An Adaptive Method for the Stochastic Orienteering Problem

Thomas C. Thayer Stefano Carpin

***Abstract*— We consider the NP-hard Stochastic Orienteering Problem, where the goal is to navigate between start and end vertices in a graph, maximizing the sum of rewards for visited vertices while obeying a travel budget over edges with stochastic cost within a given probability of failure. Previously, we solved this by finding an initial path using a deterministic orienteering solver and transformed it into a path policy using a Constrained Markov Decision Process that can skip vertices based on arrival time. In this work we augment our technique, creating a path tree which branches at vertex-time states with high probability of being skipped, allowing for new sequences of vertices in the resulting policy. We demonstrate that this adaptive path method collects significantly more reward in expectation, even when the number of branches is limited to control computation time.**

I. Introduction

The Orienteering Problem (OP) is a classic optimization problem defined over graphs with rewards associated to each vertex and costs associated to each edge. A path solving the OP maximizes rewards for all unique vertices visited while subject to a travel budget which the sum of costs for all edge traversals must be equal to or less than. Rewards for vertices visited more than once are counted only once, while costs for edges crossed multiple times are incurred every time. A path is a set of vertices which are ordered as a specific sequence, thus taking an agent from a start vertex to a goal vertex with some subset of vertices and edges from the graph in between. This problem is motivated by numerous real world examples, such as tourist sight seeing, vehicle routing, or robotic navigation, wherein an agent must plan for places of interest to visit in order to maximize the utility of their budget, which may be some limit on travel time, distance, energy consumption, etc. However, the OP is limited in its ability to properly describe these problems, as it assumes all edges have deterministic costs. In real world problems, costs are often stochastic, meaning that the goal cannot always be reached before exhausting the travel budget. It is easy to imagine situations where a simple path between vertices is not adequate, and therefore an agent following the path should have contingency plans if the situation arises where costs accumulate quicker than expected. Naturally, the ability to take shortcuts and skip one or more stops in the path is one type of contingency plan. Another type of plan is to take a different route toward the goal, maximizing the utility of the remaining budget with a new path going places yet to be visited. Both of these ideas require the use of a path policy, which determines where to go next based on not just the current vertex but also the current time. In this paper, we present a solution to this problem, called the Stochastic Orienteering Problem (SOP), where we provide a policy that utilizes both types of contingencies to meet a bound on the probability of going over budget before reaching the goal.

Our solution involves multiple steps: First, an initial path is found with a deterministic orienteering solver using the expected costs for all edges. Then, a path policy is devised using a Constrained Markov Decision Process (CMDP) architecture to determine when to take shortcuts along the initial path. Next, new paths are computed for accumulated costs and locations with high probabilities of utilizing shortcuts, such that an agent may take these instead of the shortcuts. Finally, a new policy is computed with a CMDP, combining all paths and shortcuts into a single actionable plan that optimizes overall reward and guarantees a limit on the probability of not reaching the goal vertex within the budget.

The rest of this paper is organized as follows. Section II discusses related works, and Section III formally introduces key concepts and the problem at hand. In Section IV we discuss a fixed path approach based on constrained Markov decision processes, and in Section V we introduce an adaptive path extension. Results of randomized simulations for these methods are given in Section VI, and we recap our work and propose future work.

II. Related Work

The OP first appeared in [10] where it was proven to be NP-hard, and was later shown to be APX-hard in [2]. Numerous variants and solution methods have been devised for the OP, and the reader is referred to [11] and [23] for supplementary information. On smaller problem sizes (up to 500 vertices), the OP can be solved exactly, commonly using integer programming methods, however for larger problems these methods have intractable growth in computation time. Approximation algorithms are useful when problems contain more vertices, and after extensive literature review the best we could find was proposed by [8], which gives a $(2+\varepsilon)$ approximation with a time complexity of $n^{\mathcal{O}(\frac{1}{\varepsilon})}$ where n is the number of vertices. In practice, heuristic methods informed by domain-specific knowledge offer the best balance between computation time and reward collection (without guarantees), and we highlight our recent works [17], [19], [20] which can solve very large OP instances ($\geq 50,000$ vertices)

This work is supported by grant no. 2017-67021-25925 from the USDA National Institute of Food and Agriculture. Thomas Thayer was also partially supported by the National Science Foundation under grant DGE-1633722. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the view of the U.S. Department of Agriculture or the National Science Foundation.

T. Thayer and S. Carpin are with the Department of Computer Science and Engineering, University of California, Merced, CA, USA {tthayer,scarpin}@ucmerced.edu.

extremely quickly by exploiting the peculiar structure of graphs encountered in our particular application.

The SOP is less well studied than the OP. Introduced in [5], the authors consider uncertainty in edge traversal and vertex service costs, but other versions of the problem consider non-deterministic rewards as well. In [5] an exact solutions for problems restricted to very exclusive types of graphs is provided, and also generalized heuristics that are less restrictive. The method presented in [12] studies the case of only stochastic service costs, giving a non-adaptive constant factor approximation algorithm and an adaptive (policy driven) $\mathcal{O}(\log \log B)$ approximation, where B is the cumulative cost budget for edge traversal and vertex service. The related literature [3] proves a lower bound for the worst-case ratio between the optimal rewards for adaptive and non-adaptive plans (called the adaptivity gap) of $\Omega(\sqrt{\log \log B})$. These works only seek to solve the SOP maximizing expected reward without considering the risk of failure to meet budget constraints, which is what we consider in this work.

