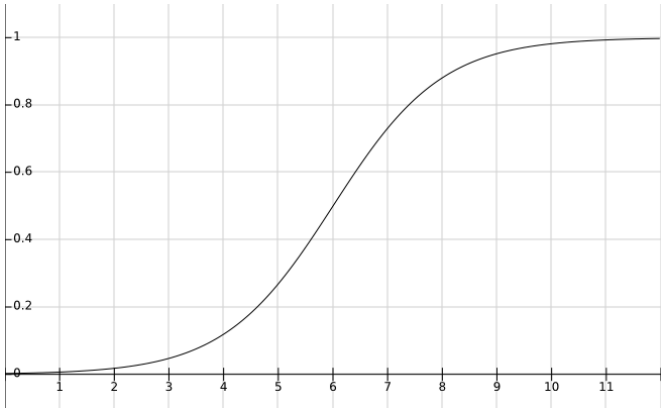


Figure 6.2: The shape of an example sigmoid function that uses time difference in day between the live task and the closest experience to predict the probability of an edge being blocked.

Drastic illumination change is just one of the factors that may cause a repeat failure. Other factors such as the presence of unexpected obstacles and challenging terrain that requires manual interventions can also interrupt the autonomy of the system. As the framework makes more attempts to repeat the map and gathers more experiences, it is also feasible to switch the blocking probability heuristic to be proportional to the frequency of manual interventions and task abortions.

6.2 Obstruction Detection and Loss Handling

The RCTP framework is dependent on the assumption that the robot is able to observe traversability status of the paths at their endpoints. Here we mainly concern two scenarios that will make a path untraversable: obstacle blockage and robot getting lost. In this section, we will thus discuss the mechanisms that the VT&R framework can utilize to achieve obstruction detection and loss handling.

6.2.1 Obstruction Detection

Obstacle detection is an important topic in mobile robotics, especially when humans are involved in the environment. Detecting humans and obstacles in an autonomous system can not only safeguard the robot from harming users (and/or the environment), but also prevent robots from damaging themselves.

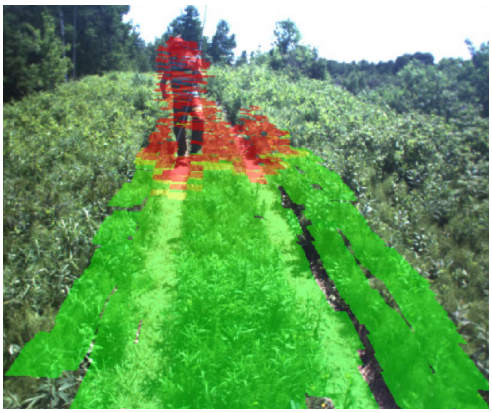


Figure 6.3: Example of the place-dependent terrain-assessment algorithm correctly classifying a human in the path as unsafe. Figure reproduced from [43].

VT&R is incorporated with a place-dependent, learning-based terrain assessment module for obstruction detection [8] [6] [7]. The MEL algorithm and the STPG data structure allows the terrain assessment module to gather information from past experiences and then train a place-dependent classifier to detect terrain changes. Specifically, a classifier will be trained at every point on the path to provide more accurate result than a general classifier (trained on all the data) at assessing the terrain in that place. If the geometric terrain change at a place is greater than a chosen threshold, then that place will be marked as unsafe to travel (being blocked). The terrain assessment module will then signal the robot to pause until the terrain becomes safe to travel again. A qualitative example of the algorithm is illustrated in Figure 6.3, in which a human blocks the path on the vegetation-rich ridge area. The green patches in the figure identify unchanged terrain that is safe to travel. The red patches identify geometric changes in the path. In this example, the human is identified and the path is marked as unsafe (blocked).

