A Resolution Adaptive Algorithm for the Stochastic Orienteering Problem with Chance Constraints

Thomas C. Thayer Stefano Carpin

***Abstract*— We study a stochastic version of the classic orienteering problem where the time to traverse an edge is a continuous random variable. For a given temporal deadline B, our solution produces a policy, i.e., a function that, based on the current position along a solution path and the elapsed time, decides whether to continue along the path or take a shortcut to avoid missing the deadline. The solution is based on a formulation using constrained Markov decision processes to ensure that the deadline is met with a preassigned confidence level. To expedite the computation, a Monte Carlo simulation on an open loop policy is run to determine how to adaptively discretize the temporal dimension and therefore reduce the number of states and the number of optimization variables in the associated linear program. Our results show that the adaptive algorithm matches the performance of the non-adaptive one while taking significantly less time.**

I. Introduction

Orienteering is a classic graph optimization problem defined over graphs where every vertex is associated with a reward and every edge is associated with a cost. A path in the graph collects rewards associated with all the vertices visited, and incurs a cost given by the sum of the costs for all traversed edges. If a path visits a vertex more than once the reward is collected only once, but if an edge is traversed multiple times, the corresponding cost is incurred each time. In the classic formulation of the Orienteering Problem (OP), one is given a graph with a maximum budget B, and the objective is to find a path (also called route in the sequel) in the graph that maximizes the collected rewards with a cost no larger than B.

Significant research efforts have been devoted to studying the OP and its variants, because it is a suitable model to tackle many practical problems related to logistics or robot task assignments. Our recent work in robot assisted precision irrigation delivery [16], [18], [21], [22], [23] brought us to consider the problem we study in this paper. When the OP is used to model a robot servicing different locations spatially distributed in a partially structured environment, the time to traverse an edge is generally not a constant, but rather a continuous random variable. Assuming that the density function is known, a simplistic way to approach this problem would be to assign each edge a deterministic cost equal to the expectation of the random variable, and to reuse any of the existing methods to compute a solution. However, such approach will generate a solution that when executed will very often lead to realizations where the budget is exceeded before the robot reaches the target vertex. In many situations this is unacceptable. For example, overrunning the budget may mean that the robot runs out of power and must be manually recovered for recharging. A more principled solution would be to compute not a route (i.e., a sequence of vertices to traverse), but a *policy* that establishes where the robot should move to next given the current position of the robot and remaining budget. This idea captures the intuition that if one is "running late" while completing the route because traversing some edges took more than expected, it would be advisable to skip some intermediate vertices to make sure the last one is reached before the temporal budget expires. A second aspect to consider is that unless the support of the distributions for all the traversal times is compact for all edges, there will always be a non-zero probability that the final vertex is not reached by the given deadline. Therefore the solution should be formulated with respect to an acceptable failure probability.

Recently, we presented works on the Stochastic OP (SOP) described above that are based on the idea of computing a policy $\pi(v, t)$ over a path defining which vertex one should move to next if at time t it is positioned at vertex v [19], [20]. These works have in common three main concepts: $1)$ the use of an initial path found by reducing the SOP to a deterministic simplification; $2)$ the idea of a composite state space that combines vertices in the initial path with arrival times; $3)$ the computation of a policy by formulating the problem within a constrained Markov decision process (CMDP) framework. In these works, the composite state space necessitates a discretization of time, however the method used was simply to discretize uniformly. In this paper, we present a new method to discretize the temporal dimension, such that we can use a coarser, yet adaptive resolution informed by the statistics we collect about the initial path. We validate this proposed approach by comparison against the original uniform resolution method and we show that it returns equally good solutions, but in significantly less time.

The rest of this paper is organized as follows. Related work is discussed in Section II, while the SOP is introduced in Section III. In Section IV we discuss a fixed resolution approach based on CMDPs, and in Section V we present its adaptive extension. Numerical simulations are given in Section VI and in Section VII we summarize our findings and discuss future work.

T.C. Thayer and S. Carpin are with the Department of Computer Science and Engineering, University of California, Merced, CA, USA.

This material is based upon work that is supported by the USDA-NIFA under award number 2017-67021-25925 and # 2021-67022-33452 (National Robotics Initiative). T. C. Thayer was also partially supported by the NSF under grant DGE-1633722. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the authors and do not necessarily reflect the view of the USDA and NSF.

II. Related Work

The deterministic version of the OP was first formalized in [12], where its NP-hardness was also proven. Later, it was also shown that the problem is APX-hard [2]. Through the years, numerous variants have been introduced, and the reader is referred to the survey [13] for an overview of the many derivative problems. Because of its intrinsic complexity, various approximation and heuristic algorithms have been proposed in literature. Approximation algorithms typically produce solutions that satisfy the budget constraint, but collect only a fraction of the optimal reward. In this context, a c approximation would collect a reward of at least R/c, where R is the reward collected by the optimal solution. To the best of our knowledge, the best approximation algorithm was proposed in [7], which provides a $(2+\varepsilon)$ approximation algorithm with time complexity of $n^{\mathcal{O}(\frac{1}{\varepsilon})}$ where n is the number of vertices in the graph. An approximation algorithm with factor $(1-\varepsilon)$ was also proposed in [8], but is applicable only to planar, complete graphs. Heuristic algorithms have been widely used in practice, especially when solving large problem instances, but heuristics working well in a wide variety of cases have not been found, yet. Our recent works [18], [21], [23] allow us to quickly solve very large instances with tens of thousands of vertices, but are only applicable to a very special type of graph particular to our application (so called *irrigation graphs*). Note that while [21], [23] provide heuristic approaches, [18] provides a provably optimal solution for an even more restricted class of graphs.

While there has been significant research devoted to the OP with deterministic costs and rewards, the stochastic version has received less attention. The SOP was introduced in [5] where the authors introduce uncertainty both in the time to traverse an edge, and in the time to service a vertex. Solutions presented in [5] are either heuristic or applicable only to specific graph topologies. Some versions of the SOP use graphs with deterministic travel times and instead introduce stochasticity with vertex processing times, i.e., the amount of time needed to collect a reward after arriving at some vertex. This case is studied in [14], where the authors give approximation algorithms using both non-adaptive and adaptive (policy driven) techniques. The non-adaptive algorithm provides a constant factor approximation and the adaptive algorithm provides a $\mathcal{O}(\log\log B)$ approximation. The related literature [3] proves a lower bound for the adaptivity gap of $\Omega(\sqrt{\log\log B})$, which comes close to the earlier provided approximation bound, however the authors stop short of providing an algorithm that meets this bound. These works consider maximizing expectation of the collected reward only and, unlike the methods we present here, do not examine risk sensitivity with respect to failure probability.