Chance constraints are used frequently when solving problems involving stochasticity, as they are able to restrict risky behavior for robust solutions. A few works have studied chance constraints within the orienteering domain, such as [14], which studied a version of the team OP with survival probabilities on edges, seeking to maximize reward while ensuring that at least one agent survives with a minimum probability. For the SOP, [24] considers risk-sensitivity on a formulation with stochastic weights and chance constraints, with an open-loop solution based on a mixed integer linear program formulation. However, this does not provide a policy allowing an agent to dynamically adjust its path based on current budget expenditure. This is different from the solution we provide in our previous work [18] and here, which instead gives a policy considering incurred travel costs after each edge traversal. To do this, we utilize the CMDP framework, which (along with the closely related Constrained Partially Observable MDP) is used extensively for chance constrained problems (see [9], [13], [15], [16], [21] for examples).

III. Definitions and Problem Formulation

We begin by defining the SOP and some useful fundamental concepts, which were introduced in [18]. From here onward, "cost" and "time" are used interchangeably, where time is a particular type of cost referring to edge traversal.

A. The Deterministic Orienteering Problem

The deterministic OP is defined on an undirected fully-connected graph $G(V, E)$, with an edge cost function $c : E \to \mathbb{R}^+$ defining the time required to traverse an edge and vertex reward function $r : V \to \mathbb{R}^+$. If G is not fully-connected, it can be made as such by inserting new edges with cost equivalent to the minimum cost path between two vertices. Given start and goal vertices $v_s, v_g \in V$ and a budget B, find a path $\mathcal{P}$ from v_s to v_g that maximizes the sum of collected rewards $R(\mathcal{P})$ on unique visited vertices, without exceeding B for the total cost $C(\mathcal{P})$ accrued every time an edge is traversed. In the case where v_s and v_g are coincident, a tour is determined. In general, the start and goal vertices do not need to be specified for the OP, however practical considerations often require that they are. Some versions of the problem require only v_s to be specified, and this is often referred to as the "rooted" OP. In any case, the OP is NP-hard regardless of whether one or both vertices are specified.

B. Path Policy

A path $\mathcal{P}$ on G is a sequence of n vertices $v_1, \dots v_n \in V$ such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq n-1$. For $v_i \in \mathcal{P}$, we define the set of vertices following v_i in $\mathcal{P}$ as $\mathcal{S}(v_i) = \{v_{i+1}, v_{i+2}, \dots, v_n\}$. Following this definition, $\mathcal{S}(v_n) = \varnothing$. Given a path $\mathcal{P}$, a path policy π over $\mathcal{P}$ is a function $\pi : \mathcal{P} \times \mathbb{R}^+ \to \mathcal{P}$ that maps each vertex-time pairing (v_j, t) to an action (the next vertex to travel to) such that $\pi(v_j, t) \in \mathcal{S}(v_j)$. This formalizes the idea of taking a *shortcut* over $\mathcal{P}$ (see Figure 1), which is useful to meet a budget constraint in the context of randomized travel times, as we will see later. Assuming $v_1 = v_s$ and $t_0 = 0$, π establishes a way for the agent to plan ahead and skip vertices based on current time and position to reach $v_n = v_g$ before the deadline B expires.

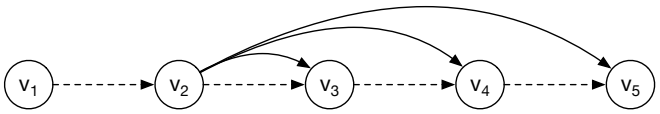

Fig. 1: For a hypothetical path of 5 vertices, π allows skipping one or more vertices along the path. In particular, from v_2, depending on the value of the temporal parameter t, the path policy defines which successive vertex an agent should move to next. By skipping one or more vertices, the agent can decrease the expected cumulative cost of reaching the last vertex in $\mathcal{P}$, although doing so will decrease the overall collected reward.

C. The Stochastic Orienteering Problem

For every edge $(v_i, v_j) \in E$, let $f_{(v_i,v_j)}$ be a probability density function (pdf) with positive support and finite expectation. The incurred cost for each traversal of edge (v_i, v_j) is a random variable $c_{i,j}$ whose pdf is $f_{(v_i,v_j)}$. Starting at time $t_0 = 0$ and vertex v_1, an agent moves along path $\mathcal{P}$ following path policy π. For $i \geq 1$, at vertex v_i and time t_i the agent moves to $v_j = \pi(v_i, t_i)$ arriving at v_j at time $t_j = c_{i,j} + t_i$, where $c_{i,j}$ is a random variable with pdf $f_{(v_i,v_j)}$. The agent continues moving along the path until it arrives at the last vertex in the path v_n. Because the movement times are random variables, the total cost and total reward collected along path $\mathcal{P}$ following path policy π are also random variables as indicated by $C_{\mathcal{P},\pi}$ and $R_{\mathcal{P},\pi}$, respectively. As only the vertices in $\mathcal{P}$ are visitable according to π, $\mathbb{E}[R_{\mathcal{P},\pi}] \leq R(\mathcal{P})$ is always true. Then, the SOP asks to find a path $\mathcal{P}$ and path policy π that maximizes the expected sum of rewards $\mathbb{E}[R_{\mathcal{P},\pi}]$ such that $\Pr[C_{\mathcal{P},\pi} > B] \leq P_f$ for a given P_f. This chance constraint aims at limiting the probability that the last vertex is reached after having exceeded the allocated budget B, and therefore P_f can be

interpreted as a failure probability. The SOP is a generalized version of the OP and is consequently NP-hard as well.

