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

O. Furkan Sahin and Vural Aksakalli

***Abstract*—We consider the Canadian Traveler Problem (CTP) wherein an agent needs to traverse a given graph's edges that may or may not be blocked. The agent can observe the actual status of an edge only upon reaching either end of the edge. To aid its traversal, the agent is given prior blockage probabilities associated with each edge. The goal is to devise an algorithm that minimizes the expected traversal cost between two given nodes. Both penalty-based and rollout-based algorithms have been shown separately to provide high quality policies for CTP. In this study, we compare these 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.**

***Index Terms*—Probabilistic path planning, canadian traveler problem, penalty-based algorithm, rollout-based algorithm.**

I. Introduction

The Canadian Traveler Problem (CTP) is a probabilistic path planning problem that is a representation of a situation Canadian drivers encounter. When a driver reaches an intersection and observes that the road ahead is blocked due to heavy snow, the driver looks for another route. In the graph theoretic analogue of this situation, an agent is given probabilities associated with traversability of each edge in a graph and the goal is to devise a policy[1] that will result in the shortest expected traversal cost between given starting and termination points.

CTP has applications in robot navigation in stochastic domains [1]-[3] adaptive traffic routing [4]-[6] and naval minefield countermeasures [7]-[12]. Along with practical applications, CTP has interesting theoretical properties, which enables it to be cast as a Markov Decision Process (MDP) with exponentially many states (hence its intractability), or Partially Observable Markov Decision Process (POMDP) with deterministic observations. It can actually be shown that CTP belongs to an intermediate set of problems, called Deterministic POMDPs that allow for state uncertainty, meanwhile avoiding noisy observations [13], [14].

An AO*-based optimal algorithm has recently been introduced for CTP that runs several orders of magnitude faster than the classical AO* and value iteration [13]. The new algorithm, called CAO*, improves upon AO* by utilizing two key features: (1) a caching mechanism to avoid re-expanding visited states, and (2) dynamic upper and lower bounds at a node level for further state-space pruning. Optimal algorithms for special cases of CTP have also been studied [14], [15]. Approximation algorithms and heuristics for CTP have been introduced as well [16]-[18]. In this context, Eyerich et al. [19] 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 [20] 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 [21]. 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 study 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 II is devoted to formal definition of CTP. Section III describes the penalty and rollout-based algorithms. The computational experiments are presented in Section IV, which is followed by a summary and our conclusions.

II. CTP Formulation

Let $G=(V,E)$ be an undirected graph. An agent wishes to

Manuscript received December 9, 2014; revised February 26, 2015. This research was supported by The Scientific and Technological Research Council of Turkey (TUBITAK), Grant No. 111M541 and 113M489.

The authors are with the Department of Industrial Engineering, Istanbul Sehir Univ., Istanbul, 34662, Turkey (e-mail: furkansahin@std.sehir.edu.tr, aksakalli@sehir.edu.tr).

[1] The terms solution and policy shall be used interchangeably in this manuscript.

DOI: 10.7763/IJMLC.2015.V5.527

traverse from $s \in V$ to $t \in V$ using edges $e \in E$ for which the following functions are defined:

- Edge length function $\lambda: E \to [0,1]$
- Blockage probability function $p: E \to [0,1]$

We assume that there are two types of edges. First, edges in the subset $E' \subseteq E$ are called *stochastic edges* for which traversability status are unknown, but blockage probabilities are known a priori by the agent in the form of the function *p*. The agent cannot traverse a stochastic edge unless it has been disambiguated and found to be unblocked. Disambiguation is defined as revealing the actual status of a stochastic edge by reaching an end node. Once disambiguated, status of a stochastic edge does not change over the course of traversal. It is further assumed that blockage probabilities of stochastic edges are independent. The second subset $E - E'$ is called the set of *deterministic edges*, which are known to be traversable without any disambiguation requirements. For convenience, blockage probabilities of deterministic edges are defined to be 0. The Canadian Traveler Problem (CTP) is defined as finding the policy that will result in the shortest expected *s*, *t* path length.

