

AO* and Penalty Based Algorithms for the Canadian Traveler Problem

A thesis submitted to the
Graduate School of Natural and Applied Sciences

by

Ömer Furkan ŞAHİN

in partial fulfillment for the
degree of Master of Science

in
Industrial and Systems Engineering



This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science in Industrial and Systems Engineering.

APPROVED BY:

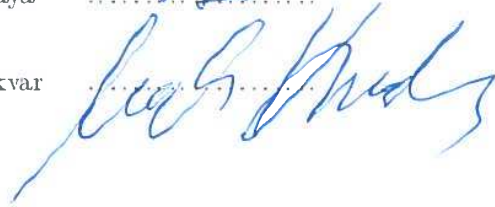
Assoc. Prof. Vural Aksakallı
(Thesis Advisor)



Asst. Prof. Ali Fuat Alkaya



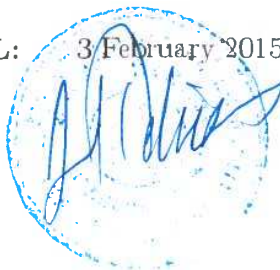
Asst. Prof. Murat Küçükvar



This is to confirm that this thesis complies with all the standards set by the Graduate School of Natural and Applied Sciences of İstanbul Şehir University:

DATE OF APPROVAL: 3 February 2015

SEAL/SIGNATURE:

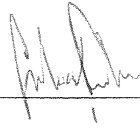


Declaration of Authorship

I, Ömer Furkan ŞAHİN, declare that this thesis titled, “AO* and Penalty Based Algorithms for the Canadian Traveler Problem” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:



Date:

03.02.2015

AO* and Penalty Based Algorithms for the Canadian Traveler Problem

Ömer Furkan ŞAHİN

Abstract

The Canadian Traveler Problem (CTP) is a challenging path planning problem on stochastic graphs where some edges are blocked with certain probabilities and status of edges can be disambiguated only upon reaching an end vertex. The goal is to devise a traversal policy that results in the shortest expected traversal length between a given starting vertex and a termination vertex.

The organization of this thesis is as follows: In the first chapter we define CTP and its variant SOSP and present an extensive literature review related to these problems. In the second chapter, we introduce an optimal algorithm for the problem, based on an MDP formulation which is a new improvement on AO* search that takes advantage of the special problem structure in CTP. The new algorithm is called CAO*, which stands for AO* with Caching. CAO* uses a caching mechanism and makes use of admissible upper bounds for dynamic state-space pruning. CAO* is not polynomial-time, but it can dramatically shorten the execution time needed to find an exact solution for moderately sized instances. We present computational experiments on a realistic variant of the problem involving an actual maritime minefield data set.

In the third chapter, we introduce a simple, yet fast and effective penalty-based heuristic for CTP that can be used in an online fashion. We present computational experiments involving real-world and synthetic data that suggest our algorithm finds near-optimal policies in very short execution times.

Another efficient method for sub-optimally solving CTP, rollout-based algorithms, have also been shown to provide high quality policies for CTP. In the final chapter, we compare the two algorithmic frameworks via computational experiments involving Delaunay and grid graphs using one specific penalty-based algorithm and four rollout-based algorithms. Our results indicate that the penalty-based algorithm executes several orders of magnitude faster than rollout-based ones while also providing better policies, suggesting that penalty-based algorithms stand as a prominent candidate for fast and efficient sub-optimal solution of CTP.

Keywords: Probabilistic path planning, Canadian traveler problem, Markov decision process, AO* search

Kanadalı Gezgin Problemi İin AO* ve Ceza Tabanlı Algoritmalar

Ömer Furkan ŞAHİN

ÖZ

Kanadalı Gezgin Problemi (KGP), stokastik graflarda, bazı kenarların belli bir olasılığa göre kapalı veya açık olabildiğı ve bu kenarların ancak komşu noktalarının ziyaret edilmesi suretiyle geçilebilirliklerinin tespit edilebildiğı, zorlu bir güzergah planlama problemidir. Bu problemde hedef, belirli bir başlangıç ve bitiş noktası arasındaki en kısa beklenen gezinme uzunluğunu veren gezinme planını bulmaktır.

Bu tezin organizasyonu şu şekildedir: Birinci bölümde, CTP ve SOSP'nin formülasyonları ve bu problemleri konu alan geniş bir literatür taraması sunulacaktır. İkinci bölümde, mevcut AO* arama algoritmasına, KGP'nin problem yapısından faydalanmaya olanak tanıyacak iyileştirmeler yapılarak elde ettiğimiz, MDP tabanlı bir optimal algoritma tanıtılacaktır. Bu yeni algoritma, CAO*, önbelleklemeli AO* (AO* with caching) olarak adlandırılmıştır. CAO*, daha önce ziyaret edilmiş durumların her seferinde yeniden genişletilmesinin önüne geçen önbellekleme mekanizması ve durum-uzayını dinamik olarak budamaya olanak tanıyan kabul edilebilir alt sınırlar kullanması olmak üzere iki önemli özelliğe sahiptir. CAO* polinom zamanlı değildir, ancak bu özellikleri sayesinde orta ölçekli problemler için optimal sonuçlar bulmada çözüm süresini ciddi ölçüde kısaltmaktadır. Son olarak, bu bölümde gerçek, mayınlı deniz alanı verileri kullanılarak hazırlanmış bilgisayar simülasyonları sunulacaktır.

Üçüncü bölümde, KGP için, çevrimiçi uygulanabilir, basit, fakat hızlı ve etkili bir ceza-tabanlı sezgisel tanıtılacaktır. Ardından bu sezgiselin optimale çok yakın çözümler verdiğini gösteren bilgisayar simülasyonları sunulacaktır.

KGP'nin suboptimal çözümünde bir diğer etkili yöntem olan, örnekleme tabanlı algoritmaların, KGP için yüksek kaliteli çözümler ürettiğini gösteren bir çalışma literatürde mevcuttur. Son bölümde, bu iki algoritmik çatının Delaunay ve grid graflar üzerinde, bir adet ceza-tabanlı ve dört adet örnekleme tabanlı algoritma kullanılarak bilgisayar simülasyonları üzerinde karşılaştırması yapılacaktır. Karşılaştırmalarımızda ceza tabanlı algoritmamızın, hem çözüm hızı hem de çözüm kalitesi açısından rollout tabanlı algoritmalara üstünlük sağlamış olması, ceza tabanlı algoritmaların, KGP'nin suboptimal çözümünde hızlı ve efektif bir aday olabileceğini göstermektedir.

Anahtar Sözcükler: Olasılıksal güzergah planlama, Kanadalı Gezgin Problemi, Markov karar süreçleri, AO* arama

Acknowledgments

I would like to express my most sincere gratitude and appreciation to my advisor Assoc. Prof. Vural Aksakallı for fostering and fueling my curiosity in science and introducing me to scientific research. I thank him for his guidance, encouragement and patience throughout my study. Without his support this thesis would not be possible.

Special thanks to my committee members, Asst. Prof. Ali Fuat Alkaya and Asst. Prof. Murat Küçükvar, for their kindness and helpful insights and suggestions.

I would also like to thank Prof. Erkan Türe for the valuable experience I had as a teaching assistant under his supervision. I learned so much from him and I am thankful for everything he has done for me.

I am extremely grateful to İbrahim Arı for his advice and assistance. He was always willing to help and give helpful insights. I would also like to thank Patrick Eyerich, Thomas Keller, and Malte Helmert for sharing their computer code for the rollout-based algorithms.

Portions of Chapter 1 and Chapter 2 were presented by Dr. Vural Aksakallı in IEEE Conference on Decision and Control in New Orleans in 2007 and a joint work of ours on these chapters is currently under minor revision in INFORMS Journal on Computing under the title “*An AO* Based Exact Algorithm for the Canadian Traveler Problem*”. The study in Chapter 3 was presented in ICAPS 2014 Planning and Robotics Workshop, New Hampshire, USA with the title “*A Fast and Effective Online Algorithm for the Canadian Traveler Problem*”. The research in Chapter 4 was published in the International Journal of Machine Learning and Computing under the title “*A Comparison of Penalty and Rollout-Based Algorithms for the Canadian Traveler Problem*”, Vol. 5(4), pp. 319–324.

Finally, I wish to thank my parents and my sister for their support and encouragement throughout my study.

Contents

Declaration of Authorship	ii
Abstract	iii
Öz	iv
Acknowledgments	v
List of Figures	viii
List of Tables	ix
Abbreviations	x
1 Introduction	1
1.1 Overview	1
1.2 The Canadian Traveler Problem	1
1.2.1 The Discrete Stochastic Obstacle Scene Problem	2
1.3 Literature Review	3
1.4 Organization of the Thesis	4
2 An AO* Based Exact Algorithm for the Canadian Traveler Problem	5
2.1 Introduction	5
2.2 MDP and POMDP Formulations	6
2.2.1 MDP Formulation and The Bellman Equation	7
2.2.2 Deterministic POMDP Formulation	9
2.3 The CAO* Algorithm	11
2.3.1 AO Trees	11
2.3.2 The AO* Algorithm	14
2.3.3 The CAO* Algorithm	16
2.4 Computational Experiments	19
2.4.1 The BAO* and PAO* Algorithms	19
2.4.2 Experimental Setup	21
2.4.3 Simulation Environment A	21
2.4.4 Simulation Environment B	22
2.4.5 Simulation Environment C	24
2.4.6 Simulation Environment D	25
2.5 Summary and Conclusions	26

3	A Fast and Effective Online Algorithm for the Canadian Traveler Problem	29
3.1	Introduction	29
3.2	The DT Algorithm	30
3.3	Computational Experiments	32
3.3.1	Environment 1	32
3.3.2	Environment 2	34
3.4	Conclusions and Future Research	34
3.4.1	Conclusions	34
3.4.2	Limitations and Future Research	35
4	A Comparison of Penalty and Rollout-Based Policies for the Canadian Traveler Problem	36
4.1	Introduction	36
4.2	Algorithms for CTP	37
4.2.1	Optimism (OMT)	37
4.2.2	Hindsight Optimization (HOP)	38
4.2.3	Optimistic Rollout (ORO)	39
4.2.4	Blind UCT (UCTB)	39
4.2.5	Optimistic UCT (UCTO)	40
4.3	Computational Experiments	41
4.3.1	Delaunay Graph Results	43
4.3.2	Grid Graph Results	45
4.4	Conclusions and Future Research	46
4.4.1	Conclusions	46
4.4.2	Limitations and Future Research	46
	A Problem Instances in Simulation Environments C and D	48
	Bibliography	51

List of Figures

2.1	A simple CTP instance with two stochastic edges (denoted by dashed lines) and three deterministic edges (denoted by solid lines). Length of each edge is given above the edge. Shown in parantheses are the marks of stochastic edges.	13
2.2	The complete AO tree corresponding to the CTP instance shown in Figure 2.1. OR nodes are depicted by squares and AND nodes by circles. . .	14
2.3	The solution tree (shown horizontally) corresponding to the CTP instance shown in Figure 2.1.	16
2.4	A CTP instance with three stochastic and four deterministic edges. . . .	17
2.5	The partial AO tree corresponding to the CTP instance shown in Figure 2.4 illustrating the state overlap phenomena. The state $(y_3, ("U", "U", "A"))$ can be reached at from two different paths.	18
2.6	Experimental Delaunay and grid graph realizations in Simulation Environment A. Stochastic edges are shown in bold.	22
2.7	An experimental realization in Simulation Environment B with $N = 10$. Stochastic edges, i.e., grid edges intersecting disks are shown in bold. . . .	23
2.8	Illustration of the COBRA data where gray intensity reflects marks of disk with darker tones indicating a higher mark. Grid edges are not shown for clarity.	25
2.9	Six COBRA-like D-SOSP instances used in Simulation Environment D. . .	28
3.1	Illustration of COBRA data set. The gray intensity of disks reflect probability of the disks being true obstacles.	33
4.1	A Delaunay graph consisting of 20 nodes and 48 edges.	41
4.2	A CTP instance on a 10x10 grid graph. Blocked edges are represented by bold edges.	41
4.3	Blockage probability density plots for $\lambda = 2$ for the Beta distribution. . . .	42
4.4	Blockage probability density plots for $\lambda = 3$ for the Beta distribution. . . .	42
4.5	Average run time as a function of size on Delaunay graphs.	45

List of Tables

2.1	Run time comparison of AO*, BAO*, and CAO* in Environment A in seconds. In the table, "-" stands for insufficient memory whereas "*" stands for non-convergence to optimality within 5 hours.	23
2.2	Run time comparison of VI, AO*, BAO*, and CAO* in Environment B in seconds. In the table, "-" stands for insufficient memory.	24
2.3	Performance of CAO* on the COBRA data set (Simulation Environment B). The columns denote the run time, number of expanded/ cached/ re-visited/ pruned nodes respectively.	25
2.4	CAO* performance averaged over the six COBRA-like instances in Simulation Environment C.	26
3.1	Performance of the DT Algorithm on COBRA data set for the K, c combinations listed.	33
3.2	Average performance of the DT Algorithm on six COBRA-like data sets for the K, c combinations listed.	35
4.1	Average cost of algorithms for Delaunay graphs. Lowest policy cost for each parameter combination is denoted in bold. Last two columns show how much DT is better than OMT and UCTO respectively in percentages.	44
4.2	Average cost of algorithms for grid-based graphs. Lowest algorithms cost for each parameter combination is denoted in bold. Last two columns show how much DT is better than OMT and UCTO respectively in percentages.	44
4.3	Run time averages (in seconds) of algorithms on Delaunay graphs.	45
4.4	Run time averages (in seconds) of algorithms on grid-based graphs.	45
A.1	Center coordinates and marks of COBRA disks.	48
A.2	Center coordinates and marks of COBRA-like Instance 1 disks.	49
A.3	Center coordinates and marks of COBRA-like Instance 2 disks.	49
A.4	Center coordinates and marks of COBRA-like Instance 3 disks.	49
A.5	Center coordinates and marks of COBRA-like Instance 4 disks.	50
A.6	Center coordinates and marks of COBRA-like Instance 5 disks.	50
A.7	Center coordinates and marks of COBRA-like Instance 6 disks.	50

Abbreviations

AO Tree	AND/OR Tree
BAO*	AO* (with) Bounds
CAO*	AO* (with) Caching
COBRA	Coastal Battlefield Reconnaissance (and) Analysis
CTP	Canadian Traveler Problem
D-SOSP	Discretized Stochastic Obstacle Scene Problem
DT	Distance-(to)-Termination
DTA	Distance-(to)-Termination Algorithm
HOP	Hindsight Optimization
MDP	Markov Decision Process
NDR	Navigate-Disambiguate-Repeat
OMT	Optimism
ORO	Optimistic Rollout
PAO*	Propagating AO*
PBA	Penalty-Based Algorithms
POMDP	Partially Observable MDP
RBA	Rollout-Based Algorithms
RDA	Reset Disambiguation Algorithm
SDP	Stochastic Dynamic Programming
SOSP	Stochastic Obstacle Scene Problem
SRA	Simulated Risk (Disambiguation) Algorithm
UCT	Upper Confidence (Bounds) (Applied) (to) Trees
UCTB	Blind UCT
UCTO	Optimistic UCT
VI	Value Iteration

Chapter 1

Introduction

1.1 Overview

In this thesis, a stochastic shortest path problem in the presence of a dynamic learning capability is considered. In a specific variant of this problem, a spatial arrangement of obstacles needs to be traversed and the actual status of these obstacles can be disambiguated during the traversal. This problem has important practical applications in path planning in partially known environments such as robot navigation [1, 2], adaptive transportation systems [3–5], and minefield countermeasures [6–8]. Several exact algorithms and heuristics for this problem are available in the literature [9–14]. In this chapter, we provide formulation for the problem and an extensive literature review.