The terrain assessment module will resume the robot to continue the path traversal once the obstacle is cleared and the terrain is marked to be safe again. To integrate with the RCTP

planner, a threshold on the robot pause time can be imposed, such that the planner will signal untraversable observation and change route when the pause time expires. Moreover, the terrain assessment module could also give a measure of terrain roughness, which can be further incorporated into edge cost assignment.

6.2.2 Loss Handling

During the repeat phase of the VT&R algorithm, the robot attempts to localize the live view to a vertex in the privileged path. When the localization is successful, the system will propagate the position estimate and send it to a model-predictive-control path tracker [38]. If the localization fails, the robot will still rely on Visual Odometry (VO) to keep going and tries to reconnect to the privileged path up to a certain distance. This mechanism can then be integrated with the RCTP planner, such that the planner will signal untraversable observation and change route when the VO distance threshold is exceeded.

However, in order to change route or switch plan, the robot still needs to recover from its lost status to get localized again. To achieve localization recovery, we propose to add a safe return module into the current design of the VT&R system. The safe return module will be activated in the background when the robot signals localization failures and starts to rely on VO. A fake teach (privileged) experience will then be recorded by the safe return module to memorize the path that the robot is driving with VO. If the robot gets re-localized again before the VO distance threshold is exceeded, the fake teach experience will be discarded. Otherwise, the robot will use this fake teach experience to return to the last localization point when the robot is lost.

Figure 6.4 depicts the proposed system design to integrate the RCTP planner into the VT&R system. A typical system flow is as follows: when the robot enters the repeat phase from idling, it will first topological localize against the STPG to find its starting position on the map. Then the RCTP planner will compute the policy to traverse to the destination. The robot achieves the autonomous route following through continuous metric localization until the goal is reached. If at some point the robot is obstructed or gets lost for a certain period of time, it will then make untraversable observations and look up the policy to adjust its route. The safe return module

will be used to help the robot to recover to the last localized position if the robot is lost. The system will report task failure if no plan can be computed to reach the goal, or the robot fails the safe return.

