Proceedings of the 2016 International Conference on Industrial Engineering and Operations Management
Kuala Lumpur, Malaysia, March 8-10, 2016

Disambiguation Points Sampling Heuristic for the Stochastic Obstacle Scene Problem

Serkan Yildirim
Dept. of Computer Engineering
Marmara University
Istanbul, Turkey
serkanyildirim@gmail.com

Vural Aksakalli
Dept. of Industrial Engineering
Istanbul Sehir University
Istanbul, Turkey
aksakalli@sehir.edu.tr

Ali Fuat Alkaya
Dept. of Computer Engineering
Marmara University
Istanbul, Turkey
falkaya@marmara.edu.tr

***Abstract*—The stochastic obstacle scene problem (SOSP) is a challenging probabilistic path planning problem wherein an agent needs to traverse a spatial arrangement of possible obstacles and the agent can disambiguate the actual status of the obstacles (as true or false) en route at a cost. The goal is to find a policy that decides what and where to disambiguate so as to minimize the expected length of the traversal. Traditionally, optimization algorithms for SOSP first perform a lattice discretization of the obstacle field and then consider all (discrete) disambiguation points associated with the possible obstacles. In this work, using an AO*-based exact algorithm for SOSP, we illustrate that a random sampling of a small subset of all the available disambiguation points results in significant computational savings (more than 300-fold in some cases) at the expense of only minor deviations from the optimal expected path length. Our methodology is illustrated via computational experiments involving synthetic data as well as an actual naval minefield data set.**

Keywords—Probabilistic path planning; Canadian traveler problem; Markov decision process; AO* search

I. INTRODUCTION

The Stochastic Obstacle Scene Problem (SOSP) is a probabilistic path planning problem wherein an agent needs to swiftly traverse from one given point to another in an obstacle field through (disk-shaped) regions that are possibly obstacles [1]. At the outset, the agent is given probabilities of each disk being a true obstacle. These probabilities are assumed to be independent from each other. The agent has a limited dynamic disambiguation capability in the sense that when situated on a disk's boundary, the agent has the option to find out actual status of a disk (as a true or a false obstacle), perhaps at a cost. The agent's goal is to find a policy that decides what and where to disambiguate en route so as to minimize the expected length of the traversal.

Optimal policies for SOSP are extremely challenging to compute in a continuous setting except for trivial instances. Therefore, previous work on SOSP has traditionally considered discretization of the obstacle field, giving rise to Discrete SOSP (D-SOSP). In particular, the obstacle field is represented by an 8-adjacency lattice wherein the agent navigates from a given starting vertex s to target vertex t by disambiguating lattice edges intersecting the possibly obstacle disks. Such edges are called stochastic edges, which can be disambiguated only upon reaching an incident vertex. The goal is to find a policy that gives the shortest expected s, t walk. A generalized version of D-SOSP is known as the NP-Hard Canadian Traveler Problem (CTP) in the literature wherein each stochastic edge is traversable or untraversable with a certain probability independent from that of other stochastic edges [2]. Thus, D-SOSP can be seen as a variant of CTP with dependent edge probabilities. CTP is a PSPACE-Complete [3] problem whose space complexity is intractable as well as its computational complexity. This implies

This research was supported by The Scientific and Technological Research Council of Turkey (TUBITAK), Grant No. 115M200.

that D-SOSP is also PSPACE-Complete. Both CTP and D-SOSP can be modeled as a Markov Decision Process, though with exponentially many states. These problems as well as closely related ones have received considerable attention in the literature recently as they have numerous practical applications in robot navigation [4, 5], minefield countermeasures [6, 7, 8], adaptive transportation management [9, 10, 11] and naval minefields [1, 12, 13, 14].

A number of exact and inexact algorithms have been proposed in the literature for D-SOSP and its variants [15, 16, 17, 18, 19, 20]. Of particular interest is the CAO* Algorithm that is an exact method for D-SOSP (and CTP) that is based on the well-known AO* search (CAO* stands for AO* with caching) [21]. CAO* makes use of a state caching mechanism to prevent re-expansion of previously visited states and it prunes the state space using dynamic upper and lower bounds during node expansion and cost propagation steps. CAO* needs considerably shorter execution times to find exact solutions for realistic instances of D-SOSP: CAO* can achieve nearly 800-fold savings in run time when compared against value iteration and about 1800-fold savings against the classical AO* search [21]. CAO* is designed such that it considers all the disambiguation points associated with the possibly obstacle disks when computing the optimal policy. However, consideration of all the disambiguation points creates a huge state space and results in significant increase in both memory and run time requirements. Such consideration of all disambiguation points also implies that the state space is a function of the resolution of the lattice discretization, which is a quite restrictive modeling limitation in practice.

