A Fast Algorithm for Stochastic Orienteering with Chance Constraints

Thomas C. Thayer, Stefano Carpin

***Abstract*— We consider the Stochastic Orienteering Problem with random traversal time for edges. In this scenario the length of the path is a random variable and we consider a formulation with chance constraints, i.e., a bound on the probability that the length of the path exceeds the allotted budget. Our proposed solution casts the problem as an instance of a suitably defined Constrained Markov Decision Process and uses a Lagrangian formulation to solve it. In particular, exploiting some structural properties of the associated decision process we can solve the Markov Decision Process using a Lagrangian approach and efficiently determine the optimal Lagrange multiplier. Our method is experimentally evaluated and demonstrated to be significantly faster than previous solutions using a linear programming approach to solve the Stochastic Orienteering Problem with chance constraints.**

I. Introduction

In this paper we consider a stochastic variant of a route optimization problem known as orienteering. The deterministic version of the problem is formulated over a graph G where every edge has a positive cost and every vertex has a positive reward. Given a budget B, the objective is to find a path in G of cost smaller than or equal to B that maximizes the sum of rewards for visited vertices. The Orienteering Problem (OP) has numerous applications in robotics because it is related to problems like warehouse logistics, environmental sampling and precision agriculture, just to name a few [3], [13], [17], [19], [28]. Our previous works studied orienteering on graphs modeled after grape vineyards [14], [22]–[24], and while these proved useful for routing robots through large vineyards, from a practical standpoint the presented algorithms were limited as they did not consider the inherent stochasticity of outdoors conditions for robotic operations (see Figure 1 for an example). In these scenarios, it is useful to model the cost of an edge as the time spent to traverse it (or as the energy consumed to move along the edge), and this parameter is often better described by a random variable rather than a deterministic cost. When this type of uncertainty is introduced, the cost of a path becomes a random variable c, and therefore one is interested in the failure probability, i.e., the probability $\Pr[c > B]$ that the cost exceeds the allocated budget. When one puts a bound on this probability, the formulation is known as *chance constrained* because it aims at bounding the probability of a certain event.

This work was partially supported by the USDA-NIFA under awards # 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 views of the funding agencies. T. Thayer and S. Carpin are with the Department of Computer Science and Engineering, University of California, Merced, CA, USA {tthayer,scarpin}@ucmerced.edu.

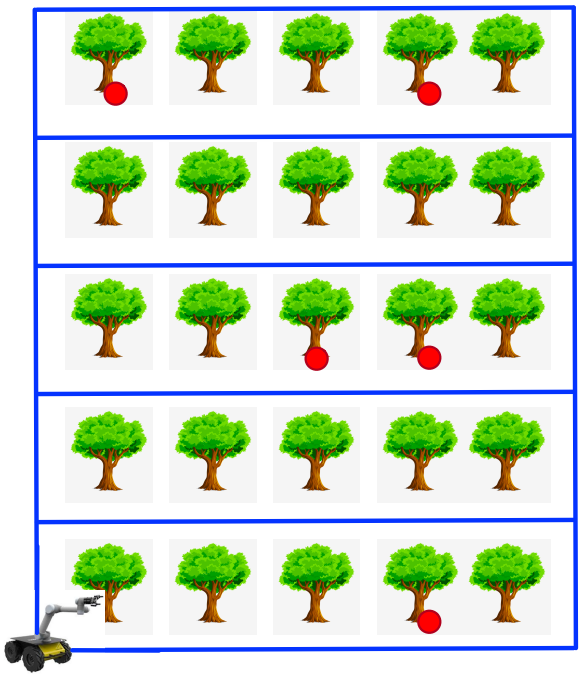

Fig. 1: A robot operating in a vineyard must service certain areas associated with grapevines (shown as red dots) while obeying movement constraints that limit travel between rows (shown as blue lines). While movement costs between service areas may be estimated as deterministic time, this cost is generally stochastic as many factors influence how long a robot takes to move from point to point. Mud, debris, human workers, machinery, etc. may be present in the vineyard and effect movement within it.