More recently, [15] studied a variation of the OP where each edge is associated with a survival probability and considered how to deploy multiple agents to collect the maximum reward while ensuring a minimum probability that at least one survives. [10] considers a variant of the SOP where travel times on edges are initially stochastic, but are deterministic after realization with the first traversal of that edge. This provides growing knowledge about the graph, allowing an adaptive policy to be built that will reuse certain edges which guarantee lower cost than what is expected for a new edge. A non-linear chance constrained mathematical program is given in [26] which provides a non-adaptive solution SOP. The authors also introduce the dynamic SOP, whereby the distributions of travel costs for each edge are functions of time rather than static. [11] developed a mixed integer program with sample average approximation and a heuristic method to obtain good paths, however these solutions are also not adaptive. The most similar work to the problem we study in this paper is found in [25] where the authors consider a risk-sensitive approach to the problem formulation with stochastic weights and chance constraints. Their solution is based on a mixed integer linear program formulation but provides an open-loop solution, differently from the solution we proposed that instead adapts online based on actual incurred travel times.

III. Problem Definition

We start by defining the deterministic OP and then introduce the stochastic version considered in [19], [20] and this work. We assume the reader is familiar with basic notions about graphs, such as paths, tours, cost of a path, etc.

1) The Deterministic Orienteering Problem: Let $G = (V, E)$ be an undirected, fully connected graph, $c : E \to \mathbb{R}^+$ be an edge cost function, and $r : V \to \mathbb{R}^+$ be a vertex reward function. Given $v_s, v_g \in V$ and $B \in \mathbb{R}^+$, determine a path $\mathcal{P}$ starting at v_s and ending at v_g which maximizes the sum of collected rewards and whose cost does not exceed B. The cost of a path $C(\mathcal{P})$ is the sum of the costs for all traversed edges, with costs accrued every time an edge is traversed. The sum of collected rewards $R(\mathcal{P})$ is the sum of the rewards for all visited vertices, but with each vertex contributing only once, i.e., if visited multiple times, a vertex's reward is counted only once.

The version of the problem we introduced specifies both the start and goal vertices, which may be coincident (requiring then to return a tour). In some instances the problem is defined without specifying the start and goal vertices. Nevertheless, even this "simpler" version is NP-hard. In the following we focus on the above definition, and later on it will become evident how our findings generalize to the case where one can pick the first and last vertex on the path. Moreover, without loss of generality we can assume that the graph G is complete. If this is not the case, additional edges can be added with cost equal to shortest path between the vertices they connect.

2) Path Policy: Let $\mathcal{P}$ be a path in G and let $v_1, v_2, \ldots, v_n$ be the sequence of all n vertices along $\mathcal{P}$. For $v_i \in \mathcal{P}$ we define $\mathcal{S}(v_i) = \{v_{i+1}, v_{i+2}, \ldots, v_n\}$, i.e., $\mathcal{S}(v_i)$ is the set of vertices following v_i in $\mathcal{P}$ and for convenience we define $\mathcal{S}(v_n) = \varnothing$. Given a path $\mathcal{P}$, a path policy π is a function defined over $\mathcal{P} \times \mathbb{R}^+ \to \mathcal{P}$ such that for each $v_j \in \mathcal{P}$ and each $t \in \mathbb{R}^+$ we have that $\pi(v_j, t) \in \mathcal{S}(v_j)$. In essence, for

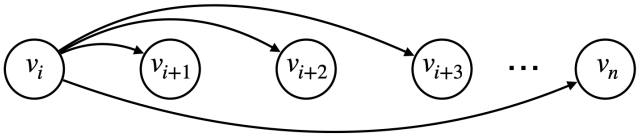

Fig. 1: Upon visiting a vertex v_i, the path policy π defines which successive vertex one should move to next for the current value of t. The possible next vertices include any further along in the path $\mathcal{P}$, such as $v_{i+1}, v_{i+2} \dots v_n$. By skipping ahead, for example passing over v_{i+1} and directly visiting v_{i+2}, one can decrease the cost to reach the last vertex along the path, although doing so will decrease the overall collected reward.

every t, $\pi(v_j, t)$ maps v_j onto one of the following vertices along the path. The reason to introduce path policies is to formalize the idea of taking *shortcuts* along a path solving an instance of the OP with random travel times along the edges. Assuming an agent starts moving along the path at time $t = 0$, the path policy introduces a formal way to skip some vertices along the way based on the current time and position. In particular, if the objective is to reach the last vertex before the temporal deadline B, a path policy π can be defined to skip vertices when the time t is approaching B (see Figure 1).

3) The Stochastic Orienteering Problem: Let G, v_s, v_g, r and B be defined as above. For every edge $e \in E$, let f_e be a probability density function (pdf) with positive support and finite expectation. Every time edge $e = (v_i, v_j)$ is traversed, the incurred cost is not constant, but rather is a random variable $c_{i,j}$ whose pdf is f_e. For a path $\mathcal{P} = \{v_1, \dots, v_n\}$ and a path policy π, an agent starts at time $t = 0$ in vertex v_1 and moves to $v_i = \pi(v_1, 0)$ arriving at time t_i where t_i is a random variable with pdf f_{v_1,v_i}. Once in v_i, the agent moves to $v_j = \pi(v_i, t_i)$ arriving at time $t_i + t_j$, where t_j a random variable with pdf f_{v_i,v_j}. The process then continues until the agent arrives at the last vertex in the path v_n. In this case both the cost to complete the path and the reward collected along the path are random variables. In particular, we indicate $C_{\mathcal{P},\pi}$ the random variable for the cost to complete path $\mathcal{P}$ following policy π and $R_{\mathcal{P},\pi}$ the random variable for the reward collected. For a given failure probability P_f, the SOP asks to determine a path $\mathcal{P}$ and a path policy π that maximizes the expected sum of rewards $\mathbb{E}[R_{\mathcal{P},\pi}]$ and such that $\Pr[C_{\mathcal{P},\pi} > B] \leq P_f$.