IV. Solving the SOP using a CMDP

A. Defining an MDP

Let us assume $\mathcal{P} = \{v_1, \ldots, v_n\}$ in G is given. The objective is to now determine a path policy such that for every vertex in $\mathcal{P}$ and time, $v_j = \pi(v_i, t_i)$ is defined where $v_j \in \mathcal{S}(v_i)$. The transition time to go from v_i to v_j is stochastic and characterized by the pdf associated with the edge between them. A natural way to formalize this is to use a Markov Decision Process (MDP) for a suitable augmented state space. We assume the reader is familiar with MDPs and refer to [4] for a comprehensive introduction. Given a set of states S, set of actions A, transition kernel $\Pr$, and reward function $r(s \in S, a \in A)$, an MDP is defined as $\mathcal{M} = \{S, A, \Pr, r\}$, with the following properties:

- $S = V \times \mathbb{T}$, where V is the set of vertices in $\mathcal{P}$ and $\mathbb{T}$ is a suitable discretization of time, such that there are $N = \lceil \frac{B}{\Delta} \rceil$ sequential time steps of length Δ between t_0 and B, where t_k describes the interval $[k\Delta, (k+1)\Delta)$. The composite state (v_i, t_k) represents an agent arriving at vertex v_i during the time interval t_k.
- The action set for each state (v_i, t_k) is the set of vertices following v_i, $A_{v_i} = \mathcal{S}(v_i)$. The action set of any state is independent of the time interval for that state. The total action set is then $A = \bigcup_{i=1}^{n} A_{v_i}$.
- $\Pr$ is the transition kernel defining the probability of landing in successor state (v_j, t_l) when executing an action $a \in A_{v_i}$ in state (v_i, t_k), indicated by $\Pr((v_i, t_k), a, (v_j, t_l))$. The next vertex is deterministically chosen by the action, and therefore all states including vertices not equal to a have a transition probability of 0. For all successor states in which $l < k$, the transition probability is also 0 because the agent cannot travel back in time. For the remaining states, the transition probability is equal to $\int_{t_i\Delta}^{(t_i+1)\Delta} [F(\Delta(t_k+1) - \xi) - F(\Delta t_k - \xi)] d\xi$, where F is the cumulative function of the pdf associated with edge (v_i, v_j).
- r is the reward function for each state/action pair, where $r((v_i, t_k), a)$ is equal to $r(v_i)$, a.k.a. the reward for v_i in G. Note the abuse of notation, as the two are indeed equivalent regardless of t_k or a.

The cumulative reward function used to determine the policy is critical to the definition of an MDP. Since traversal of $\mathcal{P}$ occurs as a single episode, we opt for a non-discounted reward and add two special states, the *failure* state s_f and the *loop* state s_l. The failure state represents any combination of vertex and time where $t_k > B$, and thus describes the condition that an agent has failed to reach the goal vertex before the deadline B. The transition kernel is extended to include the probability that the budget is expended during any transitional action, $\Pr((v_i, t_k), a, s_f)$ where the intended vertex of that action v_j is not reached, and the reward associated with transitioning from any vertex into s_f and out of s_f is 0. The loop state completes the definition of the MDP and represents the conclusion of the traversal episode. All states associated with v_n or the failure state s_f have a single action a_l, leading to s_l with probability 1. s_l also has a single action a_l leading to itself with probability 1 and reward of 0. Under this definition, the loop state cannot be exited once entered, and no more reward is collected. The structure of the MDP is illustrated in Figure 2, showing the composite state space and some example transitions.

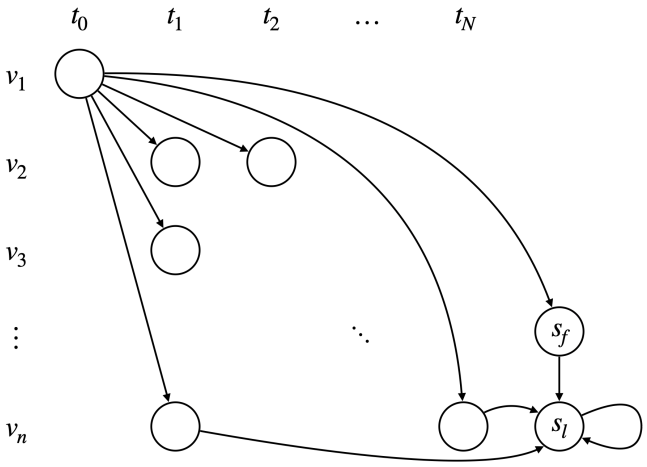

Fig. 2: States in the MDP are shown as a grid with rows representing vertices and columns representing time intervals. Arrows are depicted for some transitions with non-zero probability. From state (v_1, t_0) it is possible to go to any of the following vertices, and the next vertex arrival time can be any time interval $t_i > t_0$. Reaching a vertex after B has passed is modeled as a transition towards s_f. Note that once the state reaches the last vertex in the path (v_n), a deterministic transition is made to the loop state s_l.

This MDP structure is similar to the works [7], [9], [15], which utilize failure and loop states to design control policies with bounded failure probabilities. In this case, the failure probability is equivalent to the probability of hitting s_f when following a policy π, meaning that the last vertex along $\mathcal{P}$ could not be reached before B. The probability of reaching s_l is 1 for all valid policies, and the start state is constrained to (v_1, t_0). Therefore, the non-discounted reward function is

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

where the expectation is taken according to the probability distribution induced by π and X_i, a random variable for the state at time t. The expectation exists and is finite, since reaching s_l has probability 1 and no loops precede s_l.