The purpose of this manuscript is to introduce the idea of using a fixed number of disambiguation points per disk (such as 8 or 16 points) to overcome this modeling limitation. These points are sampled randomly from the original full set of disambiguation points, though we assure that they are roughly uniformly distributed around the associated disk. Once these points are sampled, we simply execute the CAO* Algorithm to find the optimal policy limited to these disambiguation points. Even though CAO* is an exact method and it is guaranteed to find the optimal policy, we remark that running CAO* after sampling a subset of all disambiguation points will result in a suboptimal policy in general with respect to the case of using all disambiguation points. Thus, we call this approach the Disambiguation Points Sampling (DPS) Heuristic. In this work, we illustrate that the DPS Heuristic results in significant computational savings (more than 300-fold in some cases) at the expense of only minor deviations from the optimal expected path length. Our methodology is illustrated via computational experiments involving synthetic data as well as an actual naval minefield data set. The rest of this paper is organized as follows: Section 2 formally defines D-SOSP and the CAO* Algorithm. Section 3 introduces the DPS Heuristic. In Section 4, we present computational experiments comparing CAO* and the DPS Heuristic with 8 and 16 disambiguation points on a real-world naval minefield dataset as well as synthetic problem instances resembling this minefield dataset. Our summary and conclusions are presented in Section 5.

II. D-SOSP AND THE CAO* ALGORITHM

In (continuous-space) SOSP, an agent needs to travel from a given location to another through disk-shaped obstacles whose existence is probabilistic. Probabilities of the obstacles are given at the outset and the agent has the capability of disambiguating a disk when it reaches the disk's boundary. Probability of an obstacle is called the obstacle's *mark* and a pre-specified cost is added to expected path length when an obstacle is disambiguated. The goal is to minimize the expected length of the traversal between the starting and target points. In D-SOSP, continuous SOSP is discretized via an 8-adjacency lattice graph. D-SOSP is formally defined as follows: Let $G(V,E)$ be an undirected lattice graph with $s, t \in V$. There is an edge length function $l: E \to \{1, \sqrt{2}\}$ such that horizontal and vertical edges have a length of 1 unit and length of diagonal edges is $\sqrt{2}$. In this graph, some edges, denoted by E', are stochastic whose existence is based on a probability function $\rho: E' \to [0,1)$. Stochastic edges are the edges that intersect disks. For a stochastic edge $e \in E', \rho(e)$ is called mark of that edge that denotes untraversability probability of e. The remaining edges, $E \backslash E'$, are called deterministic edges and they are known to be traversable throughout the agent's traversal. For each edge $e \in E'$, when the agent is at an endpoint, it can disambiguate e to learn if e is traversable. Stochastic edges cannot be traversed until they are definitively known to be traversable. Once disambiguated, the traversability statuses of edges do not change until end of the traversal. Disambiguation has a cost and this cost is added to expected length of the traversal. The goal is to find the s, t walk in this graph that has the shortest expected length. We assume the existence of an s, t path consisting of deterministic edges to avoid infinite expected path length. We also assume that there is a limit K on the number of available disambiguations.

CAO* Algorithm is an AO* search based exact method for D-SOSP (and CTP) that uses the Markov Decision Process (MDP) formulation of the problem [21]. In this formulation, the agent's location along with the current status of the obstacle field (as true/ false/ ambiguous for each disk) comprises a state and the number of remaining disambiguation capabilities corresponds to a stage. CAO* uses AO trees in its calculation of the optimal policy [22]. In this tree, the root node represents the MDP state with the start vertex and an ambiguous status for all disks. CAO* gradually builds a solution tree via alternating expansion and propagation steps. Expansion nodes are represented as OR nodes and children of them are represented as AND nodes that correspond to true and false disambiguation outcomes. CAO* makes use of a value for each node called true label of the node that denotes the actual cost of reaching the target vertex (known as the cost-to-go value in AO* search terminology). CAO* searches the tree by expanding nodes and finding admissible lower bounds that are guaranteed not to overestimate the true label of the node. An admissible lower bound value is the deterministic shortest path length from the