1.2 The Canadian Traveler Problem

CTP can formally be defined as follows: Let $G = (V, E)$ be an undirected graph with designated vertices $s, t \in V$, and suppose there is a function $\ell : E \rightarrow \mathbb{R}_{\geq 0}$ assigning a length to each edge; the goal here is to find a shortest s, t traversal (walk) in G . However, not all of the edges may indeed be traversable. In particular, for a given subset $E' \subseteq E$ of edges called *stochastic edges*, there is a function $\rho : E' \rightarrow [0, 1)$ such that, for each edge $e \in E'$, $\rho(e)$, called the *mark* of edge e , is the probability that e is not traversable independent of the other edges. The edges in $E \setminus E'$ are called *deterministic edges* and they are known a priori to be traversable. For any edge $e \in E'$, when the

traversal is at an endpoint of e , the agent has the option to disambiguate e , that is, learn whether e is traversable. Edges cannot be traversed until they are known to be traversable, and the traversability status of each edge is static and will never change over the course of the traversal. To avoid infinite expected length, we assume the existence of a (possibly very long) s, t path consisting of deterministic edges. Of course, if the agent follows any particular policy then the traversal is still random and will unfold depending on the results of the disambiguations. Thus, the traversal's distribution is specified through ρ . The agent's goal is to find an optimal policy in the sense of having shortest expected length. Finding such an optimal policy is the *Canadian Traveler Problem (CTP)*. Without loss of generality, we additionally assume that there is a limit $K \leq N$ on the number of available disambiguations where $N := |E'|$.

1.2.1 The Discrete Stochastic Obstacle Scene Problem

The Stochastic Obstacle Scene Problem (SOSP) is inherently a continuous-space path planning problem. In continuous-space SOSP, an agent wishes to navigate from one given location to another through an arrangement of arbitrarily shaped regions in an obstacle field which are possibly obstacles. At the outset, the agent is given the respective probabilities that the regions are truly obstacles, called the *mark* of the region. Only when situated on a region's boundary, the agent has the option to disambiguate the region, i.e., learn if the region is truly an obstacle. The goal here is to find a policy that decides what and where to disambiguate en route so as to minimize the expected length of the traversal. Without loss of generality, we assume disk-shaped obstacles with the same radii.

In D-SOSP, we consider a discrete approximation of SOSP which is, for simplicity and convenience, a grid graph G on $[1, 1] \times [i_{\max}, j_{\max}]$ with diagonal edges where i_{\max} and j_{\max} are given integers. Lengths of diagonal edges are taken as $\sqrt{2}$ and 1 for non-diagonal edges. One vertex in G is designated as the starting point s , another vertex is designated as the termination point t , and the agent is to walk from s to t in G , only traversing edges that do not intersect any untraversable or ambiguous disks. If an edge intersects any ambiguous disk, then a disambiguation of the disk may be performed at the endpoint that is outside of the disk. As before, the goal is to develop a policy that minimizes the expected length of the traversal by effective exploitation of the disambiguation capability.

1.3 Literature Review

Introduced by Papadimitriou and Yannakakis (1991), the Canadian Traveler Problem (CTP) is a challenging probabilistic path planning problem that has recently received considerable attention. CTP is concerned with an agent traversing a graph where some edges are blocked with certain probabilities and the status of these edges can be disambiguated dynamically upon reaching an incident vertex. Given a starting vertex s and a termination vertex t , the goal is to devise a policy that results in the shortest s, t walk in an expected sense. CTP has been shown to be PSPACE-Complete [16], suggesting that not only its computational complexity is intractable, but its space complexity is intractable as well.

Despite its computational difficulty, CTP stands as an important problem from both practical and theoretical viewpoints. First, CTP and closely related problems has important practical applications in path planning in partially known environments such as robot navigation [1, 2], adaptive transportation systems [3–5], and minefield countermeasures [6–8]. Second, from a theoretical point of view, CTP has rather interesting characteristics in the sense that it can be cast both as a Markov Decision Process (MDP) with exponentially many states, or as a Partially Observable MDP (POMDP) with deterministic observations. In this regard, CTP belongs to an intermediate class of problems called Deterministic POMDPs [14], which allow for state uncertainty but avoid the inherent complexity of noisy observations [17].

Regarding variants of CTP, of particular interest is the Discrete Stochastic Obstacle Scene Problem (D-SOSP) that has practical path planning applications in naval minefields [18–21]. D-SOSP is essentially a variant of CTP on grid graphs with probabilistic dependency among groups of edges. Specifically, this problem is a grid graph discretization of continuous-space SOSP wherein an agent needs to swiftly navigate from one given location to another through an arrangement of arbitrarily shaped regions in an obstacle field which are possible obstacles. At the outset, the agent is given the respective probabilities that the regions are truly obstacles and, only when situated on a region’s boundary, the agent has the option to disambiguate the region, i.e., learn at a cost if the region is truly an obstacle. The goal here is to find a policy that decides what and where to disambiguate en route so as to minimize the expected length of the traversal. Several heuristics and approximation algorithms have been introduced for CTP in

the literature [9–11] and optimal algorithms for certain special cases of CTP have been proposed [12–14].

1.4 Organization of the Thesis

The rest of the thesis is organized as follows:

Chapter 2 casts CTP as a finite-horizon MDP and a Deterministic POMDP and provides the corresponding Bellman equation. Then, an exact algorithm called CAO* Algorithm is introduced for CTP following up with the computational experiments to evaluate the performance of the exact algorithm.

In Chapter 3, a heuristic framework for CTP, called Penalty-Based Algorithms is presented. DT Algorithm, which belongs to Penalty-Based Algorithms framework, is tested on two different simulation environments, using CAO* as the benchmark. Computational experiments showed that this heuristic provides near-optimal results in very short execution times.

Chapter 4 provides a set of computational experiments to compare the penalty-based DT Algorithm against rollout-based algorithms for CTP on several different simulation environments. Our results indicate that DTA runs significantly faster than rollout-based algorithms while providing better policies.

Chapter 2

An AO* Based Exact Algorithm for the Canadian Traveler Problem

2.1 Introduction

In this chapter, the following contributions are presented:

1. We present explicit formulations of CTP as a finite-horizon MDP as well as a Deterministic POMDP.
2. Based on the MDP formulation, we introduce an optimal algorithm for CTP, which is a new improvement on AO* search that makes two key improvements by utilizing the special problem structure of CTP: (1) it employs a state caching mechanism to avoid re-expansion of previously visited states (hence, the solution structure it maintains is a graph and not a tree), and (2) it prunes the solution graph using dynamic upper bounds (in addition to lower bounds as in standard AO*) during both node expansion and cost propagation. We call our optimal algorithm CAO*, which stands for AO* with Caching.
3. We present computational experiments comparing CAO* to AO*, value iteration, and two other state-of-the-art algorithms on general Delaunay and grid graph-based CTP instances as well as the D-SOSP variant of CTP. Our choice of D-SOSP for computational experiments is that we believe D-SOSP is perhaps one of the most realistic variants of CTP in the literature.

CAO* is not polynomial-time, yet it is still relevant for the following reasons:

1. As illustrated in our computational experiments, it can drastically shorten the execution time needed to find an exact solution to realistic instances of the D-SOSP variant. In fact, our experiments indicate that CAO* can provide close to a 800-fold increase in run time compared to value iteration, and about 1,800-fold increase against classical AO*. Value iteration did not even run in some of our experiments due to its excessive memory requirements.
2. It can be used to benchmark performance of heuristic algorithms for general CTP against the optimal solution on reasonably sized instances.
3. It can potentially be used in conjunction with approximation schemes for the problem, both within the stochastic dynamic programming (SDP) framework [9, 22] or the MDP framework [23, 24].

The rest of this chapter is organized as follows: Section 2.2 casts CTP as a finite-horizon MDP and a Deterministic POMDP. This section also develops the Bellman equation corresponding to the MDP formulation. Section 2.3 introduces the CAO* Algorithm for CTP. Section 2.4 presents our computational experiments. A summary and conclusions are presented in Section 2.5. We note that the terms "solution" and "policy" are used interchangeably in this manuscript.

2.2 MDP and POMDP Formulations

CTP can formally be defined as follows: Let $G = (V, E)$ be an undirected graph with designated vertices $s, t \in V$, and suppose there is a function $\ell : E \rightarrow \mathbb{R}_{\geq 0}$ assigning a length to each edge; the goal here is to find a shortest s, t traversal (walk) in G . However, not all of the edges may indeed be traversable. In particular, for a given subset $E' \subseteq E$ of edges called *stochastic edges*, there is a function $\rho : E' \rightarrow [0, 1)$ such that, for each edge $e \in E'$, $\rho(e)$, called the *mark* of edge e , is the probability that e is not traversable independent of the other edges. The edges in $E \setminus E'$ are called *deterministic edges* and they are known a priori to be traversable. For any edge $e \in E'$, when the traversal is at an endpoint of e , the agent has the option to disambiguate e , that is, learn whether e is traversable. Edges cannot be traversed until they are known to be

traversable, and the traversability status of each edge is static and will never change over the course of the traversal. To avoid infinite expected length, we assume the existence of a (possibly very long) s, t path consisting of deterministic edges. Of course, if the agent follows any particular policy then the traversal is still random and will unfold depending on the results of the disambiguations. Thus, the traversal's distribution is specified through ρ . The agent's goal is to find an optimal policy in the sense of having shortest expected length. Finding such an optimal policy is the *Canadian Traveler Problem (CTP)*. Without loss of generality, we additionally assume that there is a limit $K \leq N$ on the number of available disambiguations where $N := |E'|$.

2.2.1 MDP Formulation and The Bellman Equation

A Markov Decision Process (MDP) is a 4-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ of states, actions, transition function, and rewards respectively where

- \mathcal{S} is a set of states: At every stage $k = 0, 1, 2, \dots, \mathcal{K}$ (where \mathcal{K} is the final stage, or $\mathcal{K} = \infty$), the agent is at one of these states;
- \mathcal{A} is a set of actions: At every stage, the agent chooses one of them depending on what his current state is;
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the state-transition function: For any $s, s' \in \mathcal{S}$ and $\alpha \in \mathcal{A}$, $\mathcal{T}(s, \alpha, s')$ is the probability of ending up in state s' in the next stage given that the agent is at state s in the current stage and chooses action α ; and,
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function: $\mathcal{R}(s, \alpha)$ represents the immediate reward the agent gains for choosing action α at state s .

The agent's objective is to maximize the expected sum of rewards, i.e., $E[\sum_{k=0}^{\mathcal{K}} R_k]$, where R_k is the reward received at stage k . CTP can be cast as a finite-horizon MDP as follows:

- **States:** In order to keep track of the agent's current knowledge of the status of the stochastic edges, we define the *information vector* $\mathcal{I} \in \{\text{"A"}, \text{"T"}, \text{"U"}\}^N$, such that, for all $i = 1, 2, \dots, N$, the i -th entry of \mathcal{I} is "A", "T", or "U", depending on

whether the current status of the i -th stochastic edge is, respectively, ambiguous, traversable, or untraversable.

Let \mathcal{Y} be the union of s , t , and the set of all disambiguation vertices, i.e., endpoints of stochastic edges. If there are certain vertices at which multiple stochastic edges can be disambiguated, these vertices are included in \mathcal{Y} with their respective multiplicities. The MDP state space \mathcal{S} is defined as $\mathcal{Y} \times \{\text{"A"}, \text{"T"}, \text{"U"}\}^N$. The state space thus represents possible disambiguation vertices at which the agent may be at a particular stage, coupled with information that describes the agent's knowledge at that stage.

- **Actions:** The set of actions \mathcal{A} is $\mathcal{Y} \setminus \{s\}$, i.e., all the vertices where a disambiguation can be performed and the termination vertex.
- **State Transition Function:** Given a state and an action, the state transitioned into is comprised of the vertex identified in the action and the information vector of the previous state updated to indicate whether the stochastic edge identified in the action is traversable or not. The respective probabilities are specified according to the mark of the disambiguated edge.
- **Rewards:** The reward for a specific action at any particular state is the negative of the shortest path distance between the vertex identified in the state and the vertex identified in the action—avoiding all ambiguous and untraversable stochastic edges as indicated by that state's information vector.

The above state space, set of actions, rewards, and state transition function comprise a Markov decision process with K stages (or N stages if there is no limit on the number of available disambiguations).

We now present the Bellman equation corresponding to the above MDP formulation, which can be solved via value iteration. The notation used in the Bellman equation is defined below.

- For $\mathbf{s} = (y, \mathcal{I}) \in \mathcal{S}$ and stage $k \leq K$, the *value function* $V_k^* : \mathcal{S} \rightarrow \mathbb{R}$ is defined as the negative of the shortest expected $y - t$ path length under an optimal policy when the status of the underlying graph is \mathcal{I} and there are k disambiguations left.

- For any $y, y' \in \mathcal{Y}$ and information vector \mathcal{I} , $q(y, y', \mathcal{I})$ is defined as the length of the shortest $y - y'$ path while avoiding all the untraversable and ambiguous stochastic edges as indicated by \mathcal{I} .
- For any $y \in \mathcal{Y}$, \mathcal{I}_y is defined as the component of \mathcal{I} corresponding to the stochastic edge associated with y .
- For any $y \in \mathcal{Y}$, $\rho(y)$ is defined as the mark of the stochastic edge associated with y .
- For information vector \mathcal{I} and $y \in \mathcal{Y}$, $T_{\mathcal{I},y}$ and $U_{\mathcal{I},y}$ are defined as the information vectors whose components are the same as \mathcal{I} except at the component corresponding to y , which is set to "T" and "U", respectively.

For $k = 1, \dots, K$, and $\mathbf{s} = (y, \mathcal{I}) \in \mathcal{S}$, the Bellman equation is as follows:

$$V_k^*(\mathbf{s}) = \max_{\substack{y' \in \mathcal{Y} \text{ s.t.} \\ y'=t \\ \text{or} \\ \mathcal{I}_{y'} = \text{"A"}}} \{ -q(y, y', \mathcal{I}) + \rho(y')V_{k-1}^*(y', T_{\mathcal{I},y'}) + (1 - \rho(y'))V_{k-1}^*(y', U_{\mathcal{I},y'}) \}. \quad (2.1)$$

The optimal solution to CTP is then given by $-V_K^*(s, (\text{"A"}, \dots, \text{"A"}))$. Note that value iteration entails exhaustively back-solving complete stages from stage 1 up to stage K , where stage 0 values $V_0^*(y, \mathcal{I})$ are given by $-q(y, t, \mathcal{I})$. Due to the exponentially many number of states, value iteration is not practical for CTP, as illustrated in our computational experiments. Yet, the MDP formulation provides valuable insight into the structure of CTP and illustrate its difficulty.

2.2.2 Deterministic POMDP Formulation

A POMDP is denoted as a 6-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, O \rangle$ where

- $\mathcal{S}, \mathcal{A}, \mathcal{T}$, and \mathcal{R} denote a Markov Decision Process;
- Ω is a set of observations the agent can make; and,
- $O : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \Omega \rightarrow [0, 1]$ is the observation function: For each current state, action, and resulting state, it specifies a probability distribution over possible observations. Specifically, $O(s, \alpha, s', o)$ is the probability of observing o when the agent is at state s , chooses action α , and ends up in state s' .

An intermediate class of problems between MDPs and POMDPs is Deterministic POMDPs, which allow for state uncertainty (as in POMDPs) but assume perfect observations (as in MDPs) [17]. As discussed in Bnaya et al. [14], CTP can in fact be cast as a Deterministic POMDP. In this section, we present an explicit Deterministic POMDP formulation of CTP. We believe that this aspect of CTP is rather important from a theoretical point of view as it offers an insight into the inherent relationship between MDPs and Deterministic POMDPs, and opens up the possibility of adapting existing MDP/ POMDP algorithms to other Deterministic POMDP/ MDP problems.

We cast CTP as a Deterministic POMDP by trimming the set of information vectors to $\{\text{"T"}, \text{"U"}\}^N$ and folding the ambiguity of stochastic edges into ambiguity of the information vector, hence the "partial observability" of the state. In our POMDP formulation, we assume that there is no limit on the number of available disambiguations. The motivation for this assumption is that the reward for each state/ action pair needs to be specified a priori and the agent may have to re-disambiguate certain stochastic edges in order to traverse the shortest path used to calculate this reward. We formulate the components of $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, O \rangle$ as follows:

- **States:** The POMDP information vector is defined as $\mathcal{I}' \in \{\text{"T"}, \text{"U"}\}^N$ such that, for all $i = 1, 2, \dots, N$, the i -th entry of \mathcal{I}' is "T" or "U". For each disambiguation vertex $y \in \mathcal{Y}$, we introduce two points: y_{out} that is an exact copy of y (called an *out-point*), and y_{in} that is y infinitesimally perturbed from its location towards the other endpoint of the edge (called an *in-point*)— y_{in} cannot be arrived at unless the associated stochastic edge is traversable. Let \mathcal{Y}' be the union of s , t , and, the in- and out-points associated with each disambiguation vertex. The POMDP state space is defined as $\mathcal{Y}' \times \mathcal{I}'$. We shall refer to the location component of a given state as the *state point*.
- **Actions:** The set of actions is $\mathcal{Y}' \setminus \{s\}$.
- **State Transition Function:** Given a state and an action, the state point of the state transitioned into is the point as identified by the action. The information vector of state transitioned into is always the same as the information vector of the current state.