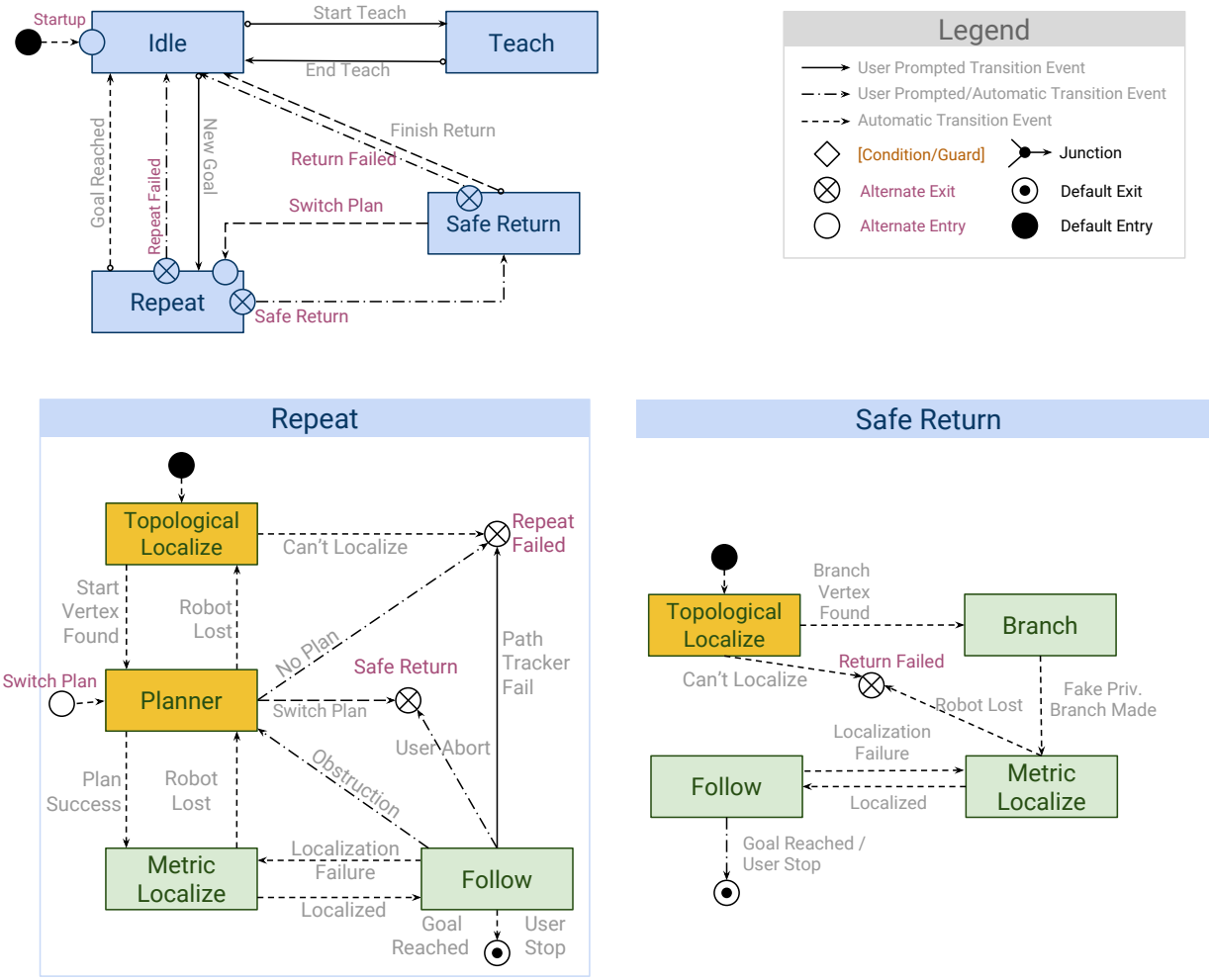


Figure 6.4: The proposed VT&R system design to integrate the Robust Canadian Traveler Planner. A safe return module is added help recover from robot getting lost.

6.3 Summary

In this chapter, we discussed some approaches to integrate the RCTP planner into VT&R - a vision-based robot navigation system. We propose to gather and analyze the data on robot traversal histories to assign blocking (failure) probabilities of path segments and then use this to compute travel policy. We then described the terrain assessment module that VT&R uses to

detect path blockage and the mechanisms to detect robot getting lost. Finally, we proposed a modified system design of VT&R to integrate the RCTP planner. A safe return module is also added to help recover from robot getting lost.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we extended the Canadian Traveler Problem (CTP) into what we call the Robust Canadian Traveler Problem (RCTP) that tries to find the traversal policy over uncertain maps with a good balance of both mean and variation of the traversal cost. The robustness was achieved by searching for the policy that has the minimum (weighted) exponential risk instead of the minimum expected cost. The exponential risk can be thought of as approximately variance-adjusted mean, but with the extra benefit of penalizing deviations above the mean more than deviations below. Unlike variance-adjusted mean, the exponential risk also maintains Bellman's principle of optimality, and thus permitting existing graph-search methods to be used for solution.

In order to solve RCTPs, we first showed how to represent RCTPs as AND/OR trees. And then two versions of the modified AO* algorithm are provided to search for the desired policy in the AND/OR tree representation: an offline version that is both optimal and complete, namely an optimal solution is guaranteed to be found if one exists; and an online version with a limited search depth to provide approximate solutions to the offline algorithm, but with significantly less computational time.

Simulations were conducted to show the benefit of our framework by comparing with the classic CTP solutions and another baseline approach that uses Dijkstra's algorithm with replanning. Specifically, the RCTP framework is able to search for sub-optimal (in terms of expected

cost) policy alternatives with significantly lower worst-case cost and computational time comparing against the optimal policy, but with little sacrifice on the expected cost. The simulation results also validate that the online algorithm is able to approximate the behavior of the offline algorithm by imposing a limit on the search depth, but with exponentially faster responding time.