B. Defining a CMDP

The goal of the SOP introduced earlier is to maximize expected rewards while bounding the probability of entering s_f. With this definition, an MDP is inadequate because it is capable of handling only a single objective function. Therefore, we need to extend our MDP definition to a Constrained MDP (CMDP), a type of model which can maximize for one objective function while ensuring bounds in expectation for others [1]. To do this, we introduce a

secondary cost for each state/action pair $d : S \times A \rightarrow \mathcal{R}^+$ that is 0 everywhere except for (s_f, a_l) where it is 1. This sets up a cost that is incurred only when an agent passes through the failure state. Because an episode can pass through the failure state only once, the probability of failure for any policy is equal to the expectation of this secondary cost $\mathbb{E}[D(\pi)]$.

A CMDP is typically solved using a linear program (see [1], ch. 8), with the optimization variable ρ that represents the occupation measure for each state/action pair $\rho(x, a) = \sum_{t=1}^{\infty} \Pr[X_t = x, A_t = a]$, and β that represents the start state distribution (which we set as 1 for (v_1, t_0) and 0 everywhere else).

$$\max_{\rho} \sum_{(x,a) \in S \times A} \rho(x, a) r(x, a) \tag{1}$$

$$\text{s.t.} \sum_{(x,a) \in S \times A} \rho(x, a) d(x, a) \leq P_f \tag{2}$$

$$\sum_{y \in S} \sum_{a \in \mathcal{S}(y)} \rho(y, a)(\delta_x(y) - \Pr(y, a, x)) = \beta(x)$$

$$\forall x \in S \setminus \{l\} \tag{3}$$

$$\rho(x, a) \geq 0 \quad \forall (x, a) \in S \times A \tag{4}$$

This formulation has an objective function defined by Eq. (1) maximizing the reward over the set of occupancy measures ρ. Constraint (2) specifies that the occupancy cost of the failure state is less than or equal to P_f. Since the failure state can be visited just once, this bounds the failure probability. Constraint (4) enforces that all state/action pairs have a non-negative occupancy measure. Constraint (3) is a flow preservation constraint defining the set of valid occupancy measures. It is related to both the initial distribution β and the transition probabilities between states (see [1], ch. 8 for details.) Here, $\delta_x(y)$ is a function with value 1 when $x = y$ and 0 in all other circumstances.

The linear program admits a solution if and only if a stationary, randomized policy π can be found that satisfies the cost constraint and is uniquely defined by ρ as follows:

$$\pi(x, a) = \frac{\rho(x, a)}{\sum_{a \in \mathcal{S}(x)} \rho(x, a)} \qquad \forall (x, a) \in S \times A \tag{5}$$

where $\pi(x, a)$ is the probability of taking action a in state x. If E.q. 5 has a 0 in the denominator for any state, then that state is unvisited and the policy for (x, a) can be defined arbitrarily. For a detailed discussion about this approach, the reader is referred to [6], [7], [9], [15]. The following theorem, which we presented and proved in [18], shows that the above linear program defines a policy satisfying the failure probability constraint P_f.

Theorem 1. *If the linear program admits a solution, then the associated policy π fails with a probability of at most P_f to reach the vertex v_n within budget B.*

The formulation based on a CMDP leads to the following algorithm to solve an instance of the SOP:

1) Create an instance of the deterministic OP assigning to every edge e the expected travel cost $\mathbb{E}[c]$.
2) Solve the deterministic OP with v_s, v_g, and B using any deterministic 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 of this proposed algorithm 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 [11]. For large problems or repeated runs of the algorithm, where using mixed integer programs becomes computationally impractical, a heuristic or approximated method may be used.

V. An Adaptive Path Method

In [18] we used the method described in Section IV to compute solutions for the SOP and showed that a deterministic path can indeed be used as the starting point for a path policy capable of reaching a goal vertex within a given failure bound while also maximizing rewards. However, this algorithm outputs a policy π that can only visit vertices in $\mathcal{P}$, and therefore limits the maximum potential reward to $R(\mathcal{P})$, with little room for improvement in expected reward. Here we present a new method which does not rely solely on the initial deterministic path, and gives the policy alternatives where reward collection increases in expectation.

This new method builds upon the previous one, but improves it by allowing the policy to deviate from the starting path. This is done by computing new paths originating from a subset of the intermediate vertices along the path. Situations where it is beneficial to allow the policy to deviate from the initial deterministic path include arriving at a vertex v_i much earlier or later than expected. In these cases it means that the current realization is deviating from the expected average behavior and it may therefore be advantageous to compute a new route based on the current position and residual budget. There may be some room in the budget to visit a different subset of V, following a new path $\mathcal{P}^{new}$ and policy π_{new} computed from v_i, such that the expected reward is higher than continuing with the initial path and policy. The new path $\mathcal{P}^{new}$ and path policy π_{new} can be computed online, however this computation is expensive and difficult to do on-the-fly, limiting the potential adaptivity gains.