- **Rewards:** At any state whose state point is an out-point, the reward for choosing the corresponding in-point as the action is 0 if the associated stochastic edge is traversable, and $-\infty$ otherwise. For any other state/ action pair, the reward is the negative of the shortest path distance between the state point and the action avoiding all the stochastic edges (regardless of their actual status) except the one that the agent is currently traversing.
- **Observations:** The set of observations Ω is {traversable, untraversable}.
- **Observation Probabilities:** Since state transitions are always deterministic, the observations only need to be specified for state/ action pairs based on the actual status of the stochastic edges associated with the actions.

Observe that in the above formulation, state transitions and observations are deterministic and all the ambiguity is folded into the information vector. The information vector, on the other hand, represents the agent’s current knowledge of the underlying graph, whose probability distribution (called the *belief state* in POMDP terminology) is specified by the marks of the stochastic edges.

2.3 The CAO* Algorithm

In this section, we first define AO trees, which can be used to represent a given CTP instance where the edges correspond to sequential decisions that can be made and their probabilistic outcomes. We then describe the AO* Algorithm for searching AO trees and introduce the CAO* Algorithm for CTP.

2.3.1 AO Trees

AO trees can be used to selectively search partial solutions of the optimality conditions without exhaustively back-solving complete stages as in value iteration [25].

An AO tree is defined as a rooted tree $T = (N, A)$ with a function $\ell : A \rightarrow \mathbb{R}_{\geq 0}$ assigning a length to each arc, and a function $p : A \rightarrow [0, 1]$ assigning a probability to each arc. The node set N is partitioned into a set of AND nodes, denoted by N_A , and a set of OR nodes, denoted by N_O . All arcs emanating from OR nodes have probability one.

Denote by $S(n)$ the successors of the node $n \in \mathbf{N}$ in the AO tree. A function $g : \mathbf{N} \rightarrow \mathbb{R}_{\geq 0}$ is said to be *consistent* on a collection of nodes $\mathbf{C} \in \mathbf{N}$ provided that, for all $n \in \mathbf{C}$, the following three conditions are satisfied:

- if $n \in \mathbf{N}_A$, $g(n) = \sum_{n' \in S(n)} [p((n, n')) \cdot (\ell(n, n') + g(n'))]$,
- if $n \in \mathbf{N}_O$, $g(n) = \min_{n' \in S(n)} \{\ell(n, n') + g(n')\}$, and,
- if $n \in \mathbf{N}$ is a leaf node, $g(n)$ is zero.

The function $f : \mathbf{N} \rightarrow \mathbb{R}_{\geq 0}$ that is consistent on \mathbf{N} is called the *cost-to-go function* and $f(n)$ is called the *true label* of node n . Typically, ℓ and p are given explicitly, and f is implicitly defined via ℓ and p . For instance, within the context of CTP, f denotes the negative of the value function V^* defined in Section 2.2.1. Given an AO tree, the goal is to compute the true label of the root node, which denotes the optimal value of an underlying decision problem.

With an appropriately chosen AO tree, a CTP instance can be solved by computing the true label of the root node. Specifically, associated with each node n is a state $s_n = (y_n, \mathcal{I}_n)$ from the state space $\mathcal{S} = \mathcal{Y} \times \{\text{"A"}, \text{"T"}, \text{"U"}\}^N$. The root r is an OR node with $y_r = s$ and $\mathcal{I}_r = (\text{"A"}, \dots, \text{"A"})$, which is the first level in the tree. All subsequent odd levels consist of OR nodes corresponding to possible disambiguation vertices and even levels consist of AND nodes which are either leaf nodes denoting direct traversal to termination, or, have two successors each of whom corresponds to traversable/untraversable disambiguation outcomes¹. In particular, for any arc $a = (n, n') \in \mathbf{A}$,

- If $n \in \mathbf{N}_O$, $\ell(a)$ is set to $q(y_n, y_{n'}, \mathcal{I}_n)$ and $p(a)$ is set to one.
- If $n \in \mathbf{N}_A$, $\ell(a)$ is set to zero; $p(a)$ is set to $\rho(y_n)$ if n' corresponds to a untraversable disambiguation outcome and $1 - \rho(y_n)$ otherwise.

Note that this construction is essentially a mapping of all the actions the agent can choose and all the disambiguation outcomes that can occur. In particular, this construction ensures that $f(n)$ is the negative of $V^*(s_n)$ for any node $n \in \mathbf{N}_O$.

¹In this work, we adopt the convention that each disambiguation resolves the ambiguity of exactly one stochastic edge. This convention especially makes sense when cost of disambiguation is nonzero.

In an AO tree representation of CTP, the optimal disambiguation policy is a collection \mathcal{C} of all the arcs whose lengths are explicitly included in the calculation of $f(r)$. Specifically, we first define the function $m : \mathbf{N}_O \rightarrow \mathbf{N}_A$ such that $m(n) := \arg \min_{n' \in S(n)} \{\ell(n, n') + f(n')\}$ for any $n \in \mathbf{N}_O$. The collection \mathcal{C} can be found recursively as follows:

- Step 1.** Set $\mathcal{C} := \emptyset$ and $n_m := r$.
- Step 2.** If $m(n_m)$ is a leaf node, augment \mathcal{C} by $(n_m, m(n_m))$. Otherwise, augment \mathcal{C} by $(n_m, m(n_m)), (m(n_m), n')$, and $(m(n_m), n'')$ where n' and n'' are the successors of $(m(n_m))$.
- Step 3.** Set $n_m := n'$. If n_m is a leaf node, stop. Otherwise go to Step 2.
- Step 4.** Set $n_m := n''$. If n_m is a leaf node, stop. Otherwise go to Step 2.

As an example, Figure 2.1 illustrates a simple CTP instance with two stochastic and three deterministic edges. In the figure, solid lines denote deterministic edges and dashed ones denote stochastic edges. Numbers above each edge denote the edge's length and marks above stochastic edges are shown in parentheses next to the edge length.

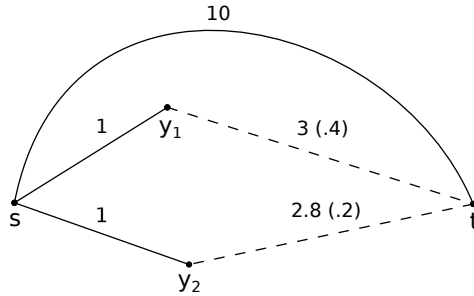


FIGURE 2.1: A simple CTP instance with two stochastic edges (denoted by dashed lines) and three deterministic edges (denoted by solid lines). Length of each edge is given above the edge. Shown in parantheses are the marks of stochastic edges.

The complete AO tree corresponding to this instance is shown in Figure 2.2. 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) by circles. In the figure, the letters "U" and "T" next to arcs emanating from AND nodes denote untraversable and traversable disambiguation outcomes respectively. The numbers next to arcs emanating from OR nodes denote their length and thick-bordered circles represent leaf nodes.

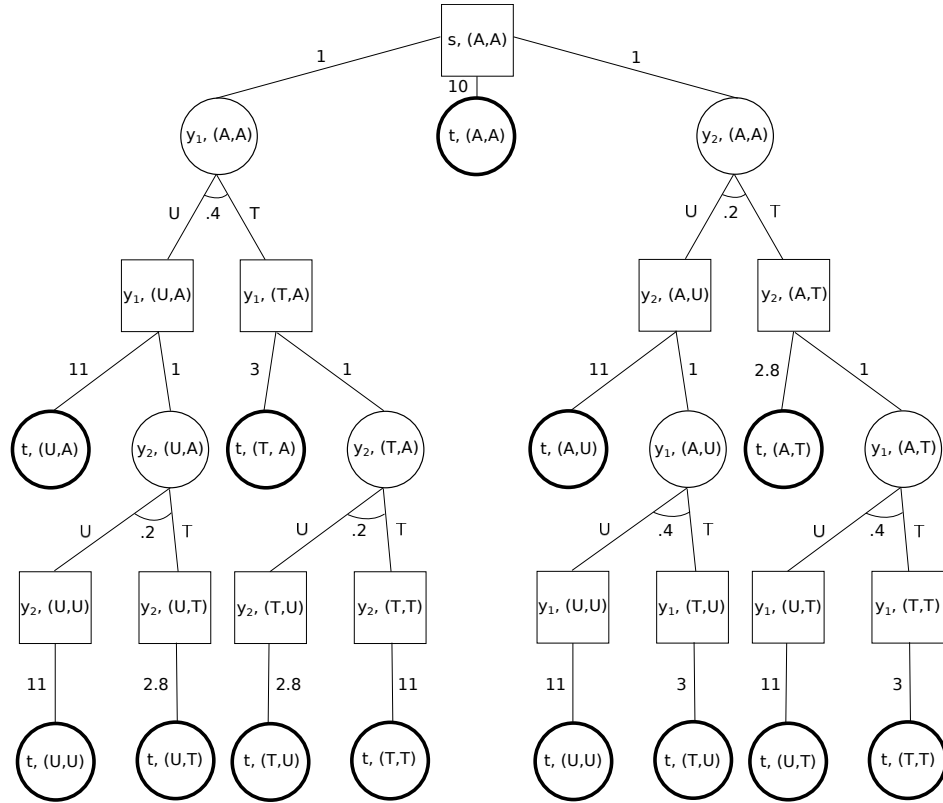


FIGURE 2.2: The complete AO tree corresponding to the CTP instance shown in Figure 2.1. OR nodes are depicted by squares and AND nodes by circles.

The benefit of the AO tree representation is that, as shown in Section 2.3.2, it allows for selectively evaluating the value function in a top-down fashion rather than back-computing all of them for every stage as in value iteration.

2.3.2 The AO* Algorithm

Theoretically, an optimal solution to a problem represented by an AO tree can be determined by computing $f(n)$ for all $n \in \mathbf{N}$ in a bottom-up fashion. However, the exponential number of nodes in the AO tree representation of CTP makes this approach prohibitively expensive. On the other hand, not all the nodes' true labels need to be calculated to determine the true label of the root node. We define *searching* an AO tree as identifying the nodes that are of interest in determining the true label of the root node.

The classical AO* Algorithm for searching AO trees [26, 27] improves upon the brute force approach by utilizing *admissible* lower bounds $h^{lower} : \mathbf{N} \rightarrow \mathbb{R}_{\geq 0}$, called *heuristic*

labels, which are lower bounds that are guaranteed not to overestimate the true label of any node. These lower bounds guide the search in a top-down fashion so that only a small portion of the complete AO tree is examined. Note that even though the labels are referred to as “heuristic” in the AO* terminology, the AO* Algorithm itself is optimal as long these lower bounds are admissible.

The AO* Algorithm grows a *solution tree* $S = (N', A')$, which is a subtree of the complete AO tree and a representation of partial solutions of the optimality conditions. S initially consists of only the root node r , and is gradually augmented by two alternating steps, *expansion* and *propagation*, until $f(r)$ is computed. A node $n' \in N'$ is said to be *terminal* if $f(n')$ has been calculated. In the expansion step, the non-terminal leaf node with the lowest h^{lower} value, called the expansion node and denoted by n'_e , is found and its successors are added to S . The successors are then assigned lower heuristic labels. In the propagation step, $h^{lower}(n'_e)$ is recalculated using the labels of its successors—true labels for successors that are leaf nodes in the complete AO tree and lower heuristic labels otherwise—and the new label is propagated up S until a node is reached whose lower heuristic label is not affected. Terminal status of nodes are also updated accordingly during the propagation step.

For instance, the solution tree associated with the instance in Figure 2.1 is shown in Figure 2.3. This solution tree is interpreted in the following way: At the outset, the optimal decision for the agent is to disambiguate the edge (y_2, t) . If it turns out to be traversable, then traverse to t and stop. Otherwise, follow the y_2, s, y_1 path and disambiguate the edge (y_1, t) . If it turns out to be traversable, then traverse to t and stop. Otherwise, traverse the y_1, s, t path and stop. Node true labels can then be calculated as follows: $f(y_1, (“A”, “U”)) = (0.6)(3) + (0.4)(11) = 6.2$; $f(y_2, (“A”, “A”)) = (0.8)(2.8) + (0.2)(1 + 6.2) = 3.68$. The optimal expected path length is then calculated as $f(r) = f(s, (“A”, “A”)) = 1 + 3.68 = 4.68$.

In CTP, an admissible lower bound on $f(n)$ for any $n \in N$ is available in the form of the deterministic shortest path length from y_n to termination while avoiding only untraversable stochastic edges as indicated by \mathcal{I}_n . That is, during the calculation of this $y_n - t$ shortest path, ambiguous stochastic edges in \mathcal{I}_n are assumed to be traversable (in addition to the deterministic edges and other stochastic edges that have already been found to be traversable), and only the stochastic edges that have been found to be

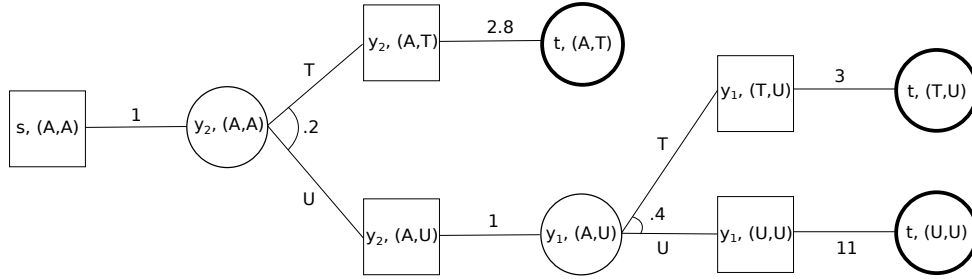


FIGURE 2.3: The solution tree (shown horizontally) corresponding to the CTP instance shown in Figure 2.1.

untraversable are avoided. We call these “natural” admissible lower bounds and denote by $\hat{h}^{lower}(n)$. Only in the best case scenario all the ambiguous stochastic edges would be revealed to be traversable for the rest of the traversal from y_n to t , and therefore $\hat{h}^{lower}(n)$ never overestimates $f(n)$. Thus, one can employ these lower bounds to solve CTP using the AO* Algorithm.

2.3.3 The CAO* Algorithm

CTP has two important properties that can be fruitfully exploited:

- **Admissible Upper Bounds:** For a node $n \in \mathbb{N}$, we call an upper bound “admissible” if it never underestimates the node’s true label $f(n)$. Similar to the natural admissible lower bounds, a naturally admissible upper bound on $f(n)$ for $n \in \mathbb{N}$ is also available in the form of the deterministic shortest path length from y_n to termination while avoiding ambiguous stochastic edges in addition to the untraversable ones as indicated by \mathcal{I}_n . We denote these upper bounds by $\hat{h}_{natural}^{upper}(n)$.

Note that $f(n)$ for $n \in \mathbb{N}$ is in fact the shortest expected $y_n - t$ path length under the information state \mathcal{I}_n , which essentially stands as a CTP instance itself. Thus, sub-optimal, yet fast algorithms designed for CTP can be executed for the instance corresponding to n , which would also be an admissible upper bound on $f(n)$. In this work, we advocate utilization of the DT Algorithm for CTP [21]. The DT Algorithm is sub-optimal, but it runs in a fraction of a second in general and yields good solutions. (This algorithm was originally cast for D-SOSP, but it can be modified for general CTP in a straightforward manner [28]). The expected path length obtained by the DT Algorithm for node n shall be denoted by $\hat{h}_{DT}^{upper}(n)$.

Given the above two admissible upper bounds for a node n , we define $\hat{h}^{upper}(n)$ to be the tighter of the two; that is,

$$\hat{h}^{upper}(n) := \min(\hat{h}_{natural}^{upper}(n), \hat{h}_{DT}^{upper}(n)).$$

- **State Overlaps in the AO Tree:** The agent might end up at the same particular state in the AO tree representation after visiting different sequences of states. Therefore, some of the states can reside in multiple nodes in the AO tree when $K > 1$.