III. 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).

A. 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, $w^{OMT}(e) \coloneqq \lambda(e)$
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 [22] and robotic motion planning problems [23], [24]. Hence, it can be considered as a baseline for evaluating solution quality of CTP algorithms.

B. 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 [19].

HOP has been successfully used in various domains such as network control [25] and probabilistic planning [26], [27]. However, as *N* approaches to infinity, it has been observed that HOP often converges to a suboptimal policy for CTP [19].

C. Optimistic Rollout (ORO)

In order to address the suboptimality issue of HOP, optimistic rollout (ORO) approach makes a subtle modification to the rollout mechanism [19]. 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.

D. Blind UCT (UCTB)

Introduced by Kocsis and C. Szepesvari [20], 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 [26], [28], including CTP [19]. 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*, $\sigma = < b, b_1, ..., b_i >$ 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 $b'_1, ..., b'_n$. 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:

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

In the above expression,

- ρ_i denotes the sequence ρ that is extended with the belief state b_i
- $Cost(\rho,\rho_i)$ is the 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 each successor of ρ once. By selecting the ρ_i that maximizes the UCT Formula, UCT optimizes the trade-off explained above.

E. 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, UCTB is modified by incorporating the optimistic approach and the resulting algorithm is referred to as UCTO [19]. 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.

IV. Penalty-Based Algorithms

Introduced by Aksakalli and Ari [8], the notion of penalty-based algorithms for CTP refers to a heuristic framework that involves successive calculation of deterministic shortest paths during the agent's *s*, *t* traversal with respect to the edge weights:

$$w^{PBA}(e) := \lambda^e(e) + 1_{e\in E'} \cdot F(e) \quad (2)$$

where 1 is the indicator function and $F(e) \geq 0$ is an edge penalty function. The idea behind using a penalty function is to discourage the agent from traversing risky edges by assigning them additional weights. In particular, a penalty-based algorithm can be seen as an extension of Optimism that uses $w^{PBA}(e)$ instead of $w^{OMT}(e)$ in calculation of deterministic shortest paths within the NDR strategy. Likewise, Optimism can be thought of as a penalty-based algorithm where $F(e) = 0$ for all $e \in E$.

Subsequent to a series of computational experiments, Aksakalli and Ari [8] advocates utilization of the following penalty function:

$$F^{DTA}(e) := \left(\frac{d_t(e)}{1-p(e)}\right)^{-\log(e-p(e))} \quad (3)$$

where $d_t(e)$ denotes the distance from the edge *e*'s middle point to *t*, hence the name "distance-to-termination" (DT). Observe that $F^{DTA}(e)$ specifically discourages the agent from traversing edges that have high blockage probabilities as well as edges that are far away from the termination node. The penalty-based algorithm that uses $F^{DTA}(e)$ as the penalty function shall be referred to as the DT Algorithm (DTA), which is the particular penalty-based algorithm used in our experiments in comparison against rollout-based ones.

V. 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 Fig. 1 whereas an example of a 10 × 10 grid graph is illustrated in Fig. 2.

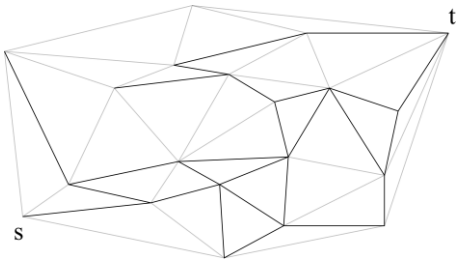

Fig. 1. A Delaunay graph consisting of 20 nodes and 48 edges. Blocked edges are represented by bold edges.

For both graph types, blockage probabilities are sampled from Beta probability distribution parameterized via what we call *sensor accuracy* and denote by λ [18]. 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 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. 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 [29].