For a given path $\mathcal{P}$ and path policy π it is always that $\mathbb{E}[R_{\mathcal{P},\pi}] \leq R(\mathcal{P})$ because π can only skip vertices along the path. The deterministic OP is evidently a special case of the SOP, and therefore the latter is NP-hard, too.

IV. A Fixed Resolution Algorithm

This section describes the basic method to obtain a path policy π, which was introduced in [19] and used in [20].

For now, we assume that a path $\mathcal{P}$ in G is given. An agent moving along the path $\mathcal{P} = \{v_1, \dots, v_n\}$ following a path policy π moves from vertex to vertex, and when at vertex v can move to any of subsequent vertices found in $\mathcal{S}(v)$. The time to make this transition is characterized by the pdf associated with the edge between the two vertices. Thus, this transition sequence can be formalized by a Markov Decision Process (MDP) with a suitably defined state space. We assume the reader is familiar with MDPs and refer to [4] for a comprehensive introduction. An MDP is defined as $\mathcal{M} = \{S, A, \Pr, r\}$ where S is the set of states, A is the set of actions, $\Pr$ is the transition kernel, and r is the reward function associated with every state/action pair. [1] For the SOP from Section III, the MDP can be defined as follows.

- $S = V \times \mathbb{T}$, where V is the set of vertices in $\mathcal{P}$ and $\mathbb{T}$ is a time discretization with step Δ. $\mathbb{T}$ is a collection of k successive time intervals starting with $t_0 \in \mathbb{T}$ and continuing through $t_j = [j\Delta, (j+1)\Delta)$. The composite state (v_i, t_j) represents the agent arriving at vertex v_i during the time interval associated with t_j. In the following, we say time t_j for the whole interval.
- For each state (v, t) the action set is $\mathcal{S}(v)$.
- The transition kernel $\Pr$ defines the probability the next state is (v_j, t_k), assuming that action a is executed from state (v, t_i), indicated as $\Pr((v, t_i), a, (v_j, t_k))$. The action a is a state in $\mathcal{S}(v)$ indicating the agent will move to v_j next. Therefore the transition probability is 0 for all states (v_i, t) with $i \neq j$. For states of the type (v_j, t_k) where $t_k \leq t_i$, the probability is 0 because the agent cannot go back in time. For the remaining states, $\Pr((v, t_i), v_j, (v_j, t_k)) = \int_{t_i\Delta}^{(t_i+1)\Delta} [F(\Delta(t_k + 1) - \xi) - F(\Delta t_k - \xi)] d\xi$ where F is the cumulative function of the pdf f associated with the edge from v to v_j. This integral can be computed numerically off-line.
- The reward function r for $((v, t), v_j)$ is $r(v)$, i.e., the reward associated with v in G.

For the defined MDP, we opt for an undiscounted reward function while ensuring the reward remains bounded using a *sink* state and a *loop* state. The sink state s_s represents the situation where the elapsed time is higher than the temporal budget B, and therefore the chosen discretization step Δ is chosen such that there will be $N = \lceil \frac{B}{\Delta} \rceil$ time intervals in $\mathbb{T}$. The transition kernel $\Pr$ is accordingly extended so that for each vertex v_i, $\Pr((v_i, t_k), v_j, s)$ is the probability that v_j is not reached before the temporal deadline B. The action set of the sink state s_s consists of a single action a_s leading to s_l with probability 1 and has an associated reward $r(s_s, a_s) = 0$. The loop state s_l absorbs all possible runs of the MDP and bounds the cumulative reward. The action set of the loop state s_l consists of a single action a_l looping to itself (hence the name) with probability 1 and has an associated reward $r(s_l, a_l) = 0$, meaning once entered no more rewards can be accrued. Finally, for all states of the type (v_n, t_i) (recall that v_n is the last vertex along the path), we add an action a_l leading to the loop state s_l with probability 1. Figure 2 illustrates the structure of the MDP we just defined.

A policy for the MDP is a function $\pi : S \to A$ mapping actions onto states, and in the proposed structure a policy

[1] With a slight abuse of notation, r refers to both the reward function in G and the reward function in $\mathcal{M}$ because the latter is derived from the former.

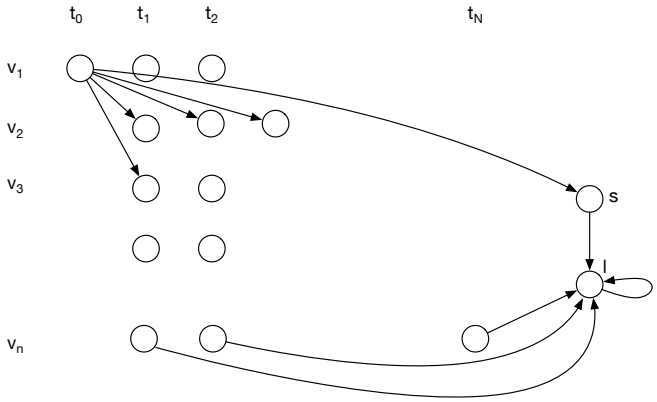

Fig. 2: The MDP states can be imagined as arranged on a grid with vertices (rows) and arrival times (columns). Here, arrows are depicted for some of the transitions with non-zero probability. From state (v_1, t_0) it is possible to go to any of the following states, and when moving towards a specific vertex, the arrival time can be any of the times $t_i > t_0$ because of the random nature of edge traversal time. Reaching a vertex after the temporal deadline B has passed is modeled as a transition to s_s. Once v_n is reached, a deterministic transition is made to s_l.