Instead, $\mathcal{P}^{new}$ can be computed offline and retrieved on demand. Doing this, π_{new} will consider a directed path tree $\mathcal{PT}$, where branching indicates possible paths to traverse, including the initial deterministic path and potential deviations. The new policy starts at state $(v_1, t = 0)$ and moves along $\mathcal{P}$ according to π up to (v_i, t_j), where it can continue along $\mathcal{P}$ or branch to $\mathcal{P}^{new}$ depending on the arrival time. If $\mathcal{P}^{new}$ is utilized by the policy, then only shortcuts to future vertices along $\mathcal{P}^{new}$ may be taken from (v_i, t_j) onward. Setting the rewards for already visited vertices to 0, $\mathcal{P}^{new}$ can be computed using a deterministic orienteering solver resulting in a single path from v_i to v_g with budget $B - t_j$ that optimizes for rewards yet to be collected. Multiple new paths are allowed from v_i, but there can be only one for each possible arrival time. Since there may be many

states where a new path is beneficial, multiple branches can be added to $\mathcal{PT}$. Because we define a directed path tree, branches are not allowed to merge. Paths are ordered sets of vertices, so any vertex along a branch implicitly encodes information about previously visited vertices and this information is used to build new branches when the original bifurcates. The merging of two branches would provide inconsistent information about which vertices have been previously visited. Thus, recombining branches would exclude vertices than an agent may not have visited since branches purposely visit different subsets of V. An example of $\mathcal{PT}$ is given in Figure 3.

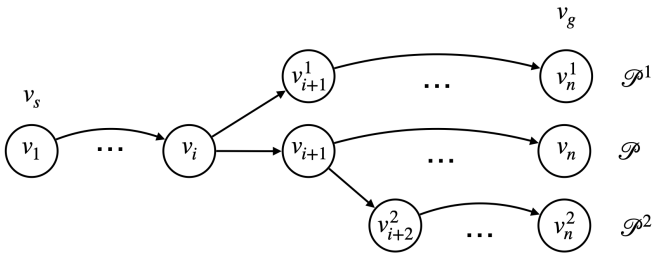

Fig. 3: An example of a path with two additional branches at v_i and v_{i+1} to create a path tree $\mathcal{PT}$. In this example, from v_1 it is possible to take a shortcut to any vertex in $\mathcal{P}$, $\mathcal{P}^1_{j>i}$, and $\mathcal{P}^2_{j>i+1}$. But from v_{i+1} it is only possible to take shortcuts to vertices further along in $\mathcal{P}$ and any vertex in $\mathcal{P}^2_{j>i+1}$, and v^1_{i+1} can only take shortcuts to vertices in $\mathcal{P}^1$.

Superscript notation is used to indicate paths following different branches of $\mathcal{PT}$ and subscript notation indicated a vertex in the path. All feasible paths start at v_1 (equivalently $\mathcal{P}^k_1$ or v_s) and end at v_n ($\mathcal{P}^k_n$ or v_g), where n indicates the total number of vertices in that path. A policy over $\mathcal{PT}$ may consider shortcuts from some v_i to vertices further along the path and any future connecting branches, but not to any branches that connected before v_i.

The idea of using a path tree $\mathcal{PT}$ instead of a singular deterministic path $\mathcal{P}$ leads to a new adaptive path algorithm for solving an instance of the SOP:

1) Create an instance of the deterministic OP assigning to every edge e the expected travel cost $\mathbb{E}[c]$.
2) Solve the deterministic OP with v_s, v_g, and B using any deterministic method and let $\mathcal{PT}$ be the returned path.
3) Use $\mathcal{PT}$ to build and solve the CMDP described in Section IV to obtain π.
4) Find the set of states $(v_i, t_i) \in S_{jump}$ where π induces an action resulting in a shortcut to a future vertex v_j where $j > i+1$.
5) For each state in S_{jump}, use any deterministic orienteering method to find a new path branch from v_i to v_g with budget $B - t_i$ and add it to the path tree $\mathcal{PT}$.
6) Use $\mathcal{PT}$ to build and solve the CMDP as described above to obtain π_{new}.

This adaptive path algorithm uses the method described in IV to create an initial policy π (steps 1-3) to determine where branches should occur (step 4), then add new branches to $\mathcal{PT}$ (step 5) and create a new policy π_{new} (step 6). In step 5, it may be the case where a new branch at v_i starts with some sequence of the same vertices in $\mathcal{P}$, i.e. $v_{i+1} \dots v_j$, therefore the branch can instead be shortened and start at v_j. Duplicate branches can also occur, and these are safely discarded. The final output is a policy π_{new} that dictates actions to take when reaching vertices in $\mathcal{PT}$ at various times to maximize expected reward and bound the probability of failure to P_f. Because of the adaptive nature of the path tree, the expected reward $\mathbb{E}[R_{\mathcal{PT},\pi_{new}}]$ will be equal to or greater than that for the policy from the method described in IV, and can possibly be greater than $R(\mathcal{P})$. It should be noted that steps $4-6$ can be repeated iteratively to create more path branches originating from the previously added branches, however this greatly increases the amount of computation required to find π_{new} even with heuristics, and we do not utilize it.

A. Branch Heuristics

When solving a CMDP using linear programming, the computational complexity is dependent on the number of state/action pairs $|\mathcal{S} \times \mathcal{A}|$. For the deterministic path method described in section IV, it grows super-linearly with respect to the size of the state space, and computation time becomes intractable with large state spaces. This number is equal to

$$|\mathbb{T}| \cdot \left(\frac{n(n-1)}{2} + 1 \right) + 2 = \mathcal{O}(|\mathbb{T}| \cdot n^2)$$

where n is the number of vertices in $\mathcal{P}$ and $|\mathbb{T}|$ is the number of time intervals.