In our recent works [20], [21] we used a formulation based on Constrained Markov Decision Processes (CMDP) to compute paths such that $\Pr[c > B] < P_f$, where P_f is an assigned bound on the failure probability. Starting from a pre-computed path, the algorithm computes a path policy that allows taking *shortcuts*, i.e., skipping some vertices along the path to ensure that the failure bound is respected. While this approach works, it does not scale well with the size of the problem because the CMDP formulation is solved using a constrained linear program (LP) that can generate tens of thousands of optimization variables. Motivated by these limitations, in this paper we tackle the same problem using a different approach. Rather than using a LP to solve the CMDP, we use a method based Lagrange multipliers. This involves solving an unconstrained Markov Decision Process (MDP) for a given choice of multipliers and then updating the multipliers themselves. These two problems are iteratively solved in alternation until convergence. In our

work we exploit some of the inherent characteristics of the Stochastic Orienteering Problem (SOP) to expedite the process. First, we utilize a composite state space ensuring that trajectories followed in the state space has no cycles. This guarantees that, for a given choice of Lagrange multipliers, the MDP can be solved with a single pass over the state space. Second, for the chance constraint, we demonstrate that its monotonic structure allows the use of a bracketed root finding method to search for the optimal Lagrange parameter. By combining these two methods together, we are able to solve problem instances at a fraction of the time spent using the LP, and, more importantly, we manage to solve instances that are too large to practically solve with the LP approach. These results are substantiated by a significant number of simulations for large problems. The remainder of this paper is organized as follows. Selected related works are presented in Section II, while in Section III we provide the mathematical background. Section IV shows how a CMDP can be used to formulate a SOP with chance constraints. Our algorithms are presented in Section V, and evaluated experimentally in Section VI. Finally, conclusions are offered in Section VII.

II. Related Work

Orienteering is a combinatorial optimization problem with a long history. The deterministic version was introduced in [11] and shown to be NP-hard. Most of its variants are also NP-hard [12] and it was later on shown that the problem is APX-hard [4]. Because of this intrinsic complexity, numerous heuristic approaches have been developed [12], either for the general case or for special cases [22]. Alternatively, exact solutions based on branch-and-bound techniques were developed [9], as well as approximate solutions [6].

The SOP is an extension of the OP where random costs or rewards are considered, and it has received much less attention. In [5], various heuristic methods were introduced to solve it, but without considering chance constraints. In [26] the authors consider a SOP with chance constraints and propose a method based on sample average approximation and a mixed integer LP formulation to solve it. The authors present results on graphs with few tens of vertices. This approach is later on extended in [27] where a risk-sensitive formulation is introduced, as well as a dynamic version of the problem where travel times along the edges are no longer independent. Different from this work, none of these references use a formulation based on MDPs or CMDPs.

III. Problem Formulation

A. Stochastic Orienteering with Chance Constraints

Let $G = (V, E)$ be a graph where V is the set of vertices and E is the set of edges. Let $r : V \to \mathbb{R}^+$ be a *reward* function associating every vertex to a positive reward. For each edge $e \in E$ let f_e be a probability density function (pdf) defining the stochastic variable modeling the time spent to traverse the edge. We assume that f_e has positive support and finite expectation. Finally, let $B > 0$ be a positive budget. Given a path p in G, the reward of the path $r(p)$ is the sum of rewards of its visited vertices while the cost of the path $c(p)$ is the random variable obtained by summing the random variables associated with each edge along the path. For a given failure probability P_f, the SOP with Chance Constraints (SOPCC) asks to determine a path p that maximizes $r(p)$ such that $\Pr[c(p) > B] < P_f$. As the deterministic OP is a special instance of the SOPCC, it follows that the SOPCC is NP-hard.

B. Sequential Stochastic Decision Making

We provide a short recap on MDPs [2] and CMDPs [1] since we use them to formulate a solution to the SOPCC. A finite MDP $\mathcal{M} = (S, A, \Pr, r)$ is defined as follows:

- S is a finite state space;
- A is a finite set of actions, with $A(s)$ being the set of actions available $s \in S$. We define the combined state/action space as $K = \{(s, a) : s \in S, a \in A(s)\}$.
- $P : K \times S \to [0, 1]$ is a transition kernel which describes the probability $\Pr(s, a, s')$ of moving to state s' after executing action a in state s.
- $r : K \to \mathbb{R}$ is a reward function[1] describing the reward obtained $r(s, a)$ for executing action a in state s.