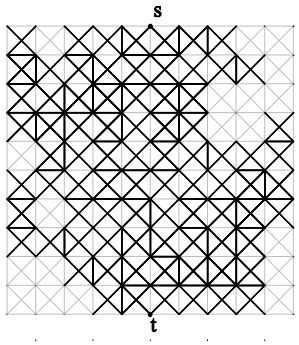

Fig. 2. A CTP instance on a 10 ×10 grid graph. Blocked edges are represented by bold edges.

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 [18]. For each graph size on both Delaunay and grid graphs, we consider two sensor accuracy levels: λ = 2 and λ = 3, which we designate as low and high sensor accuracy, respectively. Probability density plots of the respective λ values are shown in Fig. 3 and Fig. 4.

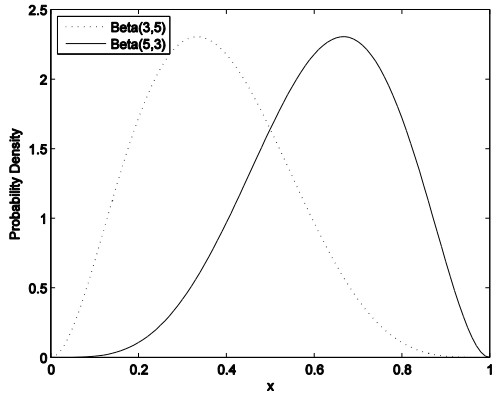

Fig. 3. Blockage probability density plots for λ = 2 for the Beta distribution.

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: 10 ×10 lattices with 420 edges and 20 ×20 lattices with 1640 edges.

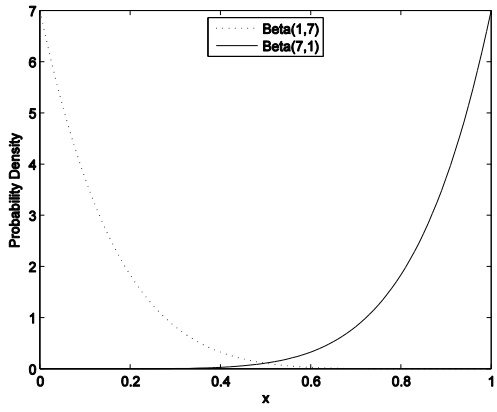

Fig. 4. Blockage probability density plots for λ = 3 for the Beta distribution.

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 10 ×10 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 20 ×20 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 [19], 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.

A. Delaunay Graph Results

Experimental results on Delaunay graphs are summarized in Table I. We observe that regarding rollout-based algorithms, UCTO exhibits the best overall performance in general, which is in line with the results of [19]. For low sensor accuracy, i.e., for λ = 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 λ = 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 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 Fig. 5 and Table III. On the other hand, one major advantage of penalty-based algorithms is that they scale relatively gracefully with the graph size[2]. 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, 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.

B. Grid Graph Results

Grid graph results are shown in Table II and Table IV 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

[2] With 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.

Delaunay graphs, DTA outperformed OMT by up to 23.7% and UCTO by up to 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.

TABLE I: AVERAGE COST OF ALGORITHMS FOR DELAUNAY GRAPHS. LOWEST ALGORITHM 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	OMT	DT	UCTO	UCTB	HOP	ORO	DT vs. OMT (%)	DT vs. UCTO (%)
2	20	1938	**1546**	1551	2024	1593	1582	25.4	0.3
	100	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
	100	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. Dev.		316	174	203	874	280	212	11.2	3.0

TABLE II: AVERAGE COST OF ALGORITHMS FOR GRID GRAPHS. LOWEST ALGORITHM 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	OMT	DT	UCTO	UCTB	HOP	ORO	DT vs. OMT (%)	DT vs. 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	-	19.6	19.9	20.9	2.8
	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. Dev.		8.3	6.8	7.1	-	7.0	6.9	2.4	0.3

TABLE III: RUNTIME 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 IV: RUNTIME AVERAGES (IN SECONDS) OF ALGORITHMS ON GRID 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