The adaptive path algorithm can compute policies with higher expected rewards than the deterministic path method, but at the cost of increased state space and computation time. New branches are added where shortcuts are taken in the initial policy π, however there may be many states where shortcuts occur. As a consequence, the number of vertices in $\mathcal{PT}$ can be many times greater than in $\mathcal{P}$, and the number of state/action pairs increases to

$$|\mathbb{T}| \cdot \sum_{\mathcal{P}^b \in \mathcal{PT}} \left(\frac{|\mathcal{P}^b|(|\mathcal{P}^b|-1)}{2} + 1 \right) + 2 = \mathcal{O}\left(|\mathbb{T}| \cdot |\mathcal{PT}|^2 \right)$$

where $|\mathcal{PT}|$ is the total number of vertices in the path tree, and $|\mathcal{P}^b|$ is the number of vertices in branch b.

Each new branch added to $\mathcal{PT}$ has a variable path length, which cannot be controlled explicitly since they are computed using a deterministic orienteering algorithm. The number of branches, however, can be controlled, and reducing this number using heuristics is a straight forward approach. An intuitive way of doing this is to only add branches that have a high likelihood of being utilized by the policy. Because non-loop states in the CMDP can be visited only once, ρ is equal to the probability that a state/action pair is executed by the policy. Therefore, the adaptive path algorithm can be modified such that only the top k_b shortcut actions with highest ρ values will add states to S_{jump}. The modified algorithm is given as follows:

1) Follow steps 1-4 of the adaptive path algorithm.
2) Sort S_{jump} according to ρ in descending order.

3) For the first k_b states in S_{jump}, use any deterministic orienteering method to find a new path branch from v_i to v_g with budget $B - t_i$ and add it to the path tree $\mathcal{PT}$.
4) Use $\mathcal{PT}$ to build and solve the CMDP as described earlier to obtain π_{new}.

This adaptive path heuristic limits the number of branches added to $\mathcal{PT}$ to k_b instead of potentially $n \times |\mathbb{T}|$. Note that in the case of $k_b = 0$, no branches are added and this method produces the same policy as the deterministic path method.

VI. Results

To compare our new adaptive path algorithm for solving the SOP to the deterministic path algorithm, we simulated the methods on randomized synthetic problems allowing us to determine the general effectiveness of our technique. Vertices $v \in V$ for G were obtained sampling the unit square with uniform distribution, and edges $e \in E$ were added between every vertex. The reward for each vertex $r(v)$ was a random sample from a uniform distribution in $[0, 1]$. Stochastic travel times along edge (v_i, v_j) were obtained from

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

where $d_{i,j}$ is the Euclidean distance between the two vertices, $\mathcal{E}(\lambda)$ is a random sample from an exponential distribution with parameter λ, and $0 < \alpha < 1$ is a parameter describing the relationship between the expected cost of the edge (equal to $d_{i,j}$) and the amount of variance $((1-\alpha)d_{i,j})^2$. Under this formulation, edges always have non-negative costs.

For both methods, we used the S-algorithm heuristic described in [22] as our deterministic orienteering solver due to its speed and robustness. An exact solver based on mixed-integer linear programming was not used because these often take longer to find a path than solving the CMDP, and would be impractical for the adaptive path algorithm which calls on the orienteering solver multiple times. The output of the deterministic orienteering solver is an initial path $\mathcal{P}$ with expected cost less than or equal to B. In each trial simulation of the algorithms, this path and the graph it navigates remained fixed for a set of parameters so that a fair comparison could be made between the two algorithms. The fraction of collected reward is the ratio of each algorithms expected reward to the total reward of $\mathcal{P}$, as $\frac{\mathbb{E}[R_{\mathcal{P},\pi}]}{R(\mathcal{P})}$. Because the adaptive path algorithm can deviate from $\mathcal{P}$, it is possible for this ratio to be greater than 1. The process was repeated for both methods on 10 different instances with unique G and $\mathcal{P}$. The results for each set of parameters were averaged across all 10 instances.

For the simulations, the parameters were set as follows. α was randomized uniformly such that all edges had a unique value. The probability of failure P_f was set to three different values, indicating the allowed proportion of failure to reach the goal vertex. The branching factor was varied as $k_b = \infty$ (all possible branches were added to $\mathcal{PT}$), $k_b = 5$ (only the top 5 branches were added to $\mathcal{PT}$), and $k_b = 0$ (no branches were added, equivalent to the deterministic path method from Section IV). As an example, Figure 4 shows one of the simulated problem graphs with the computed path tree for $k_b = 5$ overlayed onto it. Notice how different branches are able to visit different subsets of vertices. Some branches appear to overlap in the graph, however they remain separate in the path tree.

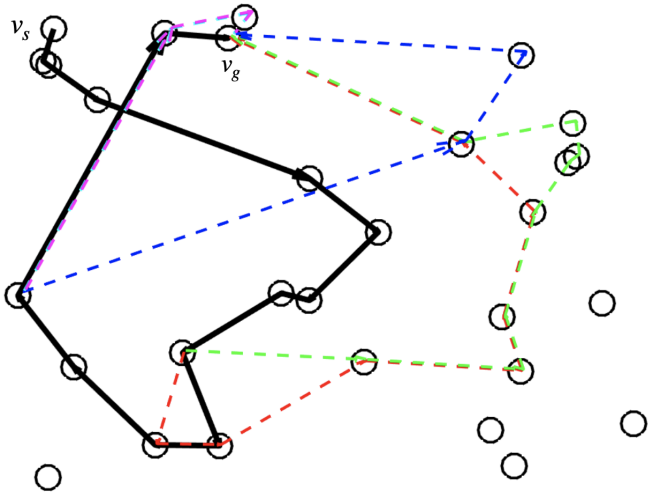

Fig. 4: The computed path tree for one of the evaluated problems. The initial path is shown in black, while the branched paths are shown in various colors.