A *policy* is a deterministic function $\pi : S \to a \in A(s)$ mapping every state to an action. For a given policy π, the sequence of states traversed is a stochastic process and different reward functions can be associated to each realization. For simplicity we consider just the total cost formulation, where we assume that there exists a special absorbing state $s_A \in S$ that is a *sink* state, i.e., once it is reached the state is trapped and no more rewards are accrued. We assume that for each policy π, the absorbing state s_A is reached with probability 1. Under these assumptions the sum of rewards for vertices visited in each realization is a random variable with finite expectation. It is known that for MDPs it is sufficient to consider deterministic policies, and let Π_D be the set of deterministic policies. For a given policy π let $r(\pi)$ be the stochastic variable describing the collected reward. The classic MDP formulation asks to find the optimal policy solving the following optimization problem:

$$\pi^* = \arg \max_{\pi \in \Pi_D} \mathbb{E}[r(\pi)]$$

The definition of an MDP can be extended to a CMDP, which is useful for decision making problems where there are multiple objectives. A finite CMDP $\mathcal{CM} = (S, A, P, r, \beta, c_j, U_j)$ includes the properties of an MDP defined above as well as the following:

- $\beta(s)$ is a probability mass function describing the probability distribution for the initial state $s \in S$.
- $c_j : K \to \mathbb{R}^+$ with $1 \leq j \leq J$ are J cost functions representing costs accrued $c_j(s, a)$ for executing action a in state s.
- $U_j > 0$ with $1 \leq j \leq J$ J are J constants representing upper bounds on accumulated costs c_j.

[1] We intentionally use the same letter used for the reward of the vertices in G because they will be coincident in the following.

To optimally solve a CMDP [1], it is necessary to consider stochastic policies, i.e., policies $\pi : S \rightarrow \mathbb{P}(A(s))$ which associate to each state a probability mass function over the set of available actions $A(s)$. Let Π_S be the set of stochastic policies. We consider again the case of absorbing CMDPs where there is a special absorbing state s_A that is reached with probability 1 under every policy and that once reached does not generate more rewards or costs. In addition to the reward $r(\pi)$, each realization generated by π incurs J costs $c_j(\pi)$ that are the sum of all costs accrued along the realization. Each of the $c_j(\pi)$, as well as $r(\pi)$, are stochastic variables with finite expectation. A CMDP asks to find the optimal policy solving the following constrained optimization problem:

$$\pi^* = \arg\max_{\pi \in \Pi_S} \mathbb{E}[r(\pi)]$$
$$s.t.\ \mathbb{E}[c_j(\pi)] \leq U_j \quad 1 \leq j \leq J$$

This means finding a stochastic policy π^* that maximizes the collected reward $r(\pi^*)$ while also limiting the accumulation of each of the J costs below the upper bounds U_j.

Remark: In literature, discounted cost criteria are more commonly found. Such formulations are appropriate for the case of continuing tasks. The total cost formulation we described above is more appropriate for episodic tasks with finite duration. As we aim at using CMDPs to solve orienteering tasks, the episodic approach and the total cost formulation are chosen because orienteering is an episodic task.

C. Solving Sequential Stochastic Decision Making Problems

MDPs are commonly solved using methods like Value Iteration (VI) and Policy Iteration (PI). These are iterative methods where repeated sweeps through the state space are used to update the value function or policy until convergence is obtained. In the case of VI one stops when the maximum variation of the value function falls below a preassigned threshold θ, while in PI the iteration ends when the policy does not change anymore. For VI, one can then extract the optimal policy from the value function. Both methods are well known applications of the dynamic programming principle. Dynamic programming is however not applicable for the case of CMDPs and therefore different methods are required to compute the optimal policy. One common approach exploits a well known theorem that converts the CMDP optimization problem into an equivalent constrained LP [1]. Our past works in this domain [20], [21] used the LP approach and the reader is referred to our former papers for details. The drawback of this approach is that one can easily generate problem instances with hundred of thousand of optimization variables and very large associated matrices defining the constraints. Alternatively, one can use a Lagrangian approach defined as follows. Let $\boldsymbol{\lambda} \in \mathbb{R}^J$ be a vector with J non negative components. For a given $\boldsymbol{\lambda}$ we introduce the following cost criterion $r^{\lambda}(\pi)$ for a policy π:

$$r^{\lambda}(\pi) = r(\pi) - \sum_{j=1}^{J} \lambda_j (c_j(\pi) - U_j)$$

As shown in [1] (theorem 9.9), a policy π^* is optimal for $\mathcal{CM}$ if and only if

$$r(\pi^*) = \sup_{\boldsymbol{\lambda}} \max \left[r(\pi) - \sum_{j=1}^{J} \lambda_j (c_j(\pi) - U_j) \right].$$

The drawback of the Lagrangian approach is that the search for appropriate multipliers often needs to be done numerically. For each choice of $\boldsymbol{\lambda}$, one has an associated MDP (called a Lagrangian MDP) that can be solved either with VI or PI. Different techniques have been proposed in literature to efficiently search for the optimal vector $\boldsymbol{\lambda}$ [7].