node to termination excluding only untraversable stochastic edges. At the beginning, solution tree consists of the root node and the tree is grown via alternating expansion and propagation steps until calculation of the true label of the root node. At the expansion step, the node to be expanded is found by searching the non-terminal leaf node with the lowest lower bound. Children of this node are added to the tree and their lower bound values are calculated. At the propagation step, expansion node's lower bound calculation is repeated using lower bound values of its children. New lower bound information is propagated up the (partial) solution tree until a node is encountered whose lower bound value is not affected. During the cost propagation step, terminal statuses of nodes are also updated. When a node is marked as terminal, it is not needed to expand that node again.

CAO* uses two important properties of D-SOSP in its execution: admissible upper bounds found by a high-performing D-SOSP heuristic [1] and state overlaps in the AO tree. As a feature, CAO* expands only OR nodes. In addition, an AND node representing direct traversal to the target vertex is added as a child node but only when sum of the lower bound value of the AND node and distance to the parent OR node is smaller than the admissible upper bound value. Admissible upper bounds are also used for dynamic AND node pruning, which is another feature of CAO*. Specifically, when a new AND node is added to the tree, CAO* computes and stores its admissible upper and lower bounds. At the propagation step, CAO* uses these bounds to prune bad AND children of OR parents for which the lower bound is more than the highest upper bound at that node level. This pruning process reduces the size of the solution tree and speeds up the search significantly. When there is only one disambiguation left at the current expansion node, CAO* marks the node as terminal and calculates the true label of that node. This feature of CAO* reduces the overhead cost of individually considering the children of this node in future expansions. One other major feature of CAO* is that it caches previously visited states when encountered at expansion step. Before adding a new AND node to the tree, CAO* looks up the cache if it has been added before. If it exists in the cache, CAO* adds this expanded OR node to the cached AND node as another parent and adds the AND node to the OR node's list of children. This caching mechanism results in significant improvements regarding memory usage as well as execution time.

III. DISAMBIGUATION POINTS SAMPLING HEURISTIC

In SOSP, size of the state space is directly related with number of disambiguation points per disk. Using all possible disambiguation points per disk when discretizing the continuous environment will make the problem more realistic, but it will significantly increase the size of the state space. Defining a fixed number of disambiguation points per disk will allow working at much higher resolutions while keeping the state space under control. We call this approach Disambiguation Points Sampling (DPS-X) Heuristic where X denotes the number of disambiguation points per disk in the heuristic. A simple (discretized) obstacle field with three disks is shown in Fig. 1. Stochastic edges intersecting the disks are shown in bold. With the resolution shown in the figure, there are about 40 disambiguation points associated with each disk, which are the endpoints of stochastic edges that are outside of the associated disk. For instance, in order to sample 8 disambiguation points for each disk as shown in the figure, we first randomly select a regular disambiguation point and then identify one additional point that is about 360/8 = 45 degrees away, and we continue in this fashion until 8 points are identified that are roughly 45 degrees away from each other. Sampling of 16 points can be performed in a similar manner. In the figure, sampled disambiguation points are shown in different colors for each disk. Reducing number of disambiguation points from 40 to 8 for each disk implies roughly 5-fold savings in the number of states at each level in the solution tree in CAO*, which results in substantial memory and computational time savings.

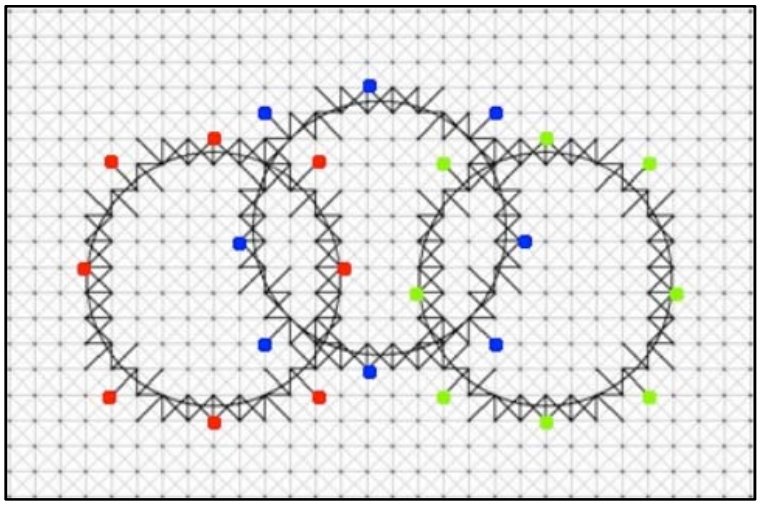

Fig. 1. A simple (discretized) obstacle field with three disks which are sampling of 8 disambiguation points per disk. Sampled points for each disk are shown in different colors.