for the MDP is a path policy for the path $\mathcal{P}$. With this MDP structure, the probability of reaching s_l is 1 under any policy π. Therefore, we can consider the following undiscounted reward function for some π and start state (v_0, t_0),

$$\mathbb{E}[R_{\mathcal{P},\pi}] = R(\pi) = \mathbb{E}\left[\sum_{i=1}^{\infty} r(X_i, \pi(X_i))\right]$$

where X_i is the random variable for the state at time i and the expectation is taken with respect to the probability distribution induced by π. This expectation exists and is finite because the state s_l will be reached with probability 1 within a finite number of transitions and no more rewards will be accrued. The MDP formulation, however, is unsuited to solve the SOP we formerly defined, which constrains the probability of exceeding the temporal deadline B to P_f. Therefore, we use a constrained MDP (CMDP), which aims to maximize a reward function while ensuring bounds in expectation for other costs [1]. Fitting the definition for a CMDP, a cost function is applied to each state/action $d : S \times A \to \mathcal{R}^+$ such that it is 0 everywhere, except for the state/action pair (s_s, a_s) where it is 1. A CMDP defined this way can be solved through the following linear program, where the optimization variables ρ are defined over the set of state/action pairs $S \times A$ and β is a function that is 1 for the start state (v_1, t_0) and 0 everywhere else.

$$\max_{\rho} \sum_{(x,a)\in S\times A} \rho(x,a) r(x,a)$$

$$\text{s.t.} \sum_{(x,a)\in S\times A} \rho(x,a) d(x,a) \leq P_f$$

$$\sum_{y\in S} \sum_{a\in\mathcal{S}(y)} \rho(y,a)(\delta_x(y) - \mathcal{P}^a_{yx}) = \beta(x) \;\; \forall x \in S \setminus \{l\}$$

$$\rho(x,a) \geq 0 \quad \forall (x,a) \in S \times A.$$

The linear program has a solution if and only if a policy π can be found that satisfies the constraint on the cost, and is uniquely defined by the solution vector ρ. The reader is referred to [6], [9], [17] for a detailed discussion about this approach. The optimization variables $\rho(x, a)$, called occupation measures, correspond to the following:

$$\rho(x,a) = \sum_{i=1}^{\infty} \Pr[X_i = x, A_i = a]$$

where X_i is the random variable for the state at time i and A_i is the random variable for the action at time i. In this particular CMDP structure, $\rho(x, a)$ can be thought of as the probability of encountering the state/action pair (x, a), and therefore for any π the probability of entering each state can be determined. We make use of this fact by setting P_f as the bound for the sum of costs over $\rho(x, a)d(x, a)$, which is equal to the probability of entering the sink state s_s and violating the budget constraint. Thus, we arrive at the following theorem, which we proved in [19].

Theorem 1: If the linear program admits a solution, then the associated policy π fails to reach the last state v_n within time B with probability at most P_f. □

Following the CMDP based formulation is an algorithm to solve an instance of the SOP introduced in Section III.

1) Create an instance of the deterministic OP, assigning to every edge e the expected travel cost $\mathbb{E}[f_e]$.
2) Solve the deterministic OP with any existing method and let $\mathcal{P}$ be the returned path.
3) Use $\mathcal{P}$ to build and solve the CMDP described above and return π.

The quality of the solution is dependent on the algorithm used in step 2 to solve the deterministic OP. For small problem instances one could obtain an exact solution using the standard mixed integer program to solve the OP [13], or a heuristic or an approximated method for larger problem instances. The difficulty of this approach is that the uniform discretization used to build $S = V \times \mathbb{T}$ is wasteful, as it uses the same resolution for states (v, t) with very small probability of being reached as well as those with high probability. In regions where there are many states with high probability, one should use a discretization with a smaller Δ to determine policies and this would increase the size of the state space and ultimately the number of optimization variables in the linear program to solve the CMDP. We tackle this problem in the next section.

V. An Adaptive Resolution Algorithm

In this section we propose a refinement of the previous algorithm that performs an adaptive discretization of the temporal dimension, as opposed to the uniform one we introduced. As the time to traverse an edge is a continuous random variable, it follows that the time when a vertex is reached is a continuous random variable, too. Ideally one would like to compute a policy of the type $\pi(v, t)$, for $t \in \mathbb{R}$. A continuous time policy can be approximated with a discretized policy and the approximation is better as Δ

shrinks. However, as we formerly discussed, this increases the computation time. The idea, instead, is to allocate a more fine grain time subdivision in *high density* temporal regions and a more coarse one in *low density* temporal regions. Here, with temporal region for a given vertex we mean the possible distribution of times when the vertex is reached. The algorithm is sketched in the following.

1) Create an instance of the deterministic OP assigning to every edge e the expected travel cost $\mathbb{E}[f_e]$.
2) Solve the deterministic OP with any existing deterministic method and let $\mathcal{P}$ be the returned path.
3) Simulate the path $\mathcal{P}$ for $\mathcal{K}$ trials and for each vertex record the $\mathcal{K}$ times when it was reached.
4) For each vertex consider a temporal split in which $\mathcal{L}$ times of arrival are in the same temporal segment.
5) Solve the CMDP obtained building the state space where every vertex is paired with its associated temporal split from in step 4, returning π.

Steps 3 and 4 deserve some additional explanation. In step 3, the path $\mathcal{P}$ produced by the deterministic orienteering algorithm is repeatedly executed without considering any policy, i.e., all vertices in $\mathcal{P}$ are sequentially traversed from the first to the last, without considering the temporal deadline B. During this process, for every vertex, we log the arrival time and we can therefore numerically approximate the temporal distribution of arrival times and its spread (see top panel in Figure 3.) These temporal distributions are then used in step 4 where, for each vertex $v \in \mathcal{P}$, we build a tailored temporal discretization based on the distribution collected in step 3. In the algorithm described in Section IV, vertex v is combined with each of the N uniform time intervals in $\mathbb{T}$ and this gives S (plus of course the sink state s_s and the loop state s_l). In the adaptive case, for each vertex v we build a vertex-dependent temporal discretization $\mathbb{T}_v$ in which every segment includes $\mathcal{L}$ of the samples collected in step 3. The number of intervals in this case is therefore $N = \lceil \frac{\mathcal{K} \leq B}{\mathcal{L}} \rceil$ (Only samples where the arrival time is less than or equal to B are considered, since arrival times larger than B are encompassed by s_s). Two special segments are created at the beginning and end, i.e., the first temporal segment in $\mathcal{T}_v$ starts at time 0, and the last one ends at time B.