IV. Using CMDPs to solve SOPCC

In this section we establish a CMDP formulation to solve instances of the SOPCC, the basic version of which was introduced in [20]. For a graph G with reward function r defined over its vertices, let f_e be the probability density function (pdf) associated with edge $e \in E$. We start by assuming that a path $p = v_1, v_2, \ldots, v_n$ in G is given. This can be built using any algorithm for the deterministic OP by using $\mathbb{E}(f_e)$ as the deterministic cost for edge e. Path p will respect the budget constraint B in expectation only, and there is no guarantee that the chance constraint $\Pr[C(p) > B] < P_f$ will be satisfied. To ensure this, we introduce the concept of a *path policy* defined over the path p. A path policy over the path may visit all its vertices in sequence, or shortcuts may be taken such that one or more vertices along the path may be intentionally skipped (see figure 2). For example, one may go directly from v_i to some other vertex v_{i+j} using edge (v_i, v_{i+j}) where $2 \leq j \leq (n-i)$. The purpose of a path policy allowing for shortcuts is so that the goal vertex v_n can be reached before the total travel time exceeds the budget B, with a probability of failure P_f. In particular, the path policy keeps track not only of the current vertex along the path, but also of the remaining time. To compute a path policy for a given path p, we frame it as a CMDP $\mathcal{CM} = (S, A, P, r, \beta, c, U)$ with the following properties:

- The state space is $S = (V \times \mathbb{T}) \cup \{s_f, s_A\}$, where $\mathbb{T}$ is a discretization of time with $N = \lceil \frac{B}{\Delta} \rceil$ time steps of length Δ between $t_0 = 0$ and $t_N = B$, and t_k is the interval between $[k\delta, (k+1)\delta)$. The state (v_i, t_j) represents arriving at vertex v_i during time interval t_j. The additional states s_f and s_A are described later.
- Each state has an action set $A_{v_i} = S(v_i)$ where S is the subset of vertices along path p occurring after v_i. Actions deterministically decide the next vertex.
- The probability of landing in successor state (v_j, t_l) from state (v_i, t_k) taking action $a_j \in A$ is $\Pr((v_j, t_l), a_j, (v_i, t_k)) = \int_{t_i\Delta}^{(t_i+1)\Delta} [F(\Delta(t_k+1) - \xi) - F_{v_i,v_j}(\Delta t_k - \xi)] d\xi$ for $v_j = a_j$ and $k \leq l$, where F is the cumulative function of the pdf for edge (v_i, v_j).
- $r((v_i, t_k), a) = r(v_i)$ which is the reward for v_i in G.
- $\beta = 1$ for state (v_1, t_0) and $\beta = 0$ for all other states, i.e., we always deterministically start from the first vertex.

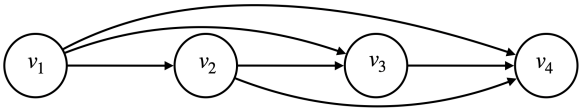

Fig. 2: An example of the shortcuts available to the path policy at each vertex in a hypothetical path of length $n = 4$.

- The failure state s_f defines when B is exceeded, and the probability of landing in s_f from state (v_j, n) with action a_j is $\Pr((v_j, n), a_j, s_F) = \int_{N_s\Delta}^{+\infty} d(\psi - n\Delta)d\psi$.
- All states at v_n and the failure state s_f have only one action moving to s_A, where $\Pr((v_n, t_k), a_A, s_A) = 1$ for all k, and $\Pr(s_f, a_A, s_A) = 1$.
- $c(s_f, a_A) = 1$ and is 0 everywhere else.
- $U = P_f$.

It is easy to show that with this definition one defines an absorbing CMDP, i.e., under every possible policy state s_A will be reached with probability 1. As we have formerly shown [20], one can prove that the following CMDP problem finds a path policy satisfying the chance constraint on the failure probability

$$\pi^* = \arg\max_{\pi \in \Pi_S} \mathbb{E}[r(\pi)] \qquad s.t.\ \mathbb{E}[c(\pi)] \leq P_f$$

The limit of this formulation is that one needs a fine grain decomposition of time (large N) and this generates a very large number of optimization variables if a LP approach is used to find π^*. Additionally, the use of an initial path limits the maximum reward of a policy and therefore provides sub-optimal results.