The fact that the same state can reside in different nodes in the AO tree is illustrated in Figure 2.5 for the simple CTP instance in Figure 2.4. This instance has three stochastic and four deterministic edges. As shown in Figure 2.5, the state $s = (y_3, ("U", "U", "A"))$ can be reached at from two different paths. Namely, under the scenario where both of the stochastic edges (y_1, y_3) and (y_2, y_3) are untraversable, the agent can arrive at s either by disambiguating first the edge (y_1, y_3) and then (y_2, y_3) ; or by disambiguating first the edge (y_2, y_3) and then (y_1, y_3) . CAO* caches the AND node corresponding to s upon first encounter. Upon the second encounter, CAO* adds the corresponding parent/child links and avoids generation of a new node for s . Whenever the heuristic label of s changes, this change is propagated recursively for both of its parents.

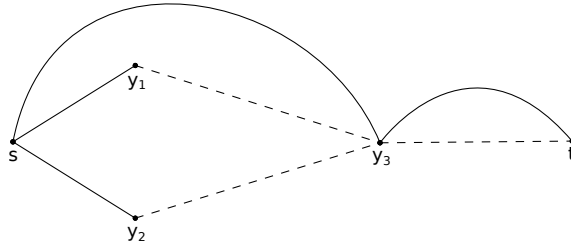


FIGURE 2.4: A CTP instance with three stochastic and four deterministic edges.

In this section, we present the CAO* Algorithm that is an improvement on AO* that takes advantage of the above two properties of CTP. Key features of CAO* are as follows:

- **Feature #1: Expansion of OR Nodes:** CAO* only expands OR nodes. During the expansion of $n_o \in \mathbf{N}_O$, the following nodes are automatically generated and added to the AO tree: successor AND nodes $n_a \in \mathbf{N}_A$ denoting traversal to reachable endpoints of currently stochastic edges, their two children OR

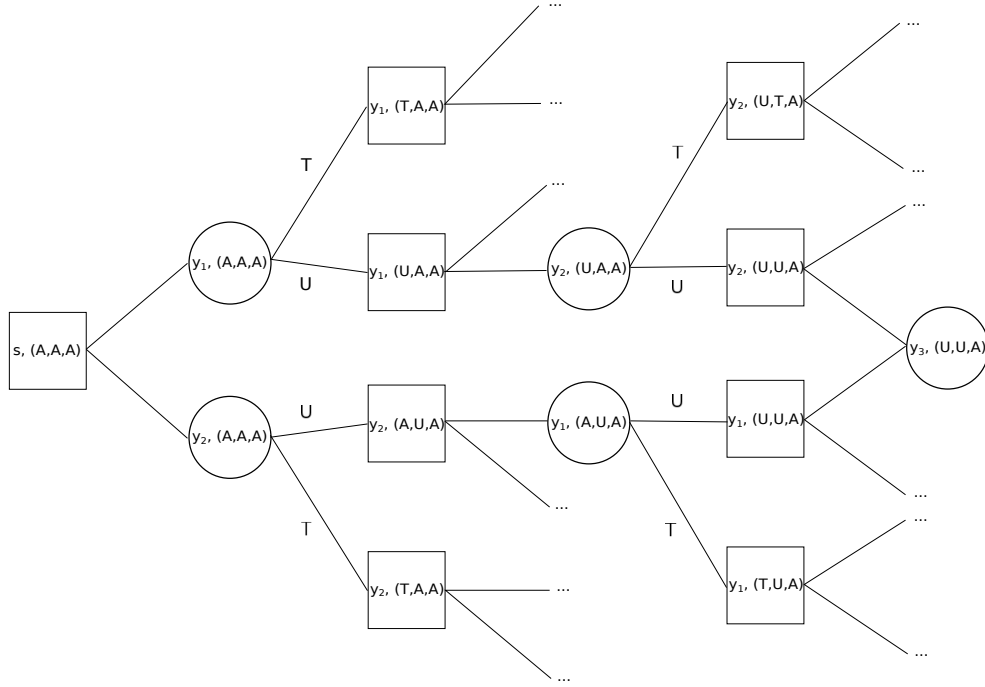


FIGURE 2.5: The partial AO tree corresponding to the CTP instance shown in Figure 2.4 illustrating the state overlap phenomena. The state $(y_3, ("U", "U", "A"))$ can be reached at from two different paths.

nodes (corresponding to traversable and untraversable disambiguation outcomes), and an AND node representing direct traversal to termination—but only when $\ell(n_o, n_a) + \hat{h}^{lower}(n_a) \leq \hat{h}^{upper}(n_o)$. For the AND nodes added to the AO tree, their heuristic labels $h^{lower}(n_a)$ are initialized to $\hat{h}^{lower}(n_a)$. Note that these heuristic labels are continuously updated during the cost propagation steps, providing better and better lower bounds during the search until the true node label has been calculated, after which the node is marked as terminal.

- Feature #2: Caching of AND Nodes:** CAO* maintains a cache of all the AND nodes added to the AO tree thus far in a hash map data structure. During the expansion step, whenever a new AND node is to be added to the tree, CAO* checks the cache to see if this node has already been generated. If so, it adds this AND node to the children list of the OR node that is being expanded and also adds the OR node to the cached AND node’s parents list. If this AND node is not in the cache, a new AND node is created and it is linked to the OR parent that is being expanded, after which this AND node is added to the cache. Thus, in CAO*, an OR node has only one parent and possibly many children whereas

an AND node has only two children and possibly many parents. During the cost propagation step, updated costs are propagated recursively for each parent of AND nodes.

The fact that an AND node can have multiple parents indicate that the data structure maintained by CAO* is technically a graph and not a tree, albeit a graph with a special tree-like structure in the sense that OR parents of an AND node reside exactly one level up from that AND node. With a slight abuse of terminology, we shall continue to refer to this structure as an AO tree.

- **Feature #3: Dynamic AND Node Pruning:** Whenever a new AND node n_a is generated and added to the tree, CAO* computes and stores its admissible upper bound $\hat{h}^{upper}(n_a)$ (this upper bound is computed only once). As the updated h^{lower} labels are propagated up the solution tree, these new labels are used to dynamically prune "bad" AND children of OR parents. Specifically, denote the OR parent by n_o and its children by n_a . Bad AND nodes n'_a are defined as nodes for which the following holds:

$$\ell(n_o, n'_a) + \hat{h}^{lower}(n'_a) > \min_{n_a} [\ell(n_o, n_a) + \hat{h}^{upper}(n_a)].$$

- **Feature #4: Reduced Overhead Cost:** If there is only one disambiguation left at the current expansion node, its true label is calculated and the node is marked as terminal. This feature eliminates the overhead cost of individually considering the successors of this OR node in future expansions.

2.4 Computational Experiments

Our goal in this section is to empirically assess the performance of CAO* on (1) general CTP instances on random Delaunay and grid graphs, and (2) the D-SOSP variant of CTP, which is essentially CTP with probabilistic dependency among edges.

2.4.1 The BAO* and PAO* Algorithms

Of particular interest is the BAO* Algorithm introduced in [29] for D-SOSP. BAO* is similar to CAO* in the sense that it is also based on the AO* Algorithm. However,

BAO* differs from CAO* in three major ways: (1) BAO* is cast for D-SOSP whereas CAO* is presented for general CTP, (2) BAO* does not employ any state caching logic, and (3) BAO* uses the static zero-risk s, t path length for pruning. Here, the zero-risk s, t path is defined as the shortest s, t path over the grid graph avoiding all stochastic edges, that is, the edges intersecting any disks. In contrast, CAO* avoids re-addition of previously encountered states to the AO tree via a state caching mechanism and uses the admissible upper bounds at a node level for dynamic state-space pruning.

Another AO*-based algorithm for CTP is the PAO* Algorithm presented in [12]. PAO* shares certain basic characteristics with CAO* and BAO* as all three are based on the classical AO* search. In that regard, PAO* also maintains and updates a partial solution tree. However, PAO* also maintains a complete AO tree representing the problem instance at hand during its execution. Yet, for a given CTP instance, the corresponding complete AO tree has exponentially many nodes and this observation essentially renders PAO* infeasible in relatively large problem instances. Thus, whereas PAO* requires storage of the complete AO tree in memory at all times until termination, CAO* attempts to minimize its memory footprint by maintaining as few nodes as possible via its caching mechanism and node pruning techniques.

One other feature of PAO* is that whenever a new lower bound is found for a node, PAO* scans the complete AO tree, finds the same nodes and updates their bounds accordingly, which is referred to as "sideways neighbors" updating. On the other hand, scanning of the complete AO tree for relevant sideways neighbors results in significant computational burden. In contrast, CAO* eliminates the need for such neighbor updating as the caching mechanism in CAO* is specifically designed to avoid recreation of the same nodes.

A third difference between CAO* and PAO* is that CAO* makes use of cost upper bounds for node pruning in the partial AO tree whenever possible whereas PAO* does not make use of any upper bounds nor it calls for any node pruning techniques. Our computational experiments involve comparison of CAO* against BAO* and PAO* in addition to standard AO* and value iteration.

2.4.2 Experimental Setup

Our computational experiments are comprised of four different simulation environments: Environments A and B empirically assess relative performance of CAO^* against VI, AO^* , BAO^* , and PAO^* on relatively small random problem instances where the algorithms can generally converge to optimality within a time limit of 5 hours per given problem instance. In particular, Environment A involves random CTP instances over Delaunay graphs with 100 vertices and 10×10 grid graphs respectively. On the other hand, Environment B involves random D-SOSP instances over a 20×15 grid with 10 and 15 disks respectively.

Environments C and D measure performance of CAO^* on relatively large D-SOSP instances. Specifically, Environment C is concerned with a real-world D-SOSP problem instance called COBRA data from a maritime minefield application, and Environment D deals with 6 random COBRA-like D-SOSP problem instances.

The experiments were conducted on a PC with a 3.9 GHz Intel Core i7 processor with 16 GB of memory. The algorithms were implemented in Java based on the **jgrapht** software package. Deterministic shortest path lengths in our experiments were computed using the A^* Algorithm where the admissible heuristic used was the Euclidean distance between the start and end grid vertices.

2.4.3 Simulation Environment A

This environment is concerned with random CTP instances over Delaunay and grid graphs, which are illustrated in Figure 2.6 respectively:

- Delaunay graphs with 100 vertices whose coordinates are randomly chosen over the region $[1,100] \times [1,100]$ on the plane. Edge lengths are set to the Euclidean distance between their end vertices and the two farthest vertices of the graph are designated as the starting and termination vertices respectively. Each grid edge has a 0.25 probability of being stochastic and marks of stochastic edges are sampled from the uniform distribution.
- Grid graphs where $i_{\max} = j_{\max} = 10$. The starting and termination vertices are taken as $s = (5, 10)$ and $t = (5, 1)$. As in Delaunay graphs, each edge has a 0.25 probability of being stochastic with uniform marks.

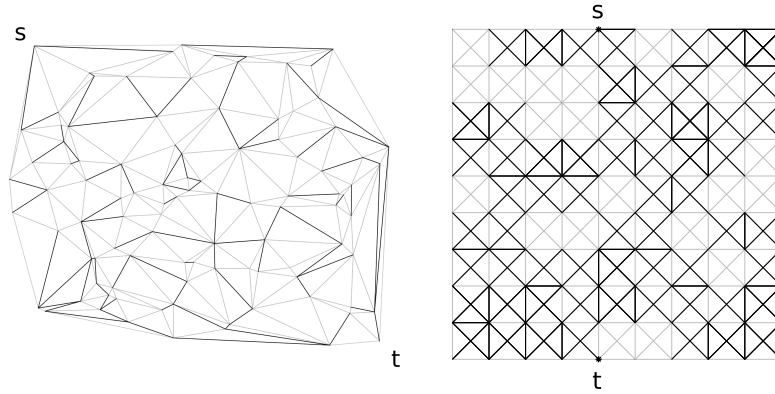


FIGURE 2.6: Experimental Delaunay and grid graph realizations in Simulation Environment A. Stochastic edges are shown in bold.

Table 2.1 compares relative performances of value AO^* , BAO^* , PAO^* , and CAO^* in this simulation environment for 10 experimental realizations for each graph type conditioned on having an admissible s, t path (to avoid infinite expected path length) for $K = 1, \dots, 4$. As discussed in Section 1.2, K denotes the number of available disambiguations. For instance, in the case of $K = 3$, the agent is allowed to disambiguate at most 3 ambiguous graph edges in its s, t traversal. Value iteration (VI) was not included in the comparisons due to its excessive memory requirements. The other algorithms were given a time limit of 5 hours for each problem instance for convergence to optimality. Any runs that did not fully converge to the optimal solution for a given problem instance within the time limit were terminated and excluded from the results.

Table 2.1 indicates that as K increases, performance gap between CAO^* and the other algorithms becomes even wider. Classical AO^* and PAO^* can only solve CTP instances upto $K = 2$ whereas BAO^* can solve upto $K = 3$ within the 5 hour limit. On the other hand, CAO^* can solve instances with $K = 4$ within several seconds.

2.4.4 Simulation Environment B

In this environment, we generated random D-SOSP instances over a grid with $i_{\max} = 20$ and $j_{\max} = 15$ with disk marks sampled from the uniform distribution and disk centers sampled over the region $[3, 18] \times [3, 12]$ with disk radii taken as 2. The starting and termination vertices were taken as $s = (10, 15)$ and $t = (10, 1)$. In particular, this setup

TABLE 2.1: Run time comparison of AO^* , BAO^* , and CAO^* in Environment A in seconds. In the table, “-” stands for insufficient memory whereas “*” stands for non-convergence to optimality within 5 hours.

Type	K	AO^*		BAO^*		PAO^*		CAO^*	
		Mean	Std.	Mean	Std.	Mean	Std.	Mean	Std.
Delaunay	1	5.64	1.19	4.73	1.93	0.45	0.14	0.13	0.07
	2	5,643	1,917	293.25	112.5	77.52	35.41	0.61	0.29
	3	*	*	4,871	2,425	*	*	1.92	1.14
	4	*	*	*	*	-	-	4.82	3.96
Grid	1	7.18	1.76	5.34	1.93	0.45	0.14	0.17	0.09
	2	7,750	2,514	337.09	128.6	109.92	41.04	0.84	0.37
	3	*	*	6,065	3,134	*	*	2.58	1.31
	4	*	*	*	*	-	-	5.43	4.29

ensures that there is always an admissible path from s to t . This simulation environment is illustrated in Figure 2.7 for $N = 10$ where N denotes the total number of disks.

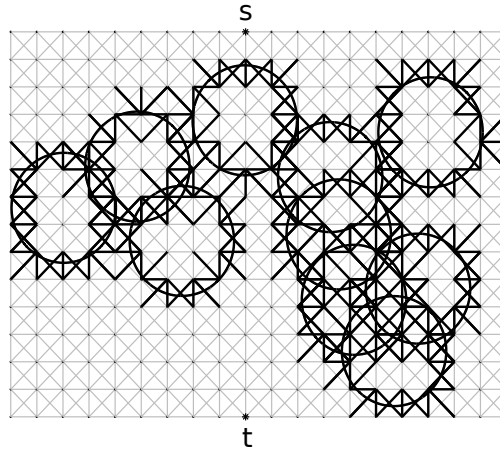


FIGURE 2.7: An experimental realization in Simulation Environment B with $N = 10$. Stochastic edges, i.e., grid edges intersecting disks are shown in bold.

We compared performances of VI, AO^* , BAO^* , PAO^* , and CAO^* for $N = 10, 15$ with $K = 1, 2$. Note that due to edge dependencies in D-SOSP, K here corresponds to the maximum number of ambiguous disks that can possibly be disambiguated in the agent’s s, t traversal. Table 2.2 shows the mean run times of the algorithms for 50 experimental realizations for each N, K combination listed along with their respective standard deviations. VI did not even run for $N = 15$ for either K due to insufficient memory. As seen in the table, mean AO^* run time was less than that of VI for $N = 10, K = 1$. However, since the same state can reside in multiple nodes in the AO

tree when $K > 1$, the number of AO* expansions can exceed the number of value function evaluations in VI, causing AO* run time surpass VI run time, as illustrated for $N = 10, K = 2$. On the other hand, CAO* avoids such re-expansions using a caching mechanism. In fact, over all the N, K combinations, CAO* was 270 times faster than BAO*, 400 times faster than PAO*, 770 times faster than VI, and 1,850 times faster than AO* on the average; illustrating the relative effectiveness of CAO* in solving random instances in Environment B.