When solving D-SOSP via CAO*, defining a fixed number of disambiguation points per disk and disambiguating disks only at these points will result in policies that are no longer guaranteed to be optimal. However, as shown in the next section, large instances of the problem can be (suboptimally) solved within very reasonable run times with this approach. The DPS Heuristic has three attractive properties:

- It allows the path planner work with any desired discretization resolution as in this case the resolution has no impact on the number of disambiguation points.
- Once the disambiguation points are sampled, the policy obtained via CAO* is optimal, which distinguishes the DPS Heuristic from other heuristic approaches for the problem that have no optimality bounds or guarantees what so ever.
- It allows the path planner make a trade-off between run time and solution quality as needed in the sense that sampling less disambiguation points will result in faster run times with lower quality solutions and vice versa.

IV. Computational Experiments

This section presents computational experiments for empirical performance assessment of the DPS Heuristic. The application domain we consider is maritime minefield navigation and a specific problem instance we use is a U.S. Navy minefield dataset (called the COBRA data) with 39 disk-shaped possible obstacles [8, 13, 21]. This dataset is shown in Fig. 2 where gray intensity scale of disks reflects marks of each disk, with darker colors indicating a higher mark (that is, a higher probability of being a true obstacle).

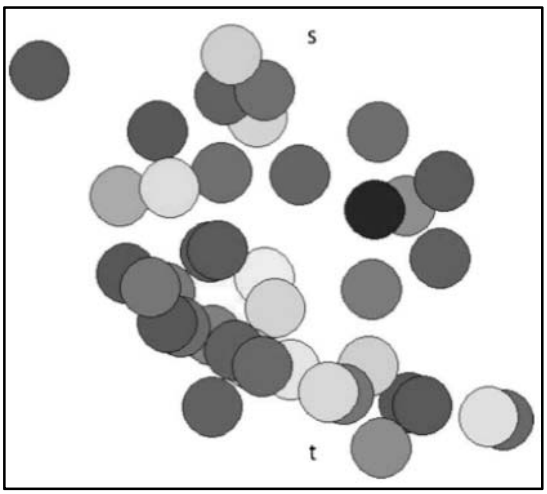

Fig. 2. The COBRA naval minefield dataset. s and t shows start and terminal points respectively. Disks with darker colors indicate a higher mark (that is, a higher probability of being a true obstacle).