For the first set of simulations, we fixed the number of vertices in $\mathcal{P}$ and varied the number of time steps, with results shown in Figure 5. For the second set of simulations, we fixed the number of time steps and varied the number of vertices in $\mathcal{P}$, with results shown in Figure 6. For both sets of simulations, the results are shown in terms of the average expected fraction of reward collected and the average total computation time for each set of parameters. All simulations were run on a computer with an Intel 6700k processor and 32GB of RAM, with our methods coded in Matlab and CPLEX used to solve the CMDP linear programs.

The results shown in Figures 5 and 6 are indicative of clear trends for the average expected reward and average computation time. For expected rewards, a $k_b = 0$ (representing the deterministic path method without branching) is significantly lower than $k_b = 5$ and $k_b = \infty$ across all numbers of time steps (Figure 5), numbers of vertices in $\mathcal{P}$ (Figure (6)), and values of P_f. For 15 vertices and 10 time steps, there is an average difference of 6.07% between $k_b = 0$ and $k_b = \infty$, and this number increases as the state space grows in size (up to 10.5% in these instances). Interestingly, $k_b = 5$ is very comparable to $k_b = \infty$, staying just below or right at the same average expected reward, showing that the branching heuristic is effective at adding only the most valuable branches to the path tree. For average computation time, the adaptive path method using $k_b = \infty$ and $k_b = 5$ is clearly slower than the deterministic path method, showing a trade off between solution quality and time to solution. Setting a lower k_b value does help however, as it limits the number of branches from growing super-linearly with the number of time steps or vertices in $\mathcal{P}$. At 15 vertices and 10 time steps, the average computation time for $k_b = 5$ is only

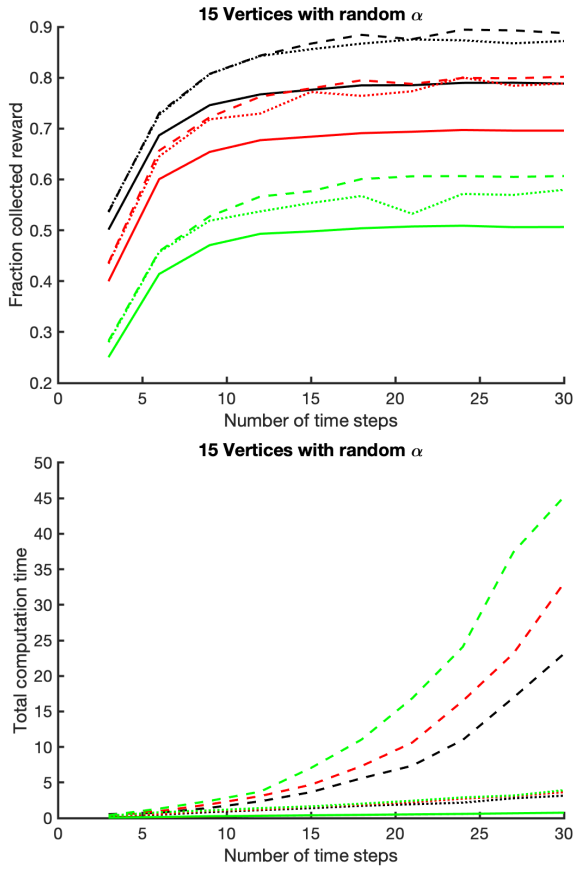

Fig. 5: Rewards and computation time when the number of vertices in the initial path is fixed at 15. *Legend*: Green indicates $P_f = 0.01$, red indicates $P_f = 0.05$, black indicates $P_f = 0.1$, solid line indicates $k_b = 0$, dashed line indicate $k_b = \infty$, and dotted line indicates $k_b = 5$.

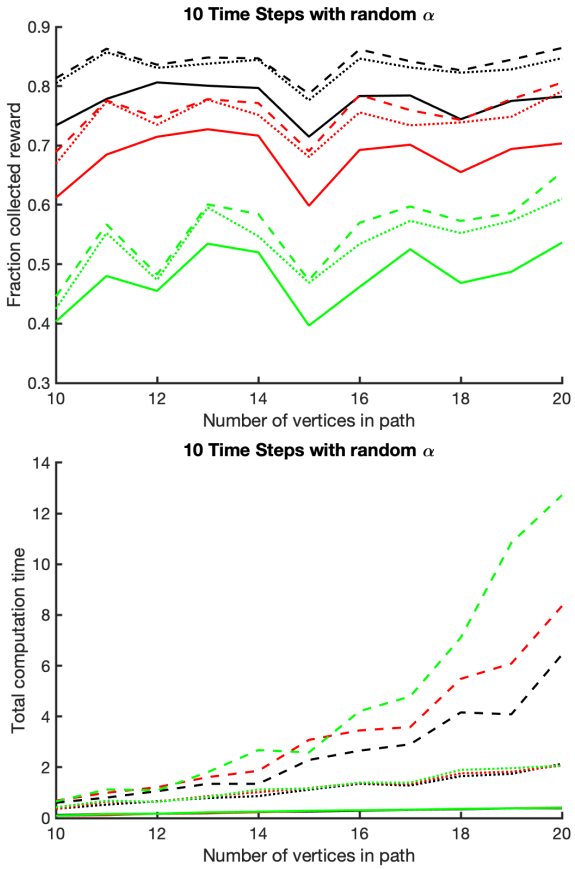

Fig. 6: Rewards and computation time when the number of time intervals is fixed at 10. *Legend*: Green indicates $P_f = 0.01$, red indicates $P_f = 0.05$, black indicates $P_f = 0.1$, solid line indicates $k_b = 0$, dashed line indicate $k_b = \infty$, and dotted line indicates $k_b = 5$.