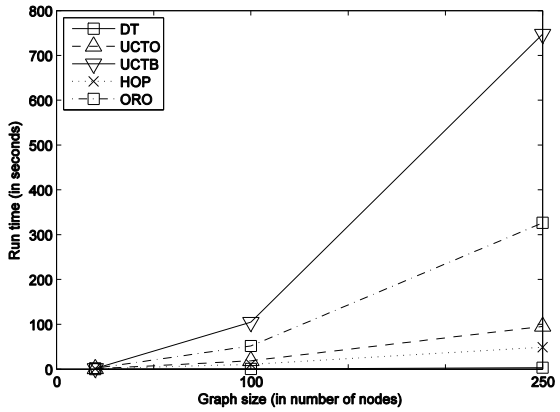

Fig. 5. Average run time as a function of size on Delaunay graphs.

VI. SUMMARY AND CONCLUSIONS

This study 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.

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 $\lambda = 2$. 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 separation between densities of blocked and unblocked edge probabilities. We leave it to future research to devise efficient methodologies for identification of better penalty functions for a given CTP instance in the case of poor sensor accuracy.

ACKNOWLEDGMENT

We thank Patrick Eyerich, Thomas Keller, and Malte Helmert for sharing with us their computer code for the rollout-based algorithms.

REFERENCES

[1] D. M. Blei and L. P. Kaelbling, "Shortest paths in a dynamic uncertain domain," in *Proc. the IJCAI Workshop on Adaptive Spatial Representations of Dynamic Environments*, CA: AAAI Press, 1999.

[2] D. Ferguson, A. Stenz, and S. Thrun, "PAO* for planning with hidden state," in *Proc. the 2004 IEEE International Conf. on Robotics and Automation*, NJ: Wiley-IEEE Press, pp. 2840-2847, 2004.

[3] M. Likhachev, G. Gordon, and S. Thrun, "Planning for Markov decision processes with sparse stochasticity," *Advances in Neural Information Processing Systems*, L. K. Saul, Y. Weiss, and L. Bottou, eds., MA: MIT Press, pp. 785-792, 2005.

[4] D. Dey, A. Kolobov, R. Caruana, E. Kamar, E. Horvitz, and A. Kapoor, "Gauss meets Canadian traveler: shortest-path problems with correlated natural dynamics," in *Proc. the AAMS*, pp. 1101-1108, 2014.

[5] J. Fawcett and P. Robinson, "Adaptive routing for road traffic," *IEEE Comp. Graphics Appl.*, vol. 20, no. 3, pp. 46-53, 2000.

[6] S. Gao and I. Chabini, "Optimal routing policy problems in stochastic time-dependent networks," *Transp. Res.*, Part B-Methodological, vol. 40, no. 2, pp. 93-122, 2006.

[7] V. Aksakalli, "The BAO* algorithm for stochastic shortest path problems with dynamic learning," in *Proc. the 46th IEEE Conf. on Decision and Control*, NJ: Wiley-IEEE Press, pp. 6003-6008, 2007.

[8] V. Aksakalli and I. Ari, "Penalty-based algorithms for the stochastic obstacle scene problem," *INFORMS Journal on Computing*, vol. 26, no. 2, pp. 370-384, 2014.

[9] V. Aksakalli and E. Ceyhan, "Optimal obstacle placement with disambiguations," *Ann. Appl. Stat.*, vol. 6, no. 4, pp. 1730-1774, 2012.

[10] V. Aksakalli, D. E. Fishkind, C. E. Priebe, and X. Ye, "The reset disambiguation policy for navigating stochastic obstacle fields," *Naval Res. Logist.*, vol. 58, pp. 389-399, 2011.

[11] 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, vol. 37, no. 5, pp. 814-823, 2007.

[12] 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.*, vol. 1, no. 3, pp. 14-27, 2010.

[13] V. Aksakalli, O. F. Sahin, and I. Ari, "An AO* based exact algorithm for the Canadian traveler problem," *INFORMS Journal on Computing*, Forthcoming, 2015.

[14] Z. Bnaya, A. Felner, D. Fried, O. Maksin, and S. E. Shimony, "Repeated-task Canadian traveler problem," in *Proc. Annual Symposium on Combinatorial Search*, CA:AAAI Press, pp. 24-30, 2011.