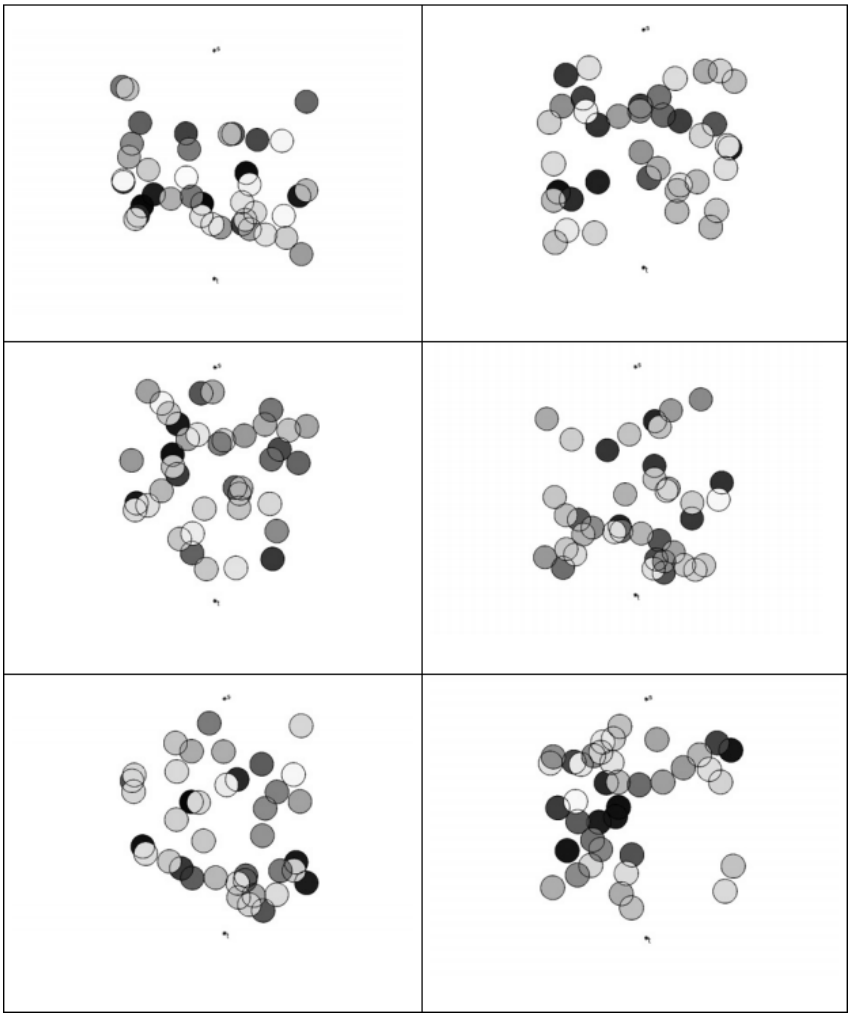

Fig. 3. Six randomly generated COBRA-like datasets that have similar characteristics as the COBRA data. Average of performance on these six datasets is shown in experiments.

In addition to the COBRA data, we make use of six additional randomly generated COBRA-like datasets that have similar characteristics as the COBRA data, as shown in Fig. 3. We present our computational experiment results on the COBRA data and we report average of performance on these six COBRA-like datasets. In our comprehensive experiments, we observed that 8 and 16 are good values for the number of disambiguation points in the DPS-X Heuristic in terms of a reasonable trade-off between solution quality and execution time, giving rise to the DPS-8 and DPS-16 Heuristics respectively.

Comparison of DPS-8 and DPS-16 Heuristics on the COBRA data against CAO* (with full set of disambiguation points) is given in Table 1 and 2 with respect to solution quality and execution time respectively. In these tables, *K* shows the agent's disambiguation limit and *Cost* shows the disambiguation cost that is added to expected path length when the agent performs one disambiguation. In the tables, we report the mean, standard deviation, median and maximum value statistics. In Table 1, we show the solution quality comparison of CAO*, DPS-8, and DPS-16 Heuristics for different *K* and *Cost* values. We compare optimal expected path lengths and present comparative results as percentage differences from optimal expected path lengths. In Table 2, we present comparison of execution times for CAO*, DPS-8, and DPS-16 Heuristics as ratios. As can be seen in Tables 1 and 2, DPS-8 heuristic runs almost 5 times faster than DPS-16 for the COBRA data and, when compared against CAO*, expected path lengths are only 0.25% worse than DPS-16 on the average. Our conclusion from this exercise is that DPS-8 seems to provide a better execution time vs. solution quality trade-off compared to DPS-16.

Average solution quality and execution time comparison of CAO* and the DPS-8 Heuristic are given in Table 3 on the six COBRA-like datasets. As can be seen in this table, DPS-8 gives only 2.15% worse results when compared against CAO*, but runs 112 times faster than CAO* on the average. Remarkably so, for *K*=5 and *Cost*=0, DPS-8 runs in 106 seconds on the average for the six COBRA-like datasets whereas CAO* runs in 35,590 seconds (a 335-fold improvement) while average expected path lengths found by DPS-8 is merely 3.12% worse than that of CAO*. The average run time for DPS-8 is only 41 seconds on the six COBRA-like datasets.