While it is useful to consider the distribution of arrival times to adaptively discretize the temporal dimension of the state space, this approach has a fatal flaw. It considers the arrival times produced by simulation the original path, rather than actual arrival times of the policy. Thus there can be instances where a state has a very coarse time discretization but the resulting policy directs an agent to this state with a high probability. For example, a policy might direct an agent to always skip a vertex, so the next vertex is visited with a time distribution that is much different than expected. Additionally, there can be value (in the form of increased expected reward) in a state space that limits the size of its largest time intervals, as smaller intervals are more accurate. There is no way of knowing what the distribution of arrival times will be for a certain policy without first computing the

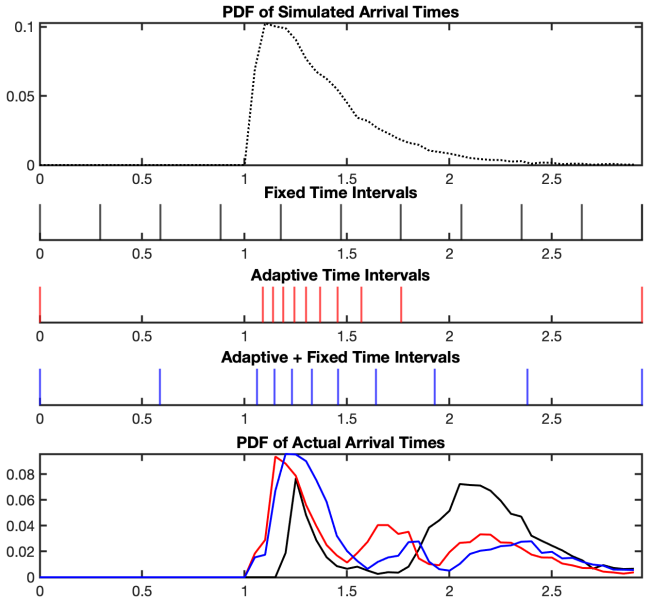

Fig. 3: *First*: distribution of arrival times at a certain vertex v_i following strictly $\mathcal{P}$ ($\mathcal{K} = 10000$). *Second*: time discretization built by the algorithm using a fixed-resolution approach with 10 intervals. *Third*: time discretization built using the adaptive-resolution approach placing $\mathcal{L} = 1000$ samples in each of the 10 intervals. *Fourth*: time discretization built by combining the adaptive-resolution approach ($\mathcal{K} = 10000$) with evenly spaced data ($\mathcal{K} = 10000$) obtaining a total of $\mathcal{L} = 2000$ samples in each of the 10 intervals. *Last*: distribution of arrival times at vertex v_i following policies computed with each type of time discretization. Note that each discretization has the same number of intervals.

policy, therefore it makes sense to generalize our estimate by augmenting the simulated arrival times with evenly spaced data on the interval $0 < t < B$. This essentially combines the fixed discretization method in Section IV with the adaptive time method described above. Figure 3 shows an example of the uniform, adaptive, and combined discretizations. Once the set $\mathbb{T}_v$ is built for every vertex, the CMDP can be built as in the fixed-resolution algorithm and solved.

An interesting aspect discussed in the next section is that by using an adaptive approach one can reduce the number of elements in each of the sets $\mathbb{T}_v$ while essentially keeping the same performance as in the fixed-resolution algorithm. Here, by same performance we mean that in expectation the two algorithms both collect the same reward while ensuring that the probability of reaching v_n after B is less than P_f. However, the combined fixed-adaptive resolution algorithm is much faster because it has a much smaller state space.

VI. Results and Discussion

In this section we present an overview of results obtained simulating the methods described earlier. Vertices of the graph G are obtained sampling the unit square with a uniform distribution, and edges are added to make a complete graph. Each vertex is associated with a constant reward sampled from a uniform distribution over the interval $[0, 1]$. The stochastic travel time between vertices is obtained as follows. Let $d_{i,j}$ be the Euclidean distance between v_i and v_j, and

$0 < \alpha < 1$. Then, the travel distance along edge (v_i, v_j) is

$$\alpha d_{i,j} + \mathcal{E}\left(\frac{1}{(1-\alpha)d_{i,j}}\right)$$

where $\mathcal{E}(\lambda)$ is a random sample obtained from an exponential distribution with parameter λ. As per the properties of the exponential distribution, it follows that the expected cost of the random variable associated with edge (v_i, v_j) is $d_{i,j}$ and the variance is $((1-\alpha)d_{i,j})^2$. This shifted exponential cost distribution is useful for modeling robotic movement (a common application of stochastic orienteering), which requires a minimum amount of time but may be significantly longer. Other distributions are not considered, however any with strict positive support may be used.

The fixed-resolution, adaptive-resolution, and combined fixed-adaptive algorithms start computing a solution to the deterministic OP using an existing algorithm. For problem instances with less than 25 vertices, we use an exact solver based on a mixed integer program formulation, while for larger instances we use the the S-algorithm heuristic described in [24] due to its relative speed and robustness. Both algorithms can handle versions of the problem where the start and end vertices are assigned or not. In either case, the output is a path $\mathcal{P}$ whose expected length is smaller than or equal to B. However, It is worth recalling that if one were to follow all vertices in the path without using a path policy π, the temporal deadline B would be often missed, and for the setup we described this happens roughly half the time.

The initial orienteering path $\mathcal{P}$ remains fixed for a given set of parameters across each of the methods introduced in Sections IV and V so that a fair comparison can be made. The fraction of collected reward is $\frac{\mathbb{E}[R_{\mathcal{P},\pi}]}{R(\mathcal{P})}$, or the expected reward collected by the policy divided by the total reward collected by the deterministic orienteering path. In every scenario, expected rewards account for failures as well, where rewards stop accruing when the budget is exhausted (no extra penalty is incurred since failures are already limited by P_f). Afterward, the route is discarded, a new graph G is generated, and the processes is repeated until the methods have been compared over 10 instances. The results computed using a particular set of parameters are averaged for all routes. We show reward results where the number of time steps is varied while the length of $\mathcal{P}$ is fixed (Figure 4), and results where the length of $\mathcal{P}$ is varied while the number of time steps is fixed (Figure 5). We also show the average computation times for each experiment in Figure 6.