Finally, we discussed the necessary map preparation steps and system requirements to integrate the RCTP planner into real-world robot navigation systems. Specifically, the map should be prepared with each path being analyzed and assigned a probability of being blocked. The robot should have the ability to observe path blockage, detect loss of localization, and recover from the loss of localization. As a demonstration, we used the VT&R system as an example, and proposed a modified system design to integrate with the RCTP planner.

In summary, the novel contributions of this thesis are as follows. First, we extended the Canadian Traveler Problem to the Robust Canadian Traveler Problem that considers not only the mean, but also the variation of the policy cost while searching for the traversal policy. Second, we proposed to use the exponential risk to evaluate policies and provided an offline as well as an online algorithm to solve the RCTP. Third, we validated the proposed algorithms through experiments on simulated maps, as well as a map that was collected from a robot field trial. The algorithms developed and the results acquired were submitted to the 2019 International Conference on Robotics and Automation (ICRA).

7.2 Future Work

There are several directions in which to extend our current work. Firstly, as with the classic stochastic CTPs, we assume the blocking probabilities of the stochastic paths are known. However, such information is often unavailable, or only noisily available in real-world scenarios; the problem can be further extended to quantify the blocking probability with uncertainties, and update its belief through Bayesian updates (for example, using Dirichlet processes). Ross et. al [46] showed an interesting research direction of doing Bayesian reinforcement learning on the Partially-Observed Markov Decision Process (POMDP). Specifically, they proposed a mathematical model called Bayes-Adaptive POMDP that allows the agent to update do-

main knowledge from interactions with the environment. Moreover, the model can plan for sequences of actions with the trade-off between improving the model and gathering reward. Since CTPs (and RCTPs) can be converted into POMDPs [2], similar approaches might be useful to enhance our framework.

Secondly, we assume the blocking probabilities are independent of each other, which is often not true in practice – spatially adjacent places often have similar properties, and other properties are temporally correlated (e.g., lighting effects). Thus the correlations between different paths could be properly studied as well in order to provide more robust plans. Dey et al. [10] proposed the Gaussian Traveler Problem (GTP), in which a belief over edge costs is modeled by the Gaussian Process (GP). The agent will observe local edge costs from online traversal and then adjust adjacent edge costs through GP updates. Similar approaches might also be combined with our framework to provide online adjustment of spatially-adjacent edges' blocking probabilities.

Thirdly, our framework assumes the robot can observe the traversability status of the stochastic paths at no cost. However, this is not always true in practice. For example in our proposed VT&R design, when the robot sees obstacles, it will wait for a certain amount of time to make sure the obstacle will not go away by itself; when the robot is lost, it also needs some recovery time to get back to the last localized position. Thus our framework can also be enhanced to take the cost of observation into consideration when computing traversal policies.

Bibliography

[1] Vural Aksakalli. The BAO* algorithm for stochastic shortest path problems with dynamic learning. In *Decision and Control, 2007 46th IEEE Conference on*, pages 6003–6008. IEEE, 2007.

[2] Vural Aksakalli, O Furkan Sahin, and Ibrahim Ari. An AO* based exact algorithm for the Canadian traveler problem. *INFORMS Journal on Computing*, 28(1):96–111, 2016.

[3] Amotz Bar-Noy and Baruch Schieber. The Canadian traveller problem. In *SODA*, volume 91, pages 261–270, 1991.

[4] Richard Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958.

[5] Marco Bender and Stephan Westphal. An optimal randomized online algorithm for the k-Canadian traveller problem on node-disjoint paths. *Journal of Combinatorial Optimization*, 30(1):87–96, 2015.

[6] Laszlo-Peter Berczi and Timothy D Barfoot. It's like déjà vu all over again: Learning place-dependent terrain assessment for visual teach and repeat. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 3973–3980. IEEE, 2016.