TABLE I. SOLUTION QUALITY COMPARISON OF CAO*, DPS-8 AND DPS-16 ON COBRA DATA

COBRA		**Optimal Exp. Path Length**	**Path Length Difference (in Percentage)**		
K	***Cost***	***CAO****	***DPS-16 – CAO****	***DPS-8 – CAO****	***DPS-8 – DPS-16***
1	0	80.02	0	0	0
	2	82.02	0	0	0
	4	84.02	0	0	0
	6	86.02	0	0	0
	8	88.02	0	0	0
	10	90.02	0	0	0
2	0	75.47	0.79	2.27	1.47
	2	79.47	0.69	0.69	0
	4	81.77	0.56	0.56	0
	6	83.98	0.54	0.54	0
	8	86.18	0.53	0.53	0
	10	88.39	0.51	0.51	0
3	0	74.2	0.99	2.67	1.69
	2	79.27	0.84	0.84	0.01
	4	81.73	0.55	0.56	0
	6	83.97	0.54	0.54	0
	8	86.18	0.53	0.53	0
	10	88.39	0.51	0.51	0
4	0	73.81	1.14	2.89	1.74
	2	79.02	0.42	0.65	0.24
	4	81.56	0.49	0.6	0.11
	6	83.85	0.69	0.69	0
	8	86.12	0.61	0.61	0
	10	88.38	0.53	0.53	0
5	0	73.51	0.59	2.54	1.95
	2	79.01	0.38	0.62	0.24
	4	81.56	0.47	0.58	0.11
	6	83.85	0.69	0.69	0
	8	86.12	0.61	0.61	0
	10	88.38	0.53	0.31	0.22
		Mean	0.49	0.73	0.25
		Std. Dev.	0.29	0.79	0.59
		Median	0.53	0.56	0
		Max.	1.14	2.89	1.95

TABLE II. EXECUTION TIME COMPARISON OF CAO*, DPS-8 AND DPS-16 ON COBRA DATA

COBRA		Execution Time (sec)			Time Ratio	
K	***Cost***	***CAO****	***DPS-16***	***DPS-8***	***CAO* / DPS-8***	***DPS-16 / DPS-8***
1	0	5.95	2.19	1.09	5.48	2.02
	2	5.47	2.13	1.09	5.01	1.95
	4	8.39	2.13	1.09	7.73	1.96
	6	8.04	2.13	1.07	7.51	1.99
	8	7.32	2.16	1.17	6.26	1.84
	10	6.5	2.09	1.15	5.68	1.83
2	0	956.16	50.56	8.2	116.6	6.17
	2	1404.12	65.99	11.44	122.79	5.77
	4	707.33	75.55	11.44	61.83	6.6
	6	1422.32	78.17	11.05	128.77	7.08
	8	1391.18	83.49	11.45	121.55	7.29
	10	1432.72	80.46	10.03	142.84	8.02
3	0	2164.24	59.23	20.59	105.11	2.88
	2	4261.26	117.61	21.53	197.92	5.46
	4	1582.92	95.19	18.5	85.59	5.15
	6	1304.64	84.85	13.3	98.09	6.38
	8	1275.14	84	18.11	70.43	4.64
	10	1250.41	82.11	16.94	73.81	4.85
4	0	3732.22	135.49	25.22	148.02	5.37
	2	4241.33	120.37	17.47	242.78	6.89
	4	2579.95	105.08	14.33	180.04	7.33
	6	2224.39	109.15	17.83	124.79	6.12
	8	2132.37	104.17	17.31	123.19	6.02
	10	1995.17	97.46	14.46	137.98	6.74
5	0	4745.53	132.25	24.16	196.46	5.47
	2	4992.36	166.58	36.65	136.24	4.55
	4	3802.77	136.93	32.38	117.46	4.23
	6	3202.9	129.08	28.76	111.39	4.49
	8	2961.52	111.08	25.22	117.43	4.4
	10	2621.5	96.53	20.26	129.39	4.76
	Mean	1947.54	80.47	15.11	104.27	4.94
	Std. Dev.	1505.24	46.9	9.62	62.38	1.88
	Median	1507.82	84.43	15.7	117.44	5.26
	Max.	4992.36	166.58	36.65	242.78	8.02