Some patterns begin to emerge when analyzing how the number of time intervals $|\mathbb{T}|$ changes the effectiveness of each discretization method, seen in Figure 4. Firstly, we see that the adaptive-resolution method performs favorably compared to the fixed-resolution method when the number of time steps is lower, however when $|\mathbb{T}|$ increases it becomes worse. This is because the adaptive-resolution is based on the arrival time distribution of the original path, without considering shortcuts, and therefore lacks flexibility normally afforded to path policies. Since Δ gets smaller as $|\mathbb{T}|$ gets larger, fixed intervals become increasingly useful. When

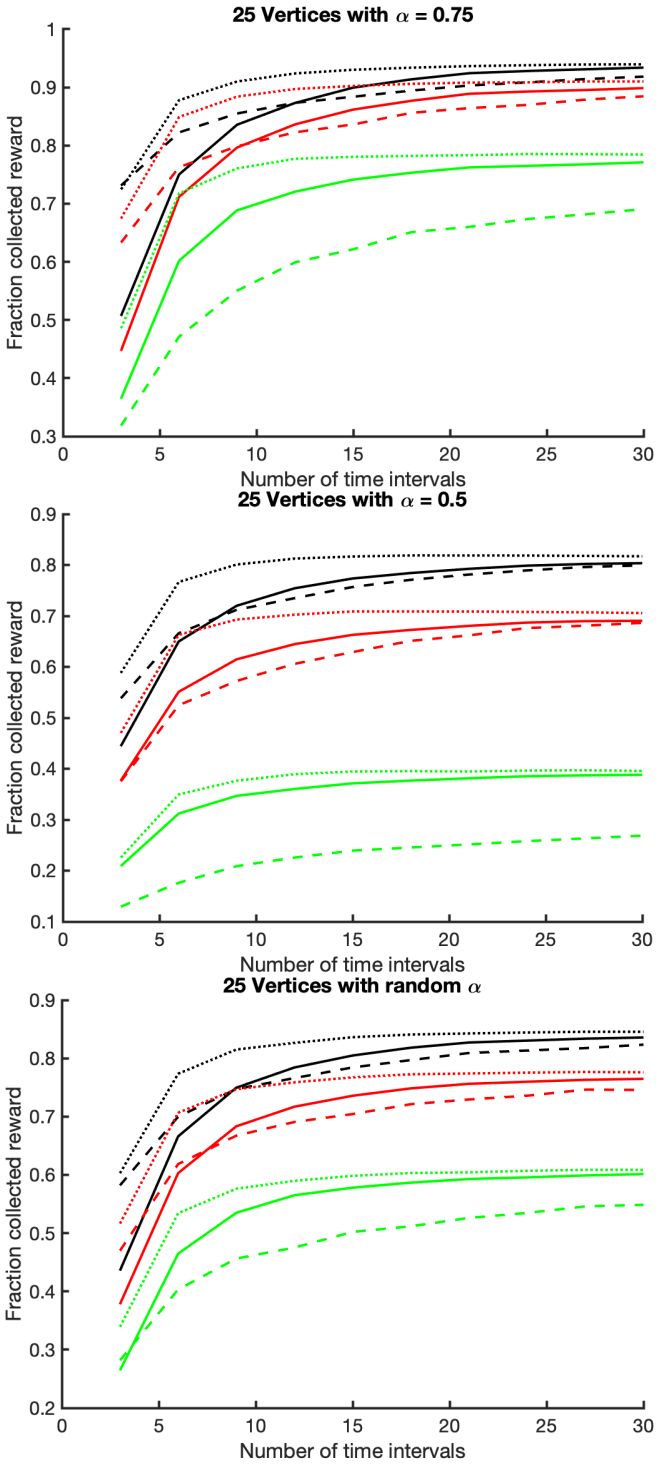

Fig. 4: *Legend*: Green indicates $P_f = 0.01$, red indicates $P_f = 0.05$, black indicates $P_f = 0.1$, solid lines indicate fixed-resolution, dashed lines indicate adaptive-resolution, and dotted lines indicates fixed-adaptive resolution. *Top*: Average rewards when $|\mathbb{T}|$ is varied and $\alpha = 0.75$. *Middle*: Average rewards when $|\mathbb{T}|$ is varied and $\alpha = 0.5$. *Bottom*: Average rewards when $|\mathbb{T}|$ is varied and α is a uniform random variable on the interval $(0, 1)$.

looking at the combined fixed-adaptive resolution method, the results change. The combined method is much better at maximizing reward collection, and it beats the other two methods in every case. Additionally, the reward curve flattens out early on, suggesting that the combined method reaches the optimum reward collection quicker than the others and less time intervals can be used for the same effectiveness, increasing computation efficiency. When α is random for each edge and $P_f = 0.1$, the fixed-adaptive resolution method with $|\mathbb{T}| = 15$ collects 83.62% of the total reward in $\mathcal{P}$, while the fixed-resolution method needs $|\mathbb{T}| = 30$ to collect 83.57% of the reward. If using $P_f = 0.05$ or $P_f = 0.01$, a nearly identical performance gap is achieved, again with the fixed-adaptive resolution method needing half as many time intervals as the fixed-resolution method. The results are similar when $\alpha = 0.5$ or $\alpha = 0.75$.

Analyzing how the results change when varying the number of vertices in the original path $|\mathcal{P}|$, seen in Figure 5, reveals more patterns. For these tests, $|\mathbb{T}|$ was fixed to 15 and the budget was adjusted for the desired $|\mathcal{P}|$. The adaptive-resolution method performs worse than the fixed-resolution method with smaller $|\mathcal{P}|$, however it gets better as this number increases. This suggests that adaptiveness is more useful as $\mathcal{P}$ gets longer, since arrivals at vertices further in the path use less of the entire time range from 0 to B and most of the fixed-resolution intervals become useless. Again, the combined fixed-adaptive resolution method performs the best in all cases. We also see a diverging pattern where the gap between this method and the others increases as $|\mathcal{P}|$ increases, again suggesting it is useful to use the ratio of $|\mathcal{P}|$ to $|\mathbb{T}|$ as a performance metric. This makes sense since we expect the resolution of each time interval to become more critical as the length of the initial path increases, due to the variation in arrival times at different vertices.