TABLE 2.2: Run time comparison of VI, AO*, BAO*, and CAO* in Environment B in seconds. In the table, "-" stands for insufficient memory.

N	K	VI		AO*		BAO*		PAO*		CAO*	
		Mean	Std.	Mean	Std.	Mean	Std.	Mean	Std.	Mean	Std.
10	1	23.03	59.76	3.93	0.25	3.02	0.30	1.03	0.21	0.01	0.01
	2	223.2	37.81	781.2	200.4	121.2	109.8	60.13	21.33	0.13	0.14
15	1	-	-	1.98	0.12	1.59	0.12	0.63	0.13	0.02	0.01
	2	-	-	397.8	126.6	46.85	27.32	133.47	45.38	0.47	0.38
Overall		123.14	48.78	296.35	82.00	43.19	34.47	64.41	16.76	0.16	0.14

2.4.5 Simulation Environment C

This environment consists of a U.S. Navy minefield data set called the COBRA data with 39 disks [8, 19–21, 30]. The COBRA data set is the only publicly available real-world instance of SOSP within the context of maritime minefield countermeasures, which constitute a rather important application area of SOSP and CTP in general. In the COBRA data, disk centers are inside the region $[10, 90] \times [10, 90]$; the starting point is $s = (54, 80)$; the termination point is $t = (54, 10)$; and disk radii are taken as 5. This data set is illustrated in Figure 3.1 and tabulated in Table A.1 in Appendix A.

Table 2.3 presents the run time, number of expanded/ cached/ revisited/ pruned nodes respectively for CAO* on the COBRA data set for $K = 1, \dots, 5$. CAO* execution time ranged from 7.32 seconds (for $K = 1$) up to 38.18 minutes (for $K = 5$). For $K = 2$, CAO* execution time was 6.5 minutes whereas that of BAO* was 37.63 hours. Thus, for $K = 2$, CAO* was about 350 times faster than BAO* on the COBRA data set. For $K = 8$, number of nodes cached by CAO* per Feature #2 was about 0.45 million. These cached nodes were revisited about 2.7 million times during the execution of the algorithm, illustrating the benefits of the caching logic in CAO*. On the other hand, the

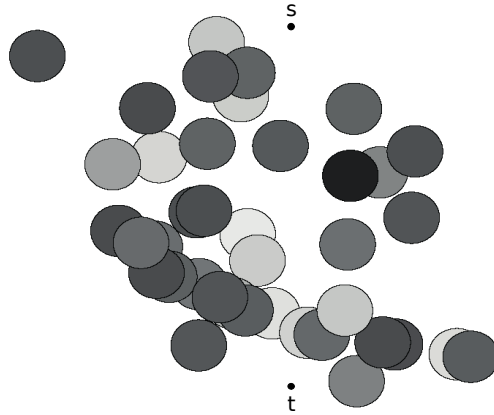


FIGURE 2.8: Illustration of the COBRA data where gray intensity reflects marks of disk with darker tones indicating a higher mark. Grid edges are not shown for clarity.

number of nodes pruned per Feature #3 for the same K value was about 3 million, which suggests that state-space search speed in D-SOSP can be increased dramatically by dynamic node pruning. Table 2.3 indicates that the caching mechanism and the dynamic node pruning feature provide significant benefits for other K values as well. In particular, note that due to the need for the agent to keep track of the status of all disks in D-SOSP, its theoretical computational complexity is $O(3^K)$. Examining the run times in Table 2.3, we observe that empirical complexity of CAO* on the COBRA data set is roughly $O(2^K)$.

TABLE 2.3: Performance of CAO* on the COBRA data set (Simulation Environment B). The columns denote the run time, number of expanded/ cached/ revisited/ pruned nodes respectively.

K	Run Time (sec)	Expanded	Cached	Revisited	Pruned
1	7.32	1	0	0	888
2	390.0	116	13,397	87,578	100,879
3	935.4	293	32,740	140,892	166,259
4	1,735.8	556	68,615	253,703	304,265
5	2,291.4	756	76,017	310,474	340,260
6	3,992.0	1,361	128,809	619,671	666,510
7	7,693.2	2,639	221,024	1,119,870	1,244,721
8	18,874.8	6,702	455,587	2,745,991	3,024,832

2.4.6 Simulation Environment D

Our goal in this section is to empirically assess general performance of CAO* on COBRA-like D-SOSP instances. This simulation environment consists of six COBRA-like instances with 39 disk-shaped obstacles with a radius of 5 over a square grid with $i_{\max} =$

$j_{\max} = 100$. Centers of these disks were sampled randomly over the region $[10, 90] \times [10, 90]$. To make the disk layout challenging, the zero-risk s, t path lengths were conditioned to be at least 130 units. The starting and termination vertices were taken as $s = (50, 100)$ and $t = (50, 1)$. These six instances are shown in Figure 2.9 and are tabulated in Appendix A.

Table 2.4 shows the average run time and node statistics for CAO* on the six COBRA-like problem instances for $K = 1, \dots, 4$. For $K = 4$, CAO* cached about 0.3 million nodes on the average. These cached nodes were revisited about 2.4 million times, underlying the importance of node caching in CAO*. In addition, CAO* pruned about 2.6 million nodes per Feature #3 for $K = 4$ on the average. Overall, Table 2.4 suggests that the caching and dynamic node pruning mechanisms result in significant computational savings when searching for the optimal policy in D-SOSP variant of CTP.

TABLE 2.4: CAO* performance averaged over the six COBRA-like instances in Simulation Environment C.

K	Run Time (sec)	Expanded	Cached	Revisited	Pruned
1	7.1	1	0	0	1,066.33
2	850.2	209.50	18,968.67	192,534.67	211,297.33
3	4,356.5	1,299.83	124,998.33	983,029.50	1,086,003.50
4	12,312.2	3,617.83	329,153.00	2,438,186.83	2,637,055.17

2.5 Summary and Conclusions

CTP is a difficult stochastic optimization problem that has practical applications in a number of probabilistic path planning domains. In this chapter, we first discuss Deterministic POMDP roots of CTP and present MDP and Deterministic POMDP formulations. Next, we introduce CAO* for CTP, which is an exact algorithm based on AO* search that takes advantage of CTP's special problem structure. In particular, CAO* uses a caching mechanism to avoid re-expansion of previously visited states and makes use of admissible lower and upper bounds at a node level for dynamic state-space pruning. CAO* is not polynomial time, but our experiments indicate that CAO* examines only a very small fraction of the state space and uses substantially less computational resources compared to AO* and value iteration to find an exact solution for CTP. In one particular case on general grid-based CTP instances, CAO* found the optimal solution

in several seconds whereas classical AO* was halted after 5 hours and value iteration did not even execute due to excessive memory needs. In one set of experiments involving D-SOSP instances, CAO* executed 770 times faster than value iteration and 1,850 times faster than the classical AO* Algorithm.

CAO* utilizes admissible lower and upper bounds at a node level for dynamic node pruning. One potential direction for future research would be to identify bounds tighter than those discussed in this chapter, which would potentially result in more aggressive node pruning and consequently reduce execution time. One other exciting direction for future research would be to use CAO* in conjunction with approximation schemes for CTP [9, 22–24]. CAO* can also be converted into a heuristic method by employing stronger, yet sub-optimal pruning techniques. In addition, CAO* can be employed to solve other variants of CTP and benchmarked against optimal solution methods other than those considered in this chapter.

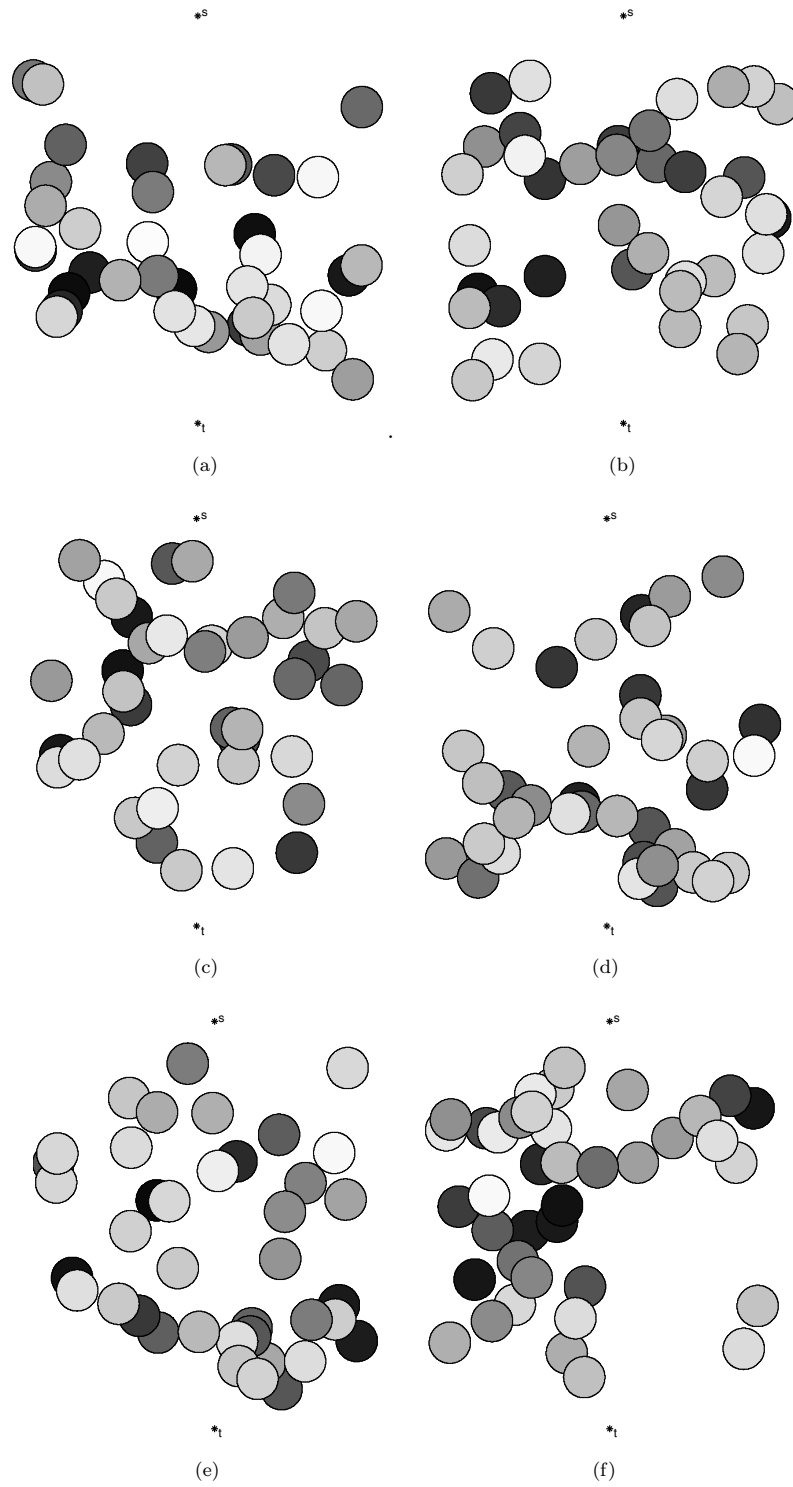


FIGURE 2.9: Six COBRA-like D-SOSP instances used in Simulation Environment D.

Chapter 3

A Fast and Effective Online Algorithm for the Canadian Traveler Problem

3.1 Introduction

There are several approximation and sub-optimal algorithms available in the literature for CTP [9–11] and there exist optimal algorithms for several special cases [13, 14]. In the previous chapter we show that CAO* significantly shortens the run time needed to find an exact solution to moderately-sized instances of D-SOSP and general CTPs. It is illustrated that CAO* runs several orders of magnitude faster than BAO*, AO*, and value iteration. Thus, we use CAO* in our computational experiments for the purpose of finding the optimal policy.

Regarding sub-optimal algorithms for CTP, of particular interest is the Distance-to-Termination (DT) Algorithm that has been originally proposed for D-SOSP by Aksakalli and Ari [21]. This algorithm involves successive calculation of deterministic shortest paths with respect to a specific edge weight function during the agent’s traversal. The authors present computational experiments that compare performance of the DT Algorithm against optimal policies obtained by the BAO* Algorithm on relatively small D-SOSP instances. Apart from DT, there are other heuristics for CTP in the literature as well. Eyerich et al. [11] evaluates these rollout-based heuristics both theoretically

and empirically. Detailed discussion and computational experiments focusing on rollout-based heuristics can be found in Chapter 3.

The contribution of this chapter is two-fold: (1) we show how the DT Algorithm can easily be adapted for general CTP and, (2) we provide computational experiments to empirically assess performance of the DT Algorithm on the D-SOSP variant where the optimal policies are obtained by the CAO* Algorithm. In particular, CAO* allows us to solve much larger problem instances to better benchmark DT Algorithm’s performance. We present experiments involving both real-world and synthetic data. Our results indicate that the DT Algorithm finds near-optimal policies in very short execution times and, its superior performance and computational savings are maintained on large problem instances as well. In what follows, we present adaptation of the DT Algorithm for general CTP, which is followed by our computational experiments.

3.2 The DT Algorithm

First introduced by Aksakalli and Ari [21], the notion of penalty-based algorithms for D-SOSP is a heuristic framework that involves successive calculation of deterministic shortest paths with respect to a specific edge weight function during the agent’s $s - t$ traversal. The idea behind using an edge weight function is to discourage traversing stochastic edges by assigning them additional weights. A penalty-based algorithm within the context of CTP employs the navigate-disambiguate-repeat (NDR) strategy described below:

1. Find the deterministic shortest path from start s to termination t in the graph where all the edge weights are assigned by the weight function.
2. Traverse the path until a vertex associated with an ambiguous stochastic edge is reached.
3. Since an ambiguous edge cannot be traversed, disambiguate the edge from the current vertex. Set the blockage probability to zero if the edge has been found to be traversable, and 1 otherwise.
4. Set the current vertex as the new starting vertex s and repeat 1 through 3 above until t is reached.

Aksakalli and Ari [21] generalizes the weight functions utilized in NDR strategy, using the notion of “penalty functions”:

$$w_D^F(e) := \ell^E(e) + \mathbf{1}_{e \in E'} \cdot F(e), \quad (3.1)$$

As a result, it is now possible to plug in different penalty functions to obtain different weight calculations. In fact, apart from DT Algorithm, the article includes an extensive discussion of two other penalty functions, Simulated Risk Disambiguation Algorithm (SRA) [8] and Reset Disambiguation Algorithm (RDA) [19]. In SRA, the penalty function F is specified as $F_{SR}(e) := \alpha \log(1 - \rho(e))^{-1}$ whereas it is defined as $F_{RD}(e) := \frac{c(e)}{1 - \rho(e)}$ for RDA. The first function is motivated by the idea of risk simulation (temporarily assuming that ambiguous edges are riskily traversable), where the second function is based on the idea of using the optimal weights for parallel graphs on arbitrary instances. A major disadvantage of SRA is that it requires to tune the parameter α for improved performance, thus, increased computational time. The lack of a tuning parameter, as it has been empirically shown, provides a significant advantage for RDA in terms of run time. However, despite its better performance and lack of tuning parameters, the weight function F_{RD} cannot be used when the disambiguation cost is zero. In other words, in a setting where the agent performs the disambiguation by simply a clear line of sight, F_{RD} is not applicable.