[7] Laszlo-Peter Berczi and Timothy D Barfoot. Looking high and low: Learning place-dependent Gaussian mixture height models for terrain assessment. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 3918–3925. IEEE, 2017.

[8] Laszlo-Peter Berczi, Ingmar Posner, and Timothy D Barfoot. Learning to assess terrain from human demonstration using an introspective Gaussian-process classifier. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3178–3185. IEEE, 2015.

[9] Zahy Bnaya, Ariel Felner, and Solomon Eyal Shimony. Canadian traveler problem with remote sensing. In *IJCAI*, pages 437–442, 2009.

[10] Debadeepta Dey, Andrey Kolobov, Rich Caruana, Ece Kamar, Eric Horvitz, and Ashish Kapoor. Gauss meets Canadian traveler: shortest-path problems with correlated natural dynamics. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 1101–1108. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[11] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[12] Patrick Eyerich, Thomas Keller, and Malte Helmert. High-quality policies for the Canadian traveler's problem. In *Third Annual Symposium on Combinatorial Search*, 2010.

[13] Dave Ferguson, Anthony Stentz, and Sebastian Thrun. PAO* for planning with hidden state. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 3, pages 2840–2847. IEEE, 2004.

[14] Seyedshams Feyzabadi and Stefano Carpin. Risk-aware path planning using hirerachical constrained Markov decision processes. In *Automation Science and Engineering (CASE), 2014 IEEE International Conference on*, pages 297–303. IEEE, 2014.

[15] Robert W Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.

[16] Lester R Ford Jr. Network flow theory. Technical report, RAND CORP SANTA MONICA CA, 1956.

[17] Dror Fried, Solomon Eyal Shimony, Amit Benbassat, and Cenny Wenner. Complexity of Canadian traveler problem variants. *Theoretical Computer Science*, 487:1–16, 2013.

[18] Paul Furgale and Timothy D Barfoot. Visual teach and repeat for long-range rover autonomy. *Journal of Field Robotics*, 27(5):534–560, 2010.

[19] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. *arXiv preprint arXiv:1404.2334*, 2014.

[20] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[21] Ronald A Howard and James E Matheson. Risk-sensitive Markov decision processes. *Management science*, 18(7):356–369, 1972.

[22] Viorela Ila, Josep M Porta, and Juan Andrade-Cetto. Information-based compact pose SLAM. *IEEE Transactions on Robotics*, 26(1):78–93, 2010.

[23] Sertac Karaman and Emilio Frazzoli. Incremental sampling-based algorithms for optimal motion planning. *Robotics Science and Systems VI*, 104:2, 2010.

[24] Sven Koenig and Maxim Likhachev. Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics*, 21(3):354–363, 2005.

[25] Sven Koenig, Maxim Likhachev, and David Furcy. Lifelong planning A*. *Artificial Intelligence*, 155(1-2):93–146, 2004.

[26] Atul Kumar, Veeraruna Kavitha, and N Hemachandra. Finite horizon risk sensitive MDP and linear programming. In *Decision and Control (CDC), 2015 IEEE 54th Annual Conference on*, pages 7826–7831. IEEE, 2015.

[27] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[28] Maxim Likhachev, Geoffrey J Gordon, and Sebastian Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in neural information processing systems*, pages 767–774, 2004.

[29] Zhan Wei Lim, David Hsu, and Wee Sun Lee. Shortest path under uncertainty: Exploration versus exploitation. In *UAI*, 2017.

[30] Kirk MacTavish, Michael Paton, and Timothy D Barfoot. Visual triage: A bag-of-words experience selector for long-term visual route following. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 2065–2072. IEEE, 2017.

[31] Shie Mannor and John Tsitsiklis. Mean-variance optimization in Markov decision processes. *arXiv preprint arXiv:1104.5601*, 2011.

[32] Shie Mannor and John N Tsitsiklis. Algorithmic aspects of mean–variance optimization in Markov decision processes. *European Journal of Operational Research*, 231(3):645–653, 2013.