Lastly, we can analyze how $|\mathbb{T}|$ changes the computation time needed for each method, seen in Figure 6. The computation times shown are averages for every trial, including time to simulate the initial route (adaptive and combined), build state transition tables (all methods), and solve CMDPs (all methods). These trials were run on an Intel 6700k processor with 32GB ram and implemented using Matlab with CPLEX to solve each CMDP linear program. Between the three methods, the fixed-resolution approach is slightly faster when keeping the time steps and number of vertices the same, as it does not require simulation of the original path or calculating how to split the time intervals (seen as the gap in computation time). However, this does not account for the increased reward when using the combined fixed-adaptive resolution approach. Indeed, when comparing the fixed-resolution method at $|\mathbb{T}| = 30$ to the fixed-adaptive method at $|\mathbb{T}| = 15$, the combined method performance is equal to the fixed method in reward collection, yet takes less than half computation time to obtain solutions (average times of $0.9165s$ and $2.211s$, respectively). Overall, the presented results clearly show the proposed algorithm combining fixed and adaptive resolution time intervals offers a compelling tradeoff between performance and computational effort.

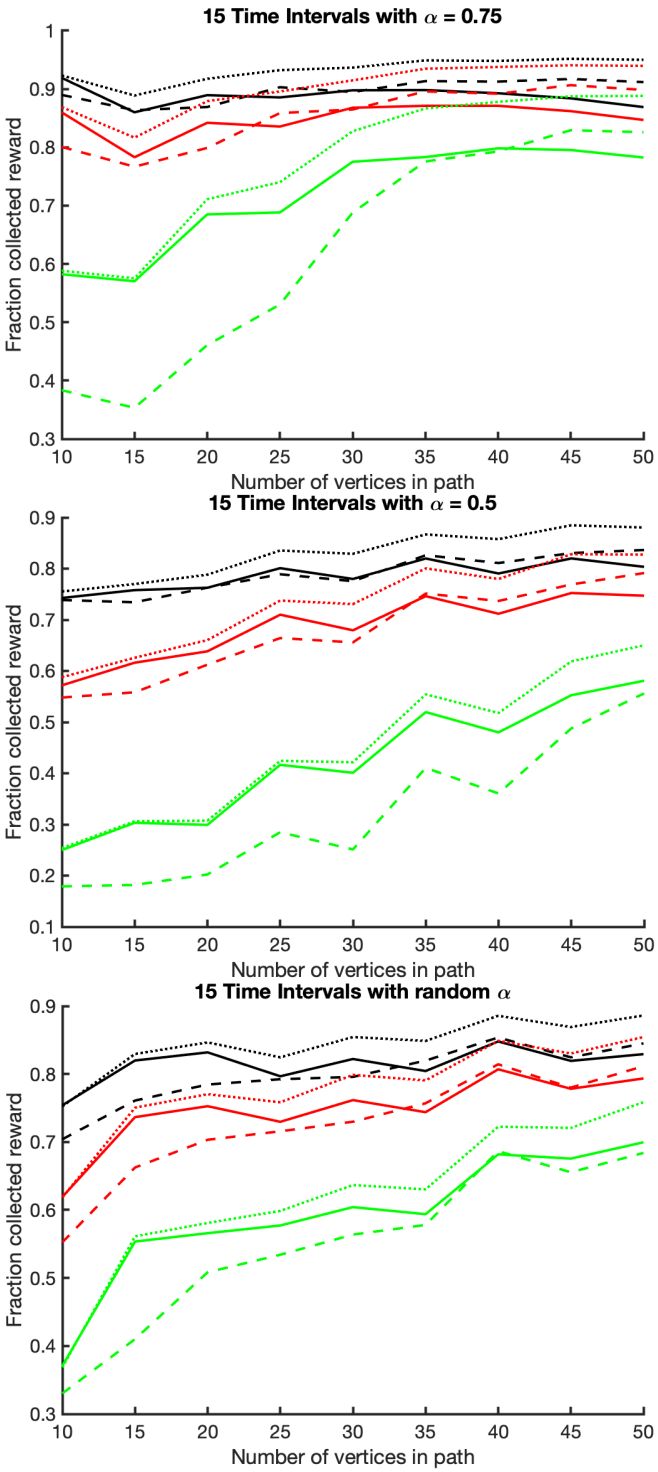

Fig. 5: *Legend*: Green indicates $P_f = 0.01$, red indicates $P_f = 0.05$, black indicates $P_f = 0.1$, solid lines indicate fixed-resolution, dashed lines indicate adaptive-resolution, and dotted lines indicate fixed-adaptive resolution. *Top*: Average rewards when $|\mathcal{P}|$ is varied and $\alpha = 0.75$. *Middle*: Average rewards when $|\mathcal{P}|$ is varied and $\alpha = 0.5$. *Bottom*: Average rewards when $|\mathcal{P}|$ is varied and α is a uniform random variable on the interval $(0, 1)$.

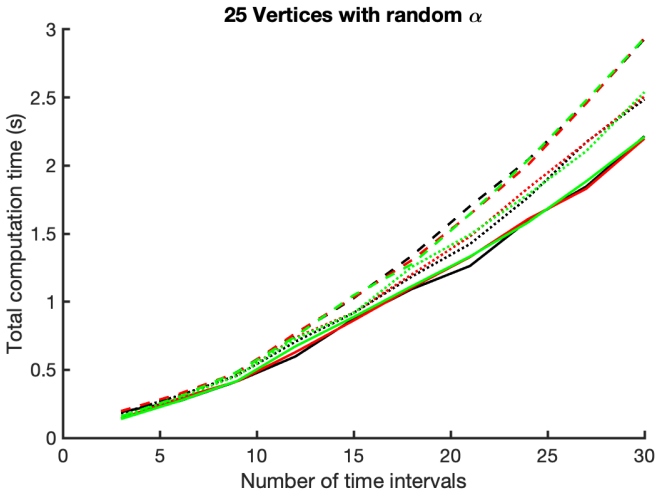

Fig. 6: Average computation time when $|\mathbb{T}|$ is varied and α is a uniform random variable in $(0, 1)$. *Legend*: Green indicates $P_f = 0.01$, red indicates $P_f = 0.05$, black indicates $P_f = 0.1$, solid lines indicate fixed-resolution, dashed lines indicate adaptive-resolution, and dotted lines indicate fixed-adaptive resolution.