The above mentioned disadvantages of F_{SR} and F_{RD} reveals the quest to find a better penalty function. After extensive computational experiments, Aksakalli and Ari [21] observed that the penalty function $F_{DT}(e) := c(e) + \left(\frac{d_t(e)}{1 - \rho(e)}\right)^{-\log(1 - \rho(e))}$ consistently outperformed both of the former functions in most of the instances. The new function utilized the cost parameter as an additive term and it was monotonically nondecreasing in $c(e)$ and $\rho(e)$ for edges that intersect possible-obstacles in discretized SOSp. In particular DT algorithm uses the following weight for D-SOSP:

$$w_D^{DTA}(e) := \ell(e) + \mathbf{1}_{e \in E'} \cdot \left(c(e) + \left(\frac{d_t(e)}{1 - \rho(e)} \right)^{-\log(1 - \rho(e))} \right)$$

Above, $\mathbf{1}$ is the indicator function and $d_t(e)$ denotes the distance of edge e 's midpoint to t , hence the name “distance-to-termination”. The DT Algorithm thus calculates at most K deterministic paths and therefore it is extremely fast. It can also be used in an online fashion as the agent traverses the graph. Note, however, that computation of the

expected path length requires $O(2^K)$ path calculations. The authors, however, make the following observation regarding the DT Algorithm:

“Despite the fact that DTA performed remarkably well [...] in our simulations, it may or may not perform at the same level on obstacle fields with different topologies or with non-circular obstacle regions. Further research on instances with different characteristics is required in order to confirm that high performance of DTA is consistent across various problem settings. To that end, it might as well be the case that perhaps a different penalty function outperforms that of DTA in certain problem environments. Nonetheless, the NDR strategy guided by appropriate penalty functions seems to be an efficient and effective algorithmic framework for SOSP, and this chapter could be seen as a show case of this framework using the DT penalty function on an important real-world variant of the problem”.

3.3 Computational Experiments

In this section, performance of the DT Algorithm is empirically compared to CAO* on the D-SOSP variant of CTP. The computational experiments are conducted in a maritime minefield navigation domain. Our simulations were performed in two different environments: Environment 1 that is concerned with a real-world data set, and Environment 2 that involves synthetic data. In both environments, we consider cases with disambiguation limit $K = 1, \dots, 5$ and disambiguation cost $c = 0, 2, 4, 6$.

3.3.1 Environment 1

In the first environment, we consider a U.S. Navy minefield data set, called COBRA data, which was used in [7, 8, 19, 20]. This data set has 39 disk-shaped potential obstacles with disk radius $r = 5$ on a 100×100 integer lattice. A visual representation of the COBRA environment is shown in Figure 3.1.

Table 3.1 shows the experiment results performed on the COBRA data set where the “zero-risk” column denotes the length of the $s - t$ path avoiding all disks without performing any disambiguations. On the average, policies found by the DT Algorithm was only 1.3% worse than the optimal policy, yet mean DT run time was 7.8 seconds, which

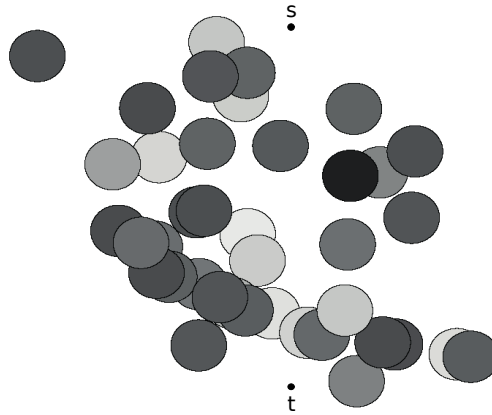


FIGURE 3.1: Illustration of COBRA data set. The gray intensity of disks reflect probability of the disks being true obstacles.

TABLE 3.1: Performance of the DT Algorithm on COBRA data set for the K, c combinations listed.

K	c	Expected Distance (units)				Run Time (seconds)		
		Zero-Risk	OPT	DTA	% Diff.	OPT	DTA	Ratio
1	0	104.33	80.02	80.17	0.18	7.32	3.46	2.12
	2	104.33	82.02	82.17	0.18	5.47	3.21	1.70
	4	104.33	84.02	84.17	0.17	8.39	3.71	2.26
	6	104.33	86.02	86.17	0.17	8.04	3.68	2.18
2	0	104.33	75.47	80.25	6.34	389.80	4.50	86.62
	2	104.33	79.47	79.74	0.34	1404.12	4.72	297.48
	4	104.33	81.77	81.94	0.21	707.33	4.47	158.24
	6	104.33	83.98	84.15	0.21	1422.32	4.81	295.70
3	0	104.33	74.20	78.20	5.39	935.58	9.49	98.59
	2	104.33	79.27	79.78	0.63	4261.26	10.43	408.56
	4	104.33	81.73	82.02	0.36	1582.92	9.70	163.19
	6	104.33	83.97	84.27	0.35	1304.64	9.56	136.47
4	0	104.33	73.81	76.93	4.23	1736.94	9.80	177.24
	2	104.33	79.02	79.54	0.66	4241.33	11.08	382.79
	4	104.33	81.56	81.82	0.31	2579.95	10.01	257.74
	6	104.33	83.85	84.09	0.29	2224.39	10.22	217.65
5	0	104.33	73.51	76.93	4.64	2291.41	10.10	226.87
	2	104.33	79.01	79.54	0.67	4992.36	11.92	418.82
	4	104.33	81.56	81.82	0.32	3802.77	11.21	339.23
	6	104.33	83.85	84.09	0.29	3202.90	11.08	289.07
Mean		104.33	80.40	81.39	1.30	1855.46	7.86	198.13
Std.		0.00	3.72	2.59	2.02	1565.04	3.25	138.14
Median		104.33	81.56	81.82	0.32	1502.62	9.63	156.03

is about 200 times faster than CAO*. In fact, median percent difference for DT in terms of expected path length was merely 0.3%.

3.3.2 Environment 2

In Environment 2, we randomly sampled six “COBRA-like” instances with 39 disks with a radius of 5 units on a 100×100 integer lattice. To make the environment even more challenging, the instances were conditioned to have a zero-risk path length of at least 130 units. The results are shown in Table 3.2. Similar to Environment 1, DT Algorithm found solutions very close to the optimal within very short execution times. On the average, policies found by the DT Algorithm was 3.17% worse than the optimal policy. However, mean DT run time was 8.15 seconds, which is about 740 times faster than CAO*. In fact, DT Algorithm ran up to 3300 times faster than CAO*. On the other hand, median percent difference for DT in terms of expected path length was only 0.96%. It can be also observed that computational benefits of DT Algorithm get more significant as the disambiguation limit K is increased.

3.4 Conclusions and Future Research

3.4.1 Conclusions

CTP is a difficult stochastic path planning problem and D-SOSP is perhaps the most realistic variant of CTP. These problems have practical applications in robot navigation, adaptive traffic routing, and mine-field navigation. In this chapter, we consider the DT Algorithm for CTP, which is a sub-optimal online algorithm that is fast and effective. This algorithm involves successive calculation of deterministic shortest paths with respect to a certain edge weight function during the agent’s traversal. We provide computational experiments to empirically assess performance of the DT Algorithm on the D-SOSP variant. In our experiments, the optimal policies are obtained by the CAO* Algorithm, which is a state-of-the-art exact algorithm for CTP based on the classical AO* Search. We present computational experiments involving both real-world and synthetic data. Our results indicate that the DT Algorithm finds near-optimal policies in very short execution times.

TABLE 3.2: Average performance of the DT Algorithm on six COBRA-like data sets for the K, c combinations listed.

K	c	Expected Distance (units)				Run Time (seconds)		
		Zero-Risk	OPT	DTA	% Diff.	OPT	DTA	Ratio
1	0	138.27	119.21	132.70	11.32	10.01	3.44	2.91
	2	138.27	121.21	134.70	11.13	10.44	4.01	2.60
	4	138.27	123.21	137.03	11.22	10.12	4.03	2.51
	6	138.27	125.21	131.01	4.63	9.94	3.76	2.64
2	0	138.27	110.52	112.16	1.49	1214.73	4.22	287.85
	2	138.27	113.58	114.96	1.21	1376.05	5.13	268.24
	4	138.27	116.38	116.83	0.39	1413.53	5.13	275.54
	6	138.27	119.17	123.63	3.75	1609.05	5.59	287.84
3	0	138.27	107.72	109.71	1.85	6179.67	11.02	560.77
	2	138.27	111.21	112.65	1.29	5249.41	9.98	525.99
	4	138.27	114.36	115.50	1.00	4987.30	9.72	513.10
	6	138.27	117.34	118.68	1.15	4808.78	9.79	491.19
4	0	138.27	106.22	109.10	2.71	17609.29	11.45	1537.93
	2	138.27	110.76	112.10	1.20	12125.10	11.37	1066.41
	4	138.27	113.97	115.01	0.91	8897.27	10.68	833.08
	6	138.27	116.97	118.04	0.91	7911.87	10.04	788.03
5	0	138.27	105.54	109.00	3.27	35590.07	10.76	3307.63
	2	138.27	110.17	112.03	1.69	17621.04	10.92	1613.65
	4	138.27	113.45	114.69	1.09	13920.19	10.92	1274.74
	6	138.27	116.53	117.80	1.09	13065.81	10.94	1194.31
Mean		138.27	114.64	118.37	3.17	7680.98	8.15	741.85
Std.		0.00	5.42	8.75	3.63	8832.52	3.19	788.59
Median		138.27	114.16	115.26	0.96	5118.35	9.89	517.79

3.4.2 Limitations and Future Research

Computational benefits of the DT Algorithm become more significant as the problem instances get larger. In particular, our results show that percent deviation from the optimal policies found by the DT Algorithm can be as low as 0.2%, and DT can run up to 3300 times faster than CAO*. However, the computational experiments presented in this chapter is limited only to D-SOSP instances. Although a comparison of DT Algorithm with several other heuristics for general CTP instances is presented in the following chapter, a thorough comparison of heuristic methods, including DT Algorithm, for solving D-SOSP is currently not available in the literature. We leave for the future research, benchmarking and comparing different heuristics for SOSP and D-SOSP.

Chapter 4

A Comparison of Penalty and Rollout-Based Policies for the Canadian Traveler Problem

4.1 Introduction

Approximation algorithms and heuristics for CTP are available in the literature [9, 10, 31]. In this context, [11] made a significant contribution by introducing and evaluating sampling-based (also known as rollout-based) probabilistic algorithms for CTP on both theoretical and empirical fronts. Although they show that a new UCT-based [32] rollout algorithm (called Optimistic UCT) converges to a global optimum, a major limitation of rollout-based approaches in general is that they do not scale well with large instances in terms of execution time. Hence, the need for efficient and effective CTP algorithms arises.

A penalty-based algorithm for CTP generalizes the well-known optimism approach by incorporating a penalty term in the agent's traversal that discourages the agent from traversing edges that are farther away from the termination and/or edges that have high blockage probability. In particular, a penalty-based algorithm calls for successive execution of a deterministic shortest path algorithm with respect to a particular penalty function until the agent's arrival at the termination. One particular penalty-based algorithm called the Distance-to-Termination (DT) Algorithm was evaluated by utilizing

CAO* as a benchmark and it was shown to find high quality policies in very short execution times [28]. One attractive feature of penalty-based algorithms is that they scale quite well in terms of the problem size relative to rollout-based approaches.

Our goal in this chapter is to compare the penalty-based DT Algorithm against four rollout-based ones both in terms of execution time and solution quality for random CTP instances defined on Delaunay and grid graphs. Our purpose is to assess relative merits of these two algorithmic frameworks on an empirical basis. The rest of this manuscript is organized as follows: Section 2 is devoted to formal definition of CTP. Section 3 describes the penalty and rollout-based algorithms. The computational experiments are presented in Section 4, which is followed by a summary and our conclusions.

4.2 Algorithms for CTP

We consider a total of six algorithms for CTP. The first algorithm is Optimism that does not require any rollouts, yet it can be used as a benchmark due to its simplicity and popularity. The next four are the rollout-based methods: Hindsight Optimization, Optimistic Rollout, Blind UCT, and Optimistic UCT. The last algorithm is the penalty-based DT Algorithm (DTA).

4.2.1 Optimism (OMT)

The Optimism Algorithm (OMT) employs a popular technique from robotic motion planning called free-space assumption. The agent assumes that all edges are traversable and calculates the deterministic shortest path and re-calculates it again whenever a blocked edge is encountered. Optimistic policy does not take probabilistic information (in this case blockage probabilities) into account. Within the context of CTP, OMT employs the following navigate-disambiguate-repeat (NDR) strategy:

1. Find the deterministic s, t shortest path in the graph where all the edge weights are set to the edge lengths. That is,

$$w^{OMT}(e) := \ell(e)$$

2. Traverse the path until a node associated with an ambiguous stochastic edge is reached.
3. Since an ambiguous edge cannot be traversed, disambiguate the edge from the current node. Set the blockage probability to zero if the edge has been found to be traversable, and 1 otherwise.
4. Set the current node as the new starting node s and repeat 1 through 3 until t is reached.

Despite its simplicity, Optimism is a common approach for solving both CTP [33] and robotic motion planning problems [34, 35]. Hence, it can be considered as a baseline for evaluating solution quality of CTP algorithms.

4.2.2 Hindsight Optimization (HOP)

Hindsight optimization (HOP) solves a sequence of determinized problems to calculate a policy in a stochastic setting. However, unlike Optimism, HOP uses graph-specific probabilistic information by generating a set of samples from the graph and performs a sequence of actions called *rollouts*. In each rollout, HOP creates a determinized instance of the graph where some edges are blocked and some are traversable according to their blockage probabilities. Next, the algorithm solves a deterministic shortest path problem in each rollout to estimate an average travel cost to determine the next action. The algorithm determines the next course of action by greedily choosing the step that gives the minimum average travel cost estimate. The number of rollouts, denoted by N , is an algorithm parameter. Solution quality is directly proportional to N while run time is inversely proportional. In our experiments, for all rollout-based algorithms, N is fixed to 10,000 which has been shown to provide a good trade-off between solution quality and run time [11].

HOP has been successfully used in various domains such as network control [36] and probabilistic planning [37, 38]. However, as N approaches to infinity, it has been observed that HOP often converges to a suboptimal policy for CTP [11].

4.2.3 Optimistic Rollout (ORO)

In order to address the suboptimality issue of HOP, optimistic rollout (ORO) approach makes a subtle modification to the rollout mechanism [11]. Both algorithms perform N number of rollouts to compute cost estimates to select the next action. However, ORO executes the optimistic policy to assign the distance traveled as the rollout cost. In other words, in ORO rollouts, the underlying deterministic subgraph is hidden to the agent (whereas in HOP, it is revealed to the agent, hence the deterministic shortest path calculations). In practice, the agent traverses the deterministic subgraph while following the optimistic policy, and it re-plans the path whenever a blocked edge is encountered. ORO selects successor edges which give the minimum optimistic policy cost until the termination node is reached.

4.2.4 Blind UCT (UCTB)

Introduced by Kocsis and Szepesvári [32], UCT (Upper Confidence Bounds Applied to Trees) has shown success in sequential decision making problems ranging from multi-armed bandit problems to general Markov Decision Processes [37, 39], including CTP [11]. UCT follows the logic of the previous algorithms and calculates a cost estimate by averaging the cost of N rollouts. Similar to ORO, in every rollout, the underlying subgraph is hidden to the agent. However, how the algorithm chooses the next action during the rollouts is quite different from the previous algorithms. Let b denote the initial belief state of the agent prior to its s, t traversal. Starting from the belief state b , $\sigma = \langle b, b_1, \dots, b_i \rangle$ is called a *belief sequence* consisting of a particular order of belief states. In each rollout, a belief state is added to the sequence until the agent reaches the termination node.

The critical part is how UCT selects a b' amongst the alternative successor states b'_1, \dots, b'_n . This is where the fundamental difference between the previous rollout-based algorithms and UCT reveals itself. In HOP and ORO, each rollout is an independent simulation whereas in UCT, rollouts affect each other to allow exploiting the graph-specific information. In simple terms, to select the next action, UCT biases the selection towards successors that (1) produce low cost estimates and (2) remain unexplored in the previous rollouts. This trade-off between exploitation and exploration is balanced with

respect to what is called the *UCT Formula* below:

$$B \left(\frac{\log R^k(\rho)}{R^k(\rho_i)} \right)^{1/2} - \text{cost}(\rho, \rho_i) - C^k(\rho_i)$$

In the above expression,

- ρ_i denotes the sequence ρ that is extended with the belief state b_i .
- $\text{cost}(\rho, \rho_i)$ is cost of traversing from ρ to ρ_i ,
- $R^k(\sigma)$ denotes the number of rollouts starting with σ among rollouts 1 through k ,
and
- $C^k(\sigma)$ is the average travel cost of rollouts $R^k(\sigma)$.