[15] E. Nikolova and D. R. Karger, "Route planning under uncertainty: the Canadian traveller problem," in *Proc. the 23rd AAAI Conf. on Artificial Intelligence*, Chicago, IL, CA: AAAI Press, pp. 969-974, 2008.

[16] M. Baglietto, G. Battistelli, F. Vitali, and R. Zoppoli, "Shortest path problems on stochastic graphs: a neurodynamic programming approach," in *Proc. the 42nd IEEE Conf. on Decision and Control*, NJ: Wiley-IEEE Press, pp. 6187-6193, 2003.

[17] E. D. Demaine, Y. Huang, C. S. Liao, and K. Sadakane, "Canadians should travel randomly," *Automata, Languages, and Programming, Lecture Notes in Computer Science*, J. Esparza, P. Fraigniaud, T. Husfeldt, and E. Koutsoupias, eds., Springer Berlin Heidelberg, vol. 8572, pp. 380-391, 2014.

[18] Y. Xu, M. Hu, B. Su, B. Zhu, and Z. Zhu, "The Canadian traveller problem and its competitive analysis," *J. Combinatorial Opt.*, vol. 18, pp. 195-205, 2009.

[19] P. Eyerich, T. Keller, and M. Helmert, "High-quality policies for the Canadian traveler problem," in *Proc. the 24th AAAI Conf. on Artificial Intelligence*, Atlanta, GA, Palo Alto, CA:AAAI Press, pp. 51-58, 2010.

[20] L. Kocsis and C. Szepesvari, "Bandit based Monte-Carlo planning," in *Proc. the ECML*, NY: Springer, pp. 282-293, 2006.

[21] O. F. Sahin and V. Aksakalli, "A fast and effective online algorithm for the Canadian traveler problem," in *Proc. the ICAPS Workshop on Planning and Robotics*, 2014.

[22] Z. Bnaya, A. Felner, and S. E. Shimony, "Canadian traveler problem with remote sensing," in *Proc. the IJCAI*, AAAI Press, pp. 437-442, 2009.

[23] S. Koenig and M. Likhachev, "D* lite," in *Proc. the AAAI/IAAI*, pp. 476-483, 2002.

[24] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *Proc. the ICRA*, pp. 3310-3317, 1994.

[25] E. K. P. Chong, R. L. Givan, and H. S. Chang, "A framework for simulation-based network control via hindsight optimization," in *Proc. the 39th IEEE Conf. on Decision and Control*, pp. 1433-1438, 2000.

[26] R. Bjarnason, A. Fern, and P. Tadepalli, "Lower bounding Klondike solitaire with Monte-Carlo planning," in *Proc. the ICAPS*, pp. 26-33, 2009.

[27] S. W. Yoon, A. Fern, R. Givan, and S. Kambhampati, "Probabilistic planning via determinization in hindsight," in *Proc. the 23rd National Conference on Artificial Intelligence*, vol. 2, pp. 1010-1016, 2008.

[28] S. Gelly and D. Silver, "Combining online and offline knowledge in UCT," in *Proc. the ICML*, pp. 273-280, 2007.

[29] 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. the SPIE*, pp. 500-508, 1995.

O. F. Sahin has received his B.Sc. degree in manufacturing systems engineering from Sabanci University, Turkey. He is currently a graduate student in industrial and systems engineering at Istanbul Sehir University, Turkey.

V. Aksakalli received his B.Sc. degree in mathematics from Middle East Technical University in Ankara, Turkey, his M.Sc. degree in industrial engineering and operations research from North Carolina State University in Raleigh, NC, and M.Sc. and Ph.D. degrees in applied mathematics & statistics from Johns Hopkins University in Baltimore, Maryland. He spent a number of years in the U.S. working in the industry as an operations research analyst, optimization software engineer, and senior business technologies consultant. He is currently an associate professor of industrial engineering at Istanbul Sehir University in Turkey. His research interests are in stochastic optimization, data mining, and applied probability and statistics.