TABLE III. SOLUTION QUALITY AND EXECUTION TIME COMPARISONS OF CAO* AND DPS-8 ON SIX COBRA-LIKE DATASETS

COBRA-Like		**Difference (Percentage)**	**Execution Time (sec)**		**Time Ratio**
K	***Cost***	***DPS-8 – CAO****	***CAO****	***DPS-8***	***CAO*/DPS-8***
1	0	0.92	10.01	0.99	10.13
	2	0.91	10.44	0.98	10.69
	4	0.89	10.12	0.98	10.32
	6	0.88	9.94	0.98	10.14
	8	0.87	9.61	0.98	9.85
	10	0.85	9.06	0.98	9.24
2	0	4.08	1214.73	16.31	74.48
	2	3.66	1376.05	18.46	74.56
	4	3.34	1413.53	19.56	72.28
	6	2.96	1609.05	19.64	81.95
	8	2.59	1751.26	15.98	109.57
	10	2.35	1813.29	18.32	98.98
3	0	2.31	6179.67	48.94	126.27
	2	2.12	5249.41	49.86	105.29
	4	2.1	4987.3	48.63	102.55
	6	2.17	4808.78	48.21	99.76
	8	2.28	4260.22	45.07	94.52
	10	2.28	3875.53	42.55	91.08
4	0	2.77	17609.29	77.29	227.85
	2	1.86	12125.1	69.96	173.31
	4	1.88	8897.27	64.02	138.97
	6	1.98	7911.87	57.75	137
	8	2.12	6963.56	54.06	128.8
	10	2.15	5735.81	48.79	117.57
5	0	3.12	35590.07	106.42	334.42
	2	2.19	17621.04	87.67	200.98
	4	2.15	13920.19	75.33	184.78
	6	2.2	13065.81	69.92	186.88
	8	2.3	11911.74	65.5	181.86
	10	2.32	10282.61	61.85	166.25
	Mean	2.15	6674.41	41.2	112.35
	Std. Dev.	0.82	7643.58	29.8	75.2
	Median	2.18	4898.04	48.42	103.92
	Max.	4.08	35590.07	106.42	334.42

V. Summary and Conclusions

In this study, we consider the Discrete Stochastic Obstacle Scene Problem: a challenging stochastic optimization problem that has practical applications in a number of stochastic path planning domains. We describe the CAO* Algorithm which is an exact algorithm for the problem based on the well-known AO* search. CAO* is not polynomial time but it examines a very small fraction of state space and uses less computational resources compared to traditional AO* and value iteration. CAO* accomplishes this by using the special structure of D-SOSP. In particular, CAO* uses a caching mechanism to avoid re-expansion of previously visited states in order to reduce the size of the state space and it performs dynamic state-space pruning using admissible lower and upper bounds at a node level in order to speed up the search.

Classical CAO* considers all available (discrete) disambiguation points to find the optimal policy. This consequently increases the run time of CAO* when a relatively high resolution is used during the discretization step. We introduce the DPS Heuristic for CAO* that (suboptimally) solves D-SOSP by limiting the disambiguation points around disks to a fixed number. In our computational experiments, we use 8 and 16 disambiguation points per obstacle, which give rise to the DPS-8 and DPS-16 Heuristics respectively. We use the actual naval minefield COBRA dataset and six randomly generated COBRA-like datasets to be used in our computational experiments. Our experiments show that DPS-8 runs almost 112 times faster than CAO* and 5 times faster than DPS-16 and the paths it finds are 0.25% worse than DPS-16 and 2.5% worse than CAO* with respect to the optimal expected path length. Our conclusion is that the DPS-8 Heuristic achieves an attractive trade-off between execution time and solution quality by speeding up the search by more than 100-fold on the average in exchange for a slight degradation in solution quality.

References

[1] V. Aksakalli, I. Ari, “Penalty-based algorithms for the stochastic obstacle problem,” Informs J. Comput., Vol. 26, 2014, pp. 370–384.

[2] C.H. Papadimitriou, M. Yannakakis, “Shortest paths without a map,” Theoretical Comput. Sci., Vol. 84, 1991, pp. 127–150.

[3] D. Fried, S.E. Shimony, A. Benbassat, C. Wenner, “Complexity of Canadian traveler problem variants,” Theoretical Computer Science, Vol. 487, 2013, pp. 1–16.

[4] D.M. Blei, L.P. Kaelbling, “Shortest paths in a dynamic uncertain domain,” In Proc. IJCAI Workshop on Adaptive Spatial Representations of Dynamic Environments. AAAI Press, Palo Alto, CA, 1999.

[5] M. Likhachev, A. Stentz, “Probabilistic planning with clear preferences on missing information,” Artificial Intelligence, 2009, Vol. 173, pp. 696-721.

[6] C.E. Priebe, D.E. Fishkind, L. Abrams, C.D. Piatko, “Random disambiguation paths for traversing a mapped hazard field,” Naval Res. Logist., Vol. 52, 2005, pp. 285-292.