To avoid the case where $R_k = 0$ that makes the UCT Formula approach to ∞ , the algorithm starts the first m rollouts with visiting ρ 's each successor once. By selecting the ρ_i that maximizes the UCT Formula, UCT optimizes the trade-off explained above.

4.2.5 Optimistic UCT (UCTO)

The UCT Algorithm explained above will be referred to as *Blind UCT* (UCTB) in the following sections. To improve solution quality and speed of convergence, Eyerich et al. [11] modifies UCTB by incorporating the optimistic approach, which they refer to as UCTO. Specifically, UCTO operates as follows:

1. During the rollouts, it breaks ties for unvisited successors by picking the one that gives the lowest optimistic policy cost.
2. It defines $R^k(\sigma)$ and $C^k(\sigma)$ using M additional rollouts for the successor belief states while calculating the cost of belief states using the optimistic policy.

Thus, during the initial rollouts, OMT helps UCTO to select better paths earlier by sensing it during the additional M rollouts. A reasonable number of additional rollouts M is determined empirically, which is taken as 20 in our computational experiments.

4.3 Computational Experiments

This section empirically compares performance of the above six algorithms for CTP instances defined on two different graph types: (1) classical Delaunay graphs on the plane and (2) grid graphs, which are essentially 8-adjacency integer lattices. An example of a Delaunay graph consisting of 20 nodes and 48 edges is shown in Figure 4.1 whereas an example of a 10x10 grid graph is illustrated in Figure 4.2.

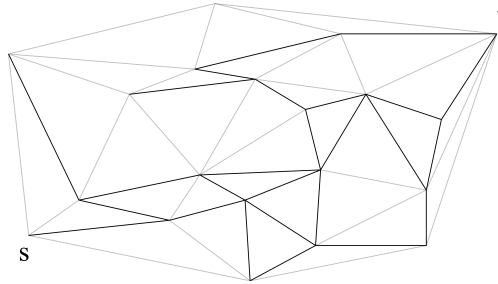


FIGURE 4.1: A Delaunay graph consisting of 20 nodes and 48 edges.

For both graph types, blockage probabilities are sampled from Beta probability distribution parameterized via what we call *sensor accuracy* and denote by λ [10]. Specifically, in any CTP instance, for randomly chosen 50% of the edges, blockage probabilities are sampled from $Beta(4 - \lambda, 4 + \lambda)$ (denoting unblocked edges in reality) and the blockage probabilities of the other 50% of the edges are sampled from $Beta(4 + \lambda, 4 - \lambda)$ (this

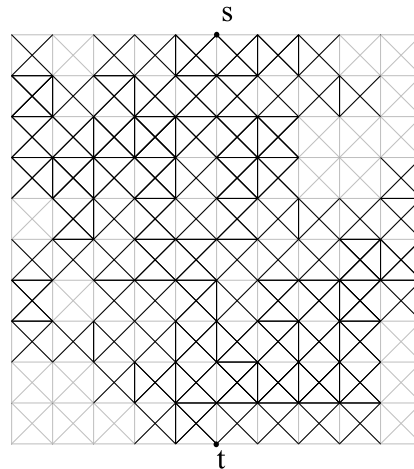


FIGURE 4.2: A CTP instance on a 10x10 grid graph. Blocked edges are represented by bold edges.

time denoting blocked edges in reality). The motivation for introducing the sensor accuracy parameter is to generate meaningful blockage probabilities which are obtained from sensors in practice in general. A real-life application of sensor-obtained blockage probabilities within a probabilistic path planning domain can be found in a U.S. Navy minefield data set called the COBRA data [30].

As λ approaches to 0 (lowest sensor accuracy), the sensor will render “useless” information about the blockage status of graph edges. On the other hand, as λ approaches to 4 (highest sensor accuracy), the sensor will render almost “perfect” information [10]. For each graph size on both Delaunay and grid graphs, we consider two sensor accuracy levels: $\lambda = 2$ and $\lambda = 3$, which we designate as low and high sensor accuracy, respectively. Probability density plots of the respective λ values are shown in Figures 4.3 and 4.4.

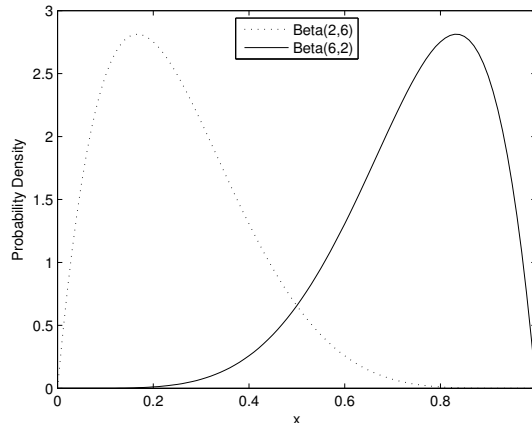


FIGURE 4.3: Blockage probability density plots for $\lambda = 2$ for the Beta distribution.

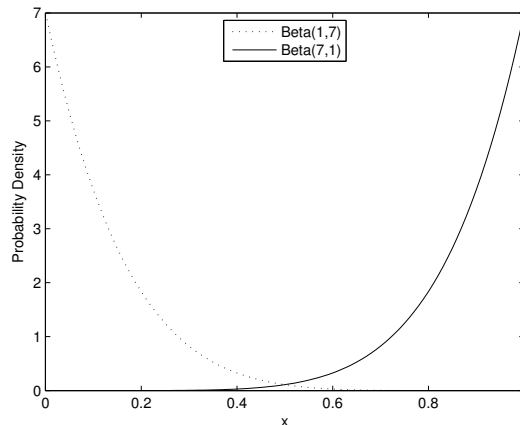


FIGURE 4.4: Blockage probability density plots for $\lambda = 3$ for the Beta distribution.

For both graph types, we consider two parameters for the random CTP instances: graph size and sensor accuracy. We use three different graph sizes for Delaunay graphs: small graphs with 20 nodes, moderate graphs with 100 nodes, and large graphs with 250 nodes. For grid-based graphs, we consider two graph sizes: 10x10 lattices with 420 edges and 20x20 lattices with 1640 edges.

For all parameter combinations, each algorithm is tested on a total of 900 instances for Delaunay graphs with 20 and 100 nodes as well as 10x10 grid graphs: 30 blockage probability realizations each from 30 different graphs. Due to high complexity and excessive run time requirements for larger problems, we perform 100 runs (10 realizations each from 10 different graphs) for both Delaunay graphs with 250 nodes and 20x20 grid graphs. Each graph was modeled to consist of only stochastic edges. The computational experiments were performed on a PC with a quad-core 3.60 GHz processor and 16 GB of memory. All algorithms were implemented in C++.

At this point, a clarification is in order. Definition of CTP calls for minimization of **expected** s, t path length cost. However, for a given CTP instance, computation of this quantity for any algorithm is exponential in the number of stochastic edges and it is prohibitively expensive. Therefore, as in Eyerich et al. [11], we first sample what is called a “weather” from the probability distribution of the stochastic edges to determine the actual blockage status of these edges. Next, based on the outcome of this sampling procedure, we form a deterministic graph where some of these edges are blocked and the others are not. Finally, we execute the algorithm under consideration and find the **actual** travel cost of the s, t path.

4.3.1 Delaunay Graph Results

Experimental results on Delaunay graphs are summarized in Table 4.1. We observe that regarding rollout-based algorithms, UCTO exhibits the best overall performance in general, which is in line with the results of Eyerich et al. [11]. For low sensor accuracy, i.e., for $\lambda = 2$, DTA outperforms all other algorithms, though not by a large margin in the case of rollout-based algorithms. On the other hand, for high sensor accuracy, that is, for $\lambda = 3$, superiority of DTA against OMT and rollout-based algorithms is more pronounced, indicating DTA’s sensitivity to reliable sensor information. In all combinations considered, DTA outperformed OMT by up to 53.4% and UCTO by up

TABLE 4.1: Average cost of algorithms for Delaunay graphs. Lowest policy cost for each parameter combination is denoted in bold. Last two columns show how much DT is better than OMT and UCTO respectively in percentages

λ	Size	Average Distance						Percent Difference	
		OMT	DT	UCTO	UCTB	HOP	ORO	DT-OMT	DT-UCTO
2	20	1938	1546	1551	2024	1593	1582	25.4	0.3
	50	2543	1892	1954	3573	2147	2020	34.4	3.3
	250	2554	1934	2010	4186	2247	2077	32.1	3.9
3	20	2035	1509	1540	2134	1562	1606	34.9	2.1
	50	2630	1715	1854	3564	1899	1937	53.4	8.1
	250	2634	1741	1870	3456	1842	1932	51.3	7.4
	Mean	2389	1723	1797	3156	1882	1859	38.6	4.2
	Median	2549	1728	1862	3510	1871	1935	34.6	3.6
	Std.	316	174	203	874	280	212	11.2	3.0

TABLE 4.2: Average cost of algorithms for grid-based graphs. Lowest algorithms cost for each parameter combination is denoted in bold. Last two columns show how much DT is better than OMT and UCTO respectively in percentages.

λ	Size	Average Distance						Percent Difference	
		OMT	DT	UCTO	UCTB	HOP	ORO	DT-OMT	DT-UCTO
2	10x10	16.1	13.7	14.0	-	13.9	14.1	17.9	2.2
	20x20	31.3	25.9	26.6	-	26.7	26.4	20.7	2.9
3	10x10	16.0	12.9	13.3	-	13.2	13.4	23.7	2.6
	20x20	29.4	24.2	25.0	-	24.6	25.4	21.4	3.2
	Mean	23.2	19.2	19.7	-	19.6	19.9	20.9	2.8
	Median	22.8	18.9	19.5	-	19.4	19.8	21.1	2.8
	Std.	8.3	6.8	7.1	-	7.0	6.9	2.4	0.3

to 8.1%. In particular, the cost of the policies found by DT was better than OMT and UCTO by 38.6% and 4.2% respectively on the average.

In terms of run time, our results show that rollout-based policies do not scale well with large instances, which can be seen in both Figure 4.5 and Table 4.3. On the other hand, one major advantage of penalty-based algorithms is that they scale relatively gracefully with the graph size¹. In all of our tests, DTA ran extremely fast. In particular, whereas UCTO ran in 95.2 seconds on graphs with 250 nodes, DTA ran in merely 0.3 seconds,

¹With at most n successive deterministic shortest path computations, run time of DTA (as well as Optimism) can be seen to be $O(n^2 \log n)$ where n is the number of graph nodes.

TABLE 4.3: Run time averages (in seconds) of algorithms on Delaunay graphs.

Size	OMT	DT	UCTO	UCTB	HOP	ORO
20	0.1	0.1	0.6	1.4	0.6	1.7
100	0.1	0.2	18.5	104.5	10.5	51.5
250	0.2	0.3	95.2	747.3	48.8	326.2

TABLE 4.4: Run time averages (in seconds) of algorithms on grid-based graphs.

Size	OMT	DT	UCTO	UCTB	HOP	ORO
10x10	0.0	0.0	11.6	—	9.1	45.1
20x20	0.1	0.2	117.5	—	70.6	453.6

which is about a 320-fold computational advantage. Of course, this is in addition to a 4.2% improvement in solution quality of DTA over UCTO on the average.

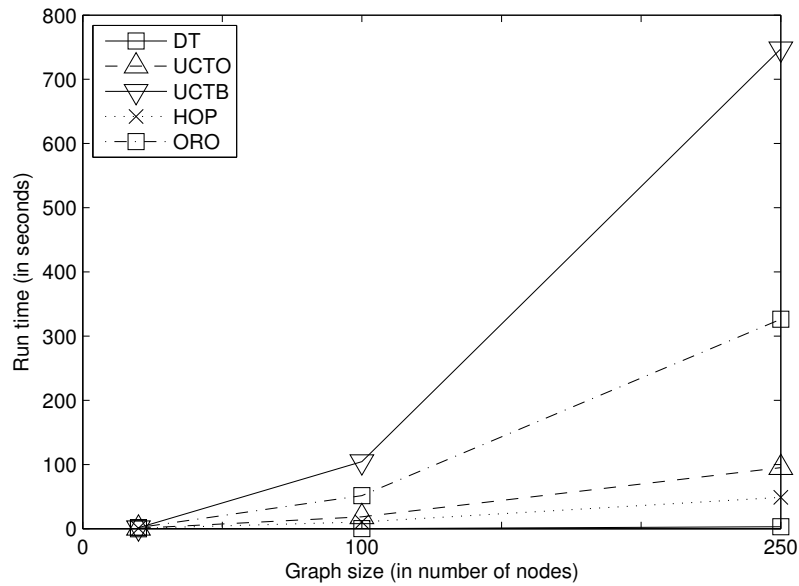


FIGURE 4.5: Average run time as a function of size on Delaunay graphs.

4.3.2 Grid Graph Results

Grid graph results are shown in Table 4.2 and Table 4.4 for solution quality and execution time respectively. Despite the high memory capacity, UCTB failed to yield a solution for grid graphs, hence the “—” mark in the tables. Similar to Delaunay graphs, DTA outperformed OMT by upto 23.7% and UCTO by upto 3.2%. On the average, the cost

of the policies found by DTA was better than OMT by 20.9% and UCTO by 2.8% respectively. In terms of execution time, DT ran about 220 times faster than UCTO on 10x10 grid graphs and about 40 times faster on 20x20 grid graphs. In particular, on 20x20 grid graphs, DT ran in just 0.2 seconds on the average whereas UCTO completed in 117.5 seconds. Solution quality improvement of DTA was even higher with the high sensor accuracy.

4.4 Conclusions and Future Research

4.4.1 Conclusions

This chapter provides a set of computational experiments to compare the penalty-based DT Algorithm against rollout-based algorithms for CTP on random problem instances defined on Delaunay and grid graphs. Our results indicate that DTA runs significantly faster than rollout-based algorithms while providing better policies in general. Run time advantages of DTA are even more pronounced as graph sizes get larger. As for solution quality, DTA outperformed all other algorithms in all combinations for both sensor accuracy levels in our tests, which we believe is quite remarkable especially taking into account how simple it is and how quickly it finds a solution.

4.4.2 Limitations and Future Research

There is one particular issue regarding sensitivity of DTA's solution quality to sensor accuracy. As illustrated in the previous section, relative performance of DTA seems to increase as λ increases for $\lambda \geq 2$. However, in our limited experiments for lower values of λ , performance of DTA took a turn for the worse, this time sometimes being outperformed by even OMT, i.e., the Optimism Algorithm. We suspect that this behavior is related to the specific form of the DT penalty function that seems to require a good amount of separability between densities of blocked and unblocked edge probabilities. This observation, on the other hand, brings up the question whether penalty functions other than DT exist that could potentially result in better policies in the case of poorly performing sensors. We actually conducted limited experiments where we attempted to

improve upon the DT penalty function for specific CTP instances with poor sensor accuracy on a case by case basis using stochastic optimization techniques, and our preliminary results are very encouraging. We leave it to future research to devise efficient methodologies for identification of an appropriate penalty function for a given CTP instance in the case of poor sensor accuracy.

Appendix A

Problem Instances in Simulation Environments C and D

This appendix presents x - and y -coordinates of disk centers and disk marks in the COBRA data set in Simulation Environment C, and the six COBRA-like instances in Simulation Environment D.

TABLE A.1: Center coordinates and marks of COBRA disks.

X	Y	Mark	X	Y	Mark	X	Y	Mark
46.13	39.61	0.0731	50.49	24.26	0.1033	83.62	16.33	0.1165
30.21	54.62	0.1379	56.83	20.50	0.1527	44.87	66.45	0.1668
47.88	34.51	0.1718	40.55	76.93	0.1939	43.43	26.22	0.2575
21.93	53.22	0.3309	69.82	51.65	0.4353	65.64	11.08	0.4412
37.36	29.94	0.4917	29.47	37.21	0.5215	59.42	20.11	0.5418
38.90	57.22	0.5609	32.07	31.37	0.5745	45.71	24.83	0.5831
86.12	15.83	0.5902	52.01	56.80	0.5994	41.14	27.41	0.6200
8.43	74.26	0.6399	37.00	43.89	0.6416	72.53	18.22	0.6527
22.98	40.29	0.6543	70.33	18.61	0.6564	29.78	32.15	0.6566
63.54	24.81	0.1887	64.04	37.65	0.5149	27.00	37.97	0.5280
46.07	71.00	0.5609	65.16	64.01	0.5653	37.36	18.03	0.6108
39.43	70.31	0.6171	75.51	42.83	0.6189	76.11	55.73	0.6405
38.29	44.20	0.6444	28.16	64.10	0.6567	64.55	50.98	0.8515

TABLE A.2: Center coordinates and marks of COBRA-like Instance 1 disks.