[33] Colin McManus, Paul Furgale, Braden Stenning, and Timothy D Barfoot. Lighting-invariant visual teach and repeat using appearance-based lidar. *Journal of Field Robotics*, 30(2):254–287, 2013.

[34] Edward F Moore. The shortest path through a maze. In *Proc. Int. Symp. Switching Theory, 1959*, pages 285–292, 1959.

[35] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN 0262018020, 9780262018029.

[36] Marek Musiela and Thaleia Zariphopoulou. Indifference prices and related measures. *preprint*, 2001.

[37] Nils J Nilsson. *Principles of artificial intelligence*. Morgan Kaufmann, 2014.

[38] Chris J Ostafew, Angela P Schoellig, Timothy D Barfoot, and Jack Collier. Learning-based nonlinear model predictive control to improve vision-based mobile robot path tracking. *Journal of Field Robotics*, 33(1):133–152, 2016.

[39] Christos H Papadimitriou and Mihalis Yannakakis. Shortest paths without a map. In *International Colloquium on Automata, Languages, and Programming*, pages 610–620. Springer, 1989.

[40] Michael Paton, Kirk MacTavish, Chris J Ostafew, and Timothy D Barfoot. It's not easy seeing green: Lighting-resistant stereo visual teach & repeat using color-constant images. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1519–1526. IEEE, 2015.

[41] Michael Paton, Kirk MacTavish, Michael Warren, and Timothy D Barfoot. Bridging the appearance gap: Multi-experience localization for long-term visual teach and repeat. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 1918–1925. IEEE, 2016.

[42] Michael Paton, François Pomerleau, Kirk MacTavish, Chris J Ostafew, and Timothy D Barfoot. Expanding the limits of vision-based localization for long-term route-following autonomy. *Journal of Field Robotics*, 34(1):98–122, 2017.

[43] Michael Paton, Kirk MacTavish, Laszlo-Peter Berczi, Sebastian Kai van Es, and Timothy D Barfoot. I can see for miles and miles: An extended field test of visual teach and repeat 2.0. In *Field and Service Robotics*, pages 415–431. Springer, 2018.

[44] Judea Pearl. Heuristics: intelligent search strategies for computer problem solving. 1984.

[45] Samuel Prentice and Nicholas Roy. The belief roadmap: Efficient planning in belief space by factoring the covariance. *The International Journal of Robotics Research*, 28(11-12): 1448–1465, 2009.

[46] Stephane Ross, Brahim Chaib-draa, and Joelle Pineau. Bayes-adaptive POMDPs. In *Advances in neural information processing systems*, pages 1225–1232, 2008.

[47] Braden Stenning and Timothy D Barfoot. Path planning on a network of paths. In *Aerospace Conference, 2011 IEEE*, pages 1–12. IEEE, 2011.

[48] Braden E Stenning, Colin McManus, and Timothy D Barfoot. Planning using a network of reusable paths: A physical embodiment of a rapidly exploring random tree. *Journal of Field Robotics*, 30(6):916–950, 2013.

[49] Anthony Stentz. Optimal and efficient path planning for partially-known environments. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 3310–3317. IEEE, 1994.

[50] Anthony Stentz et al. The focussed D* algorithm for real-time replanning. In *IJCAI*, volume 95, pages 1652–1659, 1995.

[51] Rafael Valencia and Juan Andrade-Cetto. Path planning in belief space with pose SLAM. In *Mapping, Planning and Exploration with Pose SLAM*, pages 53–87. Springer, 2018.

[52] Kevin Wang. Pathfinder. https://github.com/kevinwang1975/PathFinder, 2016.

[53] Stephen Warshall. A theorem on boolean matrices. *Journal of the ACM (JACM)*, 9(1): 11–12, 1962.

[54] Virginia R Young and Thaleia Zariphopoulou. Pricing dynamic insurance risks using the principle of equivalent utility. *Scandinavian Actuarial Journal*, 2002(4):246–279, 2002.