56.5% that of $k_b = \infty$, and this number shrinks as the state space grows in size, showing that a small branching factor pays a small price for increasing the expected reward.

Another aspect deserving investigation is the scalability of the proposed method. Therefore, in our final simulations we also attempted to find the maximum size graph $|V|$ that the adaptive path algorithm is capable of solving the SOP for. To this end, we fixed the number of time steps to $|V|/10$, where $|V|$ is the number of vertices in G, $\alpha = 0.75$, $k_b = 5$, and $P_f = 0.1$. The length of the initial path was fixed at $n = \lfloor |V|/2 \rfloor$ or half the number of vertices in G. Figure 7 shows the results on 10 unique graphs for each problem size between 100 and 220. The plot also shows the number of non-zero transition probabilities, as they relate to the number of non-zero entries in the linear constraints in Eq. (3). On average, we were able to find solutions on graphs of size $|V| = 220$ ($|n| = 110$) in $1,191$ seconds (less than 20 minutes), using the same computer setup as for the results in Figures 5 and 6. On graphs that contain more than 220 vertices, we were unable to find any solutions using the given parameters within a 24 hour time window due to the limitations of the system we use.

VII. Conclusions

In this work we studied the Stochastic Orienteering Problem (SOP) where travel times between pairs of vertices are continuous random variables. The objective of the SOP is to compute a policy that maximizes the expected reward of visiting vertices in a graph while obeying a budget constraint within a given probability. Our solution builds upon a method from our recent work using a solver for the deterministic orienteering problem to find an initial path through the graph and create a policy using a CMDP architecture for taking shortcuts along that path to reach the goal vertex within the budget and given probability. Our new method is adaptive in the sense that it is no longer bound to a single precomputed path. Instead, a path tree is built where there are multiple routes to the goal vertex, which can be advantageously used by the policy to maximize expected collected reward within the failure probability. Additionally, we introduced a heuristic for limiting the potential computational cost increases for our new method while maintaining useful adaptivity. Our testing results show that the adaptive path method is effective at increasing the expected reward and the branching heuristic is useful for limiting the increase in computation time required for the adaptive method.

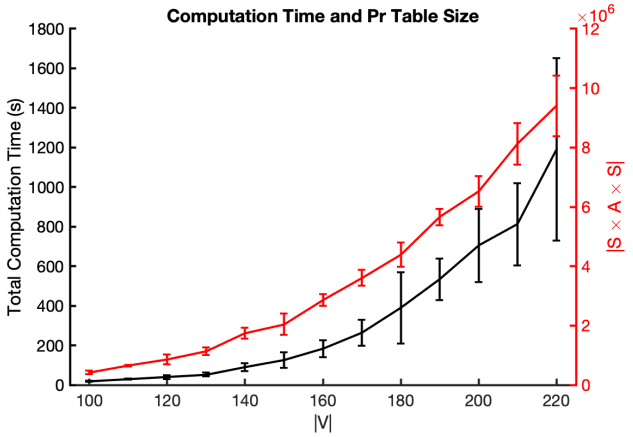

Fig. 7: Average computation time (black, left y-axis) and average number of nonzero probability state/action/state transitions (red, right y-axis) for each when solving the SOP for large size graphs with parameters $\alpha = 0.75$, $k_b = 5$, and $P_f = 0.1$. Error bars indicate ± 1 standard deviation.

There are a few avenues for further research to improve this approach. One avenue is to examine how to adaptively adjust the discretization of time to improve the expected reward and possibly reduce computation time. Another avenue is to explore removing time discretization altogether and instead solve the problem in a continuous time space. These will be the subjects of our future work on this problem.

References

[1] Eitan Altman. *Constrained Markov Decision Processes*. 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, Yin-Lam Chow, and Marco Pavone. Risk aversion in finite markov decision processes using total cost criteria and average value at risk. In *IEEE International Conference on Robotics and Automation*, pages 335–342, 2016.

[7] 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.

[8] 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.

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

[11] 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.

[12] 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.

[13] Xin Huang, Ashkan Jasour, Matthew Deyo, Andreas Hofmann, and Brian C. Williams. Hybrid risk-aware conditional planning with applications in autonomous vehicles. *IEEE Conference on Decision and Control*, pages 3608–3614, 2018.

[14] 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.

[15] 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.

[16] Pedro Santana, Sylvie Thiebaux, and Brian Williams. Rao*: An algorithm for chance-constrained POMDP's. *AAAI Conference on Artifical Intelligence*, pages 3308–3314, 2016.

[17] 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.

[18] 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.

[19] 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.

[20] 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):1–11, 2020.

[21] Felipe Trevizan, Sylvie Thiebaux, Pedro Santana, and Brian Williams. Heuristic search in dual space for constrained stochastic shortest path problems. In *AAAI International Conference on Automated Planning and Scheduling*, pages 326–334, 2016.

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

[23] Pieter Vansteenwegen, Wouter Souffriau, and Dirk Van Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research*, 209(1):1–10, 2010.

[24] 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.