X	Y	Mark	X	Y	Mark	X	Y	Mark
23.75	37.63	0.8804	65.31	22.57	0.3669	12.40	83.31	0.2423
86.69	36.95	0.9049	81.09	18.74	0.1917	62.04	34.20	0.1082
18.73	32.54	0.9539	39.09	57.18	0.5187	14.34	59.59	0.4565
16.94	28.37	0.8486	87.64	11.69	0.3810	40.25	36.77	0.5199
10.65	42.96	0.7621	79.20	60.80	0.0325	17.90	68.62	0.6199
62.21	24.77	0.7717	67.61	29.75	0.1414	44.40	28.21	0.1133
68.55	61.27	0.7142	89.76	39.35	0.2766	80.05	28.30	0.0271
10.06	84.25	0.5365	37.88	44.97	0.0108	31.16	35.74	0.3034
37.70	64.10	0.7436	52.64	23.22	0.4036	13.06	53.85	0.3261
44.69	33.62	0.9618	89.85	77.80	0.5890	63.42	26.61	0.2082
58.09	63.85	0.6684	56.70	63.69	0.2837	15.77	26.91	0.1618
63.82	46.92	0.9428	49.16	24.65	0.0986	21.40	48.30	0.1972
10.54	43.81	0.0287	72.13	20.21	0.1118	65.26	41.93	0.0476

TABLE A.3: Center coordinates and marks of COBRA-like Instance 2 disks.

X	Y	Mark	X	Y	Mark	X	Y	Mark
30.98	60.41	0.7944	72.12	36.75	0.2640	83.97	42.34	0.1246
30.95	36.73	0.8726	12.74	44.21	0.1369	10.97	61.46	0.1880
17.86	81.16	0.7772	87.57	78.57	0.2491	13.43	11.50	0.2190
48.93	49.11	0.4122	48.42	66.21	0.4743	24.94	71.51	0.7333
14.69	32.45	0.9290	65.13	35.39	0.1255	84.74	51.76	0.1242
58.16	64.48	0.5905	80.21	24.78	0.2227	75.53	82.68	0.3199
48.73	68.31	0.7618	77.77	17.87	0.2898	26.10	65.97	0.0531
20.08	29.51	0.8261	29.72	15.46	0.1679	63.89	33.08	0.2655
65.05	62.04	0.7553	56.01	42.31	0.3119	73.92	56.05	0.1632
52.23	38.43	0.6611	18.24	16.52	0.0861	27.34	84.19	0.1225
85.70	50.82	0.8955	12.68	29.04	0.2650	56.44	72.07	0.5423
79.36	60.72	0.6665	63.81	24.37	0.2717	16.20	68.20	0.4354
39.60	63.99	0.3790	81.77	82.80	0.1760	63.06	79.72	0.1297

TABLE A.4: Center coordinates and marks of COBRA-like Instance 3 disks.

X	Y	Mark	X	Y	Mark	X	Y	Mark
77.15	65.28	0.7048	27.33	47.70	0.2749	48.92	89.60	0.3360
73.77	61.05	0.5863	35.01	27.29	0.2170	45.44	40.09	0.1764
16.80	42.48	0.9004	53.61	69.05	0.1746	42.67	71.53	0.0910
85.16	59.52	0.6005	27.48	84.77	0.0296	60.11	40.51	0.2268
74.26	18.83	0.7814	58.32	49.09	0.6145	61.03	48.82	0.2949
31.98	63.07	0.9297	30.45	80.46	0.2131	62.31	71.09	0.3926
60.27	46.45	0.6989	71.00	75.81	0.3175	76.02	30.62	0.4548
34.19	76.00	0.9041	73.63	81.94	0.5261	88.72	75.02	0.3445
44.02	89.02	0.6599	32.19	58.00	0.2372	21.48	41.50	0.1210
34.00	54.69	0.7674	51.97	67.70	0.5056	46.25	14.54	0.2130
38.36	69.72	0.3458	81.04	73.33	0.2313	21.55	89.82	0.3643
40.27	21.31	0.6137	40.47	29.63	0.0691	14.69	60.59	0.3980
58.78	15.02	0.1049	73.10	42.19	0.1516	16.16	39.55	0.1375

TABLE A.5: Center coordinates and marks of COBRA-like Instance 4 disks.

X	Y	Mark	X	Y	Mark	X	Y	Mark
87.21	49.82	0.8058	43.87	28.81	0.5957	85.75	42.41	0.0252
10.95	17.43	0.3995	40.90	28.34	0.1219	45.54	44.68	0.2982
74.31	34.23	0.7832	15.12	43.51	0.2227	20.17	21.00	0.2097
18.74	12.83	0.5614	65.29	81.10	0.3898	19.94	35.67	0.2502
60.30	24.76	0.6654	31.48	30.27	0.4503	62.32	15.66	0.4404
25.62	33.70	0.6517	79.54	13.94	0.2005	27.48	27.29	0.2953
58.13	57.05	0.7858	57.80	12.52	0.1078	75.77	11.95	0.1755
58.35	76.47	0.8536	58.20	51.43	0.2258	22.40	68.44	0.1889
43.18	30.81	0.8654	78.15	85.88	0.4560	60.46	73.90	0.2337
58.94	16.54	0.6941	52.49	27.72	0.2839	74.44	41.07	0.1890
37.83	63.85	0.7916	11.73	77.43	0.3284	63.35	46.24	0.1606
62.23	10.74	0.6698	23.97	18.63	0.1335	66.46	19.63	0.3633
64.30	47.21	0.3774	70.95	14.07	0.2097	47.31	70.67	0.2289

TABLE A.6: Center coordinates and marks of COBRA-like Instance 5 disks.

X	Y	Mark	X	Y	Mark	X	Y	Mark
36.32	24.37	0.6272	65.94	42.26	0.4206	36.02	77.85	0.3244
15.40	37.67	0.9315	72.09	60.68	0.4938	81.81	56.67	0.3617
10.86	65.42	0.6859	11.82	67.85	0.1556	39.00	56.25	0.1567
80.21	31.00	0.8989	55.49	22.14	0.1318	26.69	31.36	0.2118
31.69	28.43	0.7812	16.66	34.62	0.1264	71.98	17.37	0.1334
55.33	65.94	0.8342	65.64	72.41	0.6362	49.51	77.68	0.3143
84.54	22.33	0.8894	79.30	27.57	0.1957	60.44	13.13	0.1756
43.49	89.68	0.5151	79.07	68.00	0.0291	29.78	69.28	0.1434
66.28	10.63	0.6631	55.86	16.23	0.2242	50.84	63.61	0.0675
58.99	25.40	0.6119	73.54	27.45	0.5146	46.21	24.52	0.2760
35.95	56.48	0.9661	29.26	81.33	0.2240	41.10	40.04	0.2130
11.53	60.81	0.1599	67.11	53.72	0.4518	58.66	23.42	0.6586
62.29	16.99	0.3447	82.32	88.59	0.1547	29.61	49.01	0.1834

TABLE A.7: Center coordinates and marks of COBRA-like Instance 6 disks.

X	Y	Mark	X	Y	Mark	X	Y	Mark
30.33	49.03	0.8858	39.55	19.38	0.3212	38.96	88.68	0.2409
84.91	78.84	0.9142	11.14	21.92	0.3143	41.64	27.85	0.1364
37.17	51.30	0.9008	80.66	65.64	0.1711	22.88	72.99	0.0836
21.55	49.25	0.6354	10.23	73.50	0.1004	28.06	76.60	0.4545
43.94	35.52	0.6747	26.92	31.11	0.1540	71.94	76.94	0.2831
38.42	55.24	0.9313	82.51	20.42	0.1432	20.65	57.60	0.0234
17.16	37.25	0.9138	31.01	37.74	0.4742	21.38	27.16	0.4543
33.32	65.31	0.8391	54.31	83.29	0.3468	47.02	64.52	0.5741
27.66	41.70	0.5527	36.27	83.66	0.2067	76.04	70.91	0.1244
19.97	73.98	0.7190	56.78	65.64	0.3761	31.08	78.00	0.1804
13.28	54.96	0.7638	65.22	71.59	0.3926	43.78	13.54	0.2463
79.21	81.87	0.7390	11.40	76.11	0.4327	31.89	82.25	0.0871
85.89	30.82	0.2349	38.28	65.61	0.2674	35.68	73.84	0.1055

Bibliography

- [1] D.M. Blei and L.P. Kaelbling. Shortest paths in a dynamic uncertain domain. In *In Proc. IJCAI Workshop on Adaptive Spatial Representations of Dynamic Environments*, Palo Alto, CA, 1999. AAAI Press.
- [2] M. Likhachev and A. Stentz. Probabilistic planning with clear preferences on missing information. *Artificial Intelligence*, 173:696–721, 2009.
- [3] J. Fawcett and P. Robinson. Adaptive routing for road traffic. *IEEE Comp. Graphics Appl.*, 20(3):46–53, 2000.
- [4] J.L. Bander and C.C White. A heuristic search approach for a nonstationary stochastic shortest path problem with terminal cost. *Transp. Sci.*, 36(2):218–230, 2002.
- [5] M. Fiosins, J. Fiosina, J.P. Müller, and J. Görmer. Reconciling strategic and tactical decision making in agent-oriented simulation of vehicles in urban traffic. In *Proc. the 4th Internat. ICST Conf. on Simulation Tools and Techniques*, pages 144–151, 2011.
- [6] D.L. Smith. Detection technologies for mines and minelike targets. *In Proc. SPIE*, 2496:404–408, 1995.
- [7] C.E. Priebe, D.E. Fishkind, L. Abrams, and C.D. Piatko. Random disambiguation paths for traversing a mapped hazard field. *Naval Res. Logist.*, 52:285–292, 2005.
- [8] D.E. Fishkind, C.E. Priebe, K. Giles, L.N. Smith, and V. Aksakalli. Disambiguation protocols based on risk simulation. *IEEE Trans. on Systems, Man, and Cybernetics, Part A*, 37(5):814–823, 2007.
- [9] M. Baglietto, G. Battistelli, F. Vitali, and R. Zoppoli. Shortest path problems on stochastic graphs: a neuro dynamic programming approach. In *In Proc. the*

- 42nd IEEE Conf. on Decision and Control*, pages 6187–6193, Hoboken, NJ, 2003. Wiley-IEEE Press.
- [10] Y. Xu, M. Hu, B. Su, B. Zhu, and Z. Zhu. The Canadian traveller problem and its competitive analysis. *J. Combinatorial Opt.*, 18:195–205, 2009.
- [11] P. Eyerich, T. Keller, and M. Helmert. High-quality policies for the Canadian traveler problem. In *In Proc. the 24th AAAI Conf. on Artificial Intelligence, Atlanta, Georgia*, pages 51–58, Palo Alto, CA, 2009. AAAI Press.
- [12] D. Ferguson, A. Stenz, and S. Thrun. PAO* for planning with hidden state. In *In Proc. the 2004 IEEE Internat. Conf. on Robotics and Automation*, pages 2840–2847, Hoboken, NJ, 2004. Wiley-IEEE Press.
- [13] E. Nikolova and D.R. Karger. Route planning under uncertainty: the Canadian traveller problem. In *In Proc. the 23rd AAAI Conf. on Artificial Intelligence, Chicago, Illinois*, pages 969–974, Palo Alto, CA, 2008. AAAI Press.
- [14] Z. Bnaya, A. Felner, D. Fried, O. Maksin, and S.E. Shimony. Repeated-task Canadian traveler problem. pages 24–30, Palo Alto, CA, 2011. AAAI Press.
- [15] C.H. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theoretical Comp. Sci.*, 84:127–150, 1991.
- [16] D. Fried, S.E. Shimony, A. Bensaaf, and C. Wenner. Complexity of Canadian traveler problem variants. *Theoretical Computer Science*, 487:1–16, 2013.
- [17] B. Bonet. Deterministic POMDPs revisited. In *In Proc. the 25th Conf. on Uncertainty in Artificial Intelligence*, pages 59–66, Palo Alto, CA, 2009. AAAI Press.
- [18] X. Ye and C.E. Priebe. A graph-search based navigation algorithm for traversing a potentially hazardous area with disambiguation. *Internat. J. Oper. Res. and Information Sys.*, 1(3):14–27, 2010.
- [19] V. Aksakalli, D.E. Fishkind, C.E. Priebe, and X. Ye. The reset disambiguation policy for navigating stochastic obstacle fields. *Naval Res. Logist.*, 58:389–399, 2011.
- [20] V. Aksakalli and E. Ceyhan. Optimal obstacle placement with disambiguations. *Ann. Appl. Stat.*, 6(4):1730–1774, 2012.

-
- [21] V. Aksakalli and I. Ari. Penalty-based algorithms for the stochastic obstacle problem. *INFORMS J. on Computing*, 26(2):370–384, 2014.
- [22] D.P. De Farias and B. Van Roy. The linear programming approach to approximate dynamic programming. *Oper. Res.*, 51(6):850–865, 2003.
- [23] H.S. Chang and S.I. Marcus. Approximate receding horizon approach for Markov decision processes: average reward case. *J. Math. Anal. And Appl.*, 286(2):636–651, 2003.
- [24] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2):209–232, 2002.
- [25] N.J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann, Palo Alto, CA, 1980.
- [26] C.R. Chang and J.R. Slagle. An admissible and optimal algorithm for searching and/or graphs. *Artificial Intelligence*, 2:117–128, 1971.
- [27] A. Martelli and U. Montanari. Optimizing decision trees through heuristically guided search. *Comm. ACM*, 21:1025–10039, 1978.
- [28] O.F. Sahin and V. Aksakalli. A fast and effective online algorithm for the Canadian traveler problem. In *In Proc. ICAPS Workshop on Planning and Robotics*, 2014.
- [29] V. Aksakalli. The BAO* algorithm for stochastic shortest path problems with dynamic learning. In *In Proc. the 46th IEEE Conf. on Decision and Control, New Orleans, LA*, pages 6003–6008, Hoboken, NJ, 2007. Wiley-IEEE Press.
- [30] N.H. Witherspoon, J.H. Holloway, K.S. Davis, R.W. Miller, and A.C. Dubey. The coastal battlefield reconnaissance and analysis (COBRA) program for minefield detection. *In Proc. SPIE*, 2496:500–508, 1995.
- [31] E.D. Demaine, Y. Huang, C. S Liao, and K. Sadakane. Canadians should travel randomly. In J. Esparza, P. Fraigniaud, T. Husfeldt, and E. Koutsoupias, editors, *Automata, Languages, and Programming*, volume 8572 of *Lecture Notes in Computer Science*, pages 380–391. Springer Berlin Heidelberg, 2014. ISBN 978-3-662-43947-0.
- [32] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *Proc. ECML*, pages 282–293. Springer, NY, 2006.

-
- [33] Z. Bnaya, A. Felner, and S.E. Shimony. Canadian traveler problem with remote sensing. In *Proc. IJCAI*, pages 437–442. AAAI Press, 2009.
- [34] A. Stentz. Optimal and efficient path planning for partially-known environments. In *Proc. ICRA*, pages 3310–3317, 1994.
- [35] S. Koenig and M. Likhachev. D* lite. In *Proc. AAAI/IAAI*, pages 476–483, 2002.
- [36] E.K.P. Chong, R.L. Givan, and Hyeong Soo Chang. A framework for simulation-based network control via hindsight optimization. In *Proc. 39th IEEE Conf. on Decision and Control*, pages 1433–1438, 2000.
- [37] R. Bjarnason, A. Fern, and P. Tadepalli. Lower bounding Klondike solitaire with Monte-Carlo planning. In *Proc. ICAPS*, pages 26–33, 2009.
- [38] S.W. Yoon, A. Fern, R. Givan, and S. Kambhampati. Probabilistic planning via determinization in hindsight. In *AAAI*, pages 1010–1016, 2008.
- [39] S. Gelly and D. Silver. Combining online and off-line knowledge in uct. In *Proc. ICML*, pages 273–280, 2007.