VII. Conclusions

In this paper we studied the stochastic orienteering problem where travel times between vertices are continuous random variables with known pdfs. The objective is to compute a policy that collects the maximum expected reward while ensuring the probability of missing the deadline is bounded. Our proposed solution builds upon our recent work with CMDP based planners and exploits a carefully designed MDP structure to associate failure probabilities with occupation measures of ad-hoc added states. Starting from the fixed-resolution CMDP implementation, we derived a fixed-adaptive resolution extension that achieves the same results in a fraction of the time, enabling us to solve much larger problems. There are many venues for further research on the proposed approach, and we mention two. First, it would be interesting to better understand the theoretical properties of the proposed algorithm and how its performance varies with the parameters governing its behavior. Second, it would be interesting to derive an online version of the algorithm whereby the adaptive temporal segmentation is not computed offline, but rather at run time, thus capturing the characteristics of the run being executed. These will be the subject of future work.

References

[1] Eitan Altman. *Constrained Markov Decision Processes - Stochastic Modeling, Vol. 7*. Chapman and Hall/CRC, 1999.

[2] Nikhil Bansal, Avrim Blum, Shuchi Chawla, and Adam Meyerson. Approximation algorithms for deadline-tsp and vehicle routing with time-windows. In *Proceedings of the Thirty-Sixth Anuual ACM Symposium on Theory of Computing*, pages 166–174, 2004.

[3] Nikhil Bansal and Viswanath Nagarajan. On the adaptivity gap of stochastic orienteering. *Mathematical Programming*, 154:145–172, 2015.

[4] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. 1 and 2*. Athena Scientific, 1995.

[5] Ann M. Campbell, Michel Gendreau, and Barrett W. Thomas. The orienteering problem with stochastic travel and service times. *Annals of Operations Research*, 186(1):61–81, 2011.

[6] Stefano Carpin, Marco Pavone, and Brian M. Sadler. Rapid multirobot deployment with time constraints. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1147–1154, 2014.

[7] Chandra Chekuri, Nitish Korula, and Martin Pál. Improved algorithms for orienteering and related problems. *ACM Transactions on Algorithms*, 8(3):23:1–23:27, 2012.

[8] Ke Chen and Sariel Har-Peled. The orienteering problem in the plane revisited. In *Proceedings of the twenty-second annual symposium on Computational geometry*, pages 1–12, 2006.

[9] Yin-Lam Chow, Marco Pavone, Brian M. Sadler, and Stefano Carpin. Trading safety versus performance: Rapid deployment of robotic swarms with robust performance constraints. *Journal of Dynamic Systems, Measurement, and Control*, 137:031005.1–031005.11, 2015.

[10] Irina Dolinskaya, Zhenyu Shi, and Karen Smilowitz. Adaptive orienteering problem with stochastic travel times. *Transportation Research Part E*, 109:1–19, 2018.

[11] Lanah Evers, Kristiaan Glorie, Suzanne van der Ster, Ana Isabel Barros, and Herman Monsuur. A two-stage approach to the orienteering problem with stochastic weights. *Computers and Operations Research*, 43:248–260, 2014.

[12] Bruce L. Golden, Larry Levy, and Rakesh Vohra. The orienteering problem. *Naval Research Logistics*, 34:307–318, 1987.

[13] Aldy Gunawan, Hoong Chuin Lau, and Pieter Vansteenwegen. Orienteering problem: A survey of recent variants, solution approaches, and applications. *European Journal of Operational Research*, 255(2):315–332, 2016.

[14] Anupam Gupta, Ravishankar Krishnaswamy, Viswanath Nagarajan, and R. Ravi. Running errands in time: Approximation algorithms for stochastic orienteering. *Mathematics of Operations Research*, 40(1):56–79, 2014.

[15] Stefan Jorgensen, Robert H. Chen, Mark B. Milam, and Marco Pavone. The team surviving orienteers problem: Routing robots in uncertain environments with survival constraints. In *International Conference on Robotic Computing*, pages 227–234, 2017.

[16] Xinyue Kan, Thomas C. Thayer, Stefano Carpin, and Konstantinos Karydis. Task planning on stochastic aisle graphs for precision agriculture. *IEEE Robotics and Automation Letters*, 6(2):3287–3294, 2021.

[17] Jose Luis Susa Rincon, Pratap Tokekar, Vijay Kumar, and Stefano Carpin. Rapid deployment of mobile robots under temporal, performance, perception, and resource constraints. *IEEE Robotics and Automation Letters*, 2(4):2016–2023, 2017.

[18] Franscesco Betti Sorbelli, Stefano Carpin, Federico Corò, Alfredo Navarra, and Cristina Pinotti. Optimal routing schedules for robots operating in aisle-structures. In *IEEE International Conference on Robotics and Automation*, pages 4927–4933, 2020.

[19] Thomas C. Thayer and Stefano Carpin. Solving large scale stochastic orienteering problems with aggregation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2452–2458, 2020.

[20] Thomas C. Thayer and Stefano Carpin. An adaptive method for the stochastic orienteering problem. *IEEE Robotics and Automation Letters*, 6(2):4185–4192, 2021.

[21] Thomas C. Thayer, Stavros Vougioukas, Ken Goldberg, and Stefano Carpin. Routing algorithms for robot assisted precision irrigation. In *IEEE International Conference on Robotics and Automation*, pages 2221–2228, 2018.

[22] Thomas C. Thayer, Stavros Vougioukas, Ken Goldberg, and Stefano Carpin. Bi-objective routing for robotic irrigation and sampling in vineyards. In *IEEE International Conference on Automation Science and Engineering*, pages 1481–1488, 2019.

[23] Thomas C. Thayer, Stavros Vougioukas, Ken Goldberg, and Stefano Carpin. Multirobot routing algorithms for robots operating in vineyards. *IEEE Transactions on Automation Science and Engineering*, 17(3):1184–1194, 2020.

[24] Theodore Tsiligirides. Heuristic methods applied to orienteering. *Journal of Operational Research Society*, 35(9):797–809, 1984.

[25] Pradeep Varakantham and Akshat Kumar. Optimization approaches for solving chance constrained stochastic orienteering problems. In *International Conference on Algorithmic Decision Theory*, pages 387–398, 2013.

[26] Pradeep Varakantham, Akshat Kumar, Hoong Chuin Lau, and William Yeoh. Risk-sensitive stochastic orienteering problems for trip optimization in urban environments. *Transactions on Intelligent Systems and Technology*, 9(3):24:1–24:25, 2018.