[7] D.L. Smith, “Detection technologies for mines and minelike targets,” In Proc. SPIE, 1995, pp. 404-408.

[8] D.E. Fishkind, C.E. Priebe, K. Giles, L.N. Smith, V. Aksakalli, “Disambiguation protocols based on risk simulation,” Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions, Vol. 37(5), 2007, pp. 814–823.

[9] J.Fawcett, P. Robinson, “Adaptive routing for road traffic,” IEEE Comp. Graphics Appl. Vol. 20, 2000, pp. 46-53.

[10] J.L. Bander, C.C. White. 2002, “A heuristic search approach for a nonstationary stochastic shortest path problem with terminal cost,” Transp. Sci., Vol. 36, 2002, pp. 218-230.

[11] M. Fiosins, J. Fiosina, J.P. Muller, J. Gormer, “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, 2011, pp. 144-151.

[12] X. Ye, 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, 2010, pp. 14–27.

[13] V. Aksakalli, D.E. Fishkind, C.E. Priebe, X. Ye, “The reset disambiguation policy for navigating stochastic obstacle fields,” Naval Research Logistics, Vol. 58, 2011, pp. 389–399.

[14] V. Aksakalli, E. Ceyhan, "Optimal obstacle placement with disambiguations," Ann. Appl. Stat., Vol. 6, 2012, pp. 1730–1774.

[15] M. Baglietto, G. Battistelli, F. Vitali, R. Zoppoli, “Shortest path problems on stochastic graphs: a neuro dynamic programming approach,” In Proc. the 42nd IEEE Conf. on Decision and Control, Wiley-IEEE Press, Hoboken, NJ, 2003, pp. 6187–6193.

[16] Y. Xu, M. Hu, B. Su, B. Zhu, Z. Zhu. “The Canadian traveller problem and its competitive analysis,” J. Comb. Optim., Vol. 18, 2009, pp. 195–205.

[17] P. Eyerich, T. Keller, M. Helmert, “High-quality policies for the Canadian traveler problem,” In Proc. the 24th AAAI Conf. on Artificial Intelligence, Atlanta, Georgia, AAAI Press, Palo Alto, CA, 2009, pp. 51–58.

[18] D. Ferguson, A. Stenz, S. Thrun, “PAO* for planning with hidden state,” In Proc. the 2004 IEEE Internat. Conf. on Robotics and Automation. Wiley-IEEE Press, Hoboken, NJ, 2004, pp. 2840–2847.

[19] E. Nikolova, D. R. Karger, “Route planning under uncertainty: the Canadian traveller problem,” In Proc. the 23rd AAAI Conf. on Artificial Intelligence, Chicago, Illinois, AAAI Press, Palo Alto, CA, 2008, pp. 969–974.

[20] Z. Bnaya, A. Felner, D. Fried, O. Maksin, S.E. Shimony, “Repeated-task Canadian traveler problem,” AAAI Press, Palo Alto, CA, 2011, pp. 24–30.

[21] V. Aksakalli, O. Furkan Sahin, I. Ari, “An AO* based exact algorithm for the Canadian traveler problem,” Informs J. Comput., in press.

[22] N.J. Nilsson, Principles of Artificial Intelligence, Morgan Kaufmann, Palo Alto, CA, 1980.

BIOGRAPHY

Serkan Yildirim received his B.Sc. and M.Sc. degrees in Computer Engineering from Marmara University in Istanbul, Turkey. He spent a number of years in the e-commerce industry as a software engineer, after which he started to work as a researcher at TUBITAK. He is currently employed at TUBITAK as a project manager and he is working towards his Ph.D. degree at Marmara University in the Computer Engineering department.

Vural 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.

Ali Fuat Alkaya received his B.Sc. degree in Mathematics with full scholarship from Koc University in İstanbul, Turkey, his M.Sc. degree in Computer Science and Engineering and his Ph.D. degree in Engineering Management from Marmara University, Istanbul, Turkey. Throughout his graduate studies, he worked as a research assistant in Computer Science and Engineering department at Marmara University. After his Ph.D., he worked at Johns Hopkins University as a postdoctoral researcher. He is currently an Associate Professor of Computer Science and Engineering at Marmara University. His research interests are in combinatorial optimization, algorithm design and analysis, heuristics and metaheuristics.