An extended version of the above process was developed in [21], where the path p is augmented into a *path tree*. A path tree defines "branches" on p such that upon arriving at a certain state (v_i, t_i), an agent can change its course and move toward v_n along a sequence of vertices different from the remainder of the original $v_i \dots v_n$. This new path may contain some vertices in the original path as long as they have not yet been visited, as well as new vertices that were not originally included. The path for each branch is computed in the same way as the original path, that is by utilizing a deterministic OP solver, where the initial vertex is v_i rather than v_1 and the budget is the difference between B and the beginning of t_i. Shortcuts are still allowed, however once a path branch is utilized, it is no longer feasible to take shortcuts to other branches or along the main path. Figure 3 shows how a path tree is connected, and the reader is referred to [21] for complete details regarding this method. With a path tree, the CMDP built to solve the SOPCC has the same properties described above, and therefore the same solution approach can be used to find its optimal policy without modification.

Finally, the SOPCC cast as a CMDP results in a state/action space that is acyclic, meaning the CMDP does not contain actions that allow states to be visited more than once. This acyclic property is provided by two conditions satisfied by this problem

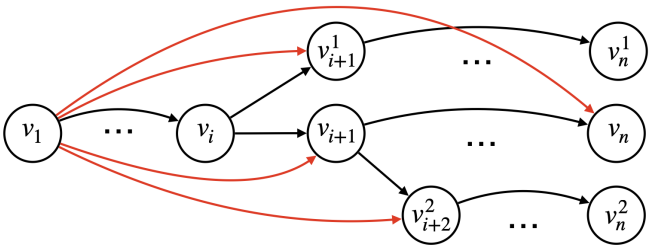

Fig. 3: An example of a path tree with two additional branches. The black arrows represent the flow of the original path as well as the branches. The red arrows are possible shortcuts available to an agent at vertex v_1, which allow it to move anywhere in the path tree.

- For any two states s_i and s_j, if s_j is reachable from s_i, then s_i is not reachable from s_j.
- Any action a leading from state s back to s must have $r(s, a) = 0$ and $c_j(s, a) = 0$ for all $1 \leq j \leq J$.

For acyclic CMDPs (and acyclic directed graphs in general) it can be shown that the state space has a consistent topological ordering, i.e., an arrangement of states that does not change for any trajectory. Formally, it means that for every action $a \in A$ leading from state s to s', s' occurs later in the ordering than s. For example, for a CMDP with a topological ordering $s_i > \dots > s_j$, all trajectories containing both s_i and s_j will have s_i occurring before s_j regardless of states in between. A path policy for the SOPCC will have this property because of the initial path p, which dictates that all vertices not skipped are visited in a certain order. Topological ordering becomes important when proving Theorem 2.

V. Efficiently Solving the SOPCC

Typically, solving CMDPs using the Lagrangian method requires two inner functions which compute a policy π for the current Lagrangian MDP and evaluate π for the unmodified CMDP, and an outer function which loops over the two inner functions and iteratively modifies the value of the vector $\boldsymbol{\lambda}$ until some stopping criteria is met [7]. In this section we show how exploiting some properties of the formulation allows us to efficiently determine the policy solving the CMDP presented in the previous section. Our method consists of a modified version of value iteration to replace the two inner functions, and two alternative ways to perform the λ update based on bracketed root finding algorithms. The Lagrangian MDP is a standard MDP whose immediate reward function parameterized by $\lambda \in [0, 1]$ is

$$r^\lambda(s, a) = (1 - \lambda)r(s, a) - \lambda c(s, a) \tag{1}$$

The following theorem from [1] establishes a relationship between the optimal stochastic policy for the CMDP and the optimal deterministic policies for Lagrangian MDPs.

Theorem 1. *The optimal policy π^* for a CMDP with a single constraint is a randomized mixture of two deterministic policies of its Lagrangian MDP, i.e.,*

$$\pi^* = \sigma\pi^*_{\lambda_1} + (1 - \sigma)\pi^*_{\lambda_2}$$

where $\pi^*_{\lambda_i}$ is the optimal policy for the Lagrangian CMDP for λ_i and σ is a suitable mixing parameter to be determined.

Note that the theorem does not imply that the optimal policy is the mixture of any two deterministic policies for the Lagrangian MDP, but rather that there exists two such policies. As we will see later, it is possible to use a bracketed root finding method to search for the optimal λ value. Classical methods proposed in literature [10] suggest to start the search for the optimal λ between 0 and a value "*sufficiently large*", but it is in general not obvious how to pick this second value. Using the formulation we provide, with $\lambda = 0$ the reward r^λ gives an MDP maximizing only r without considering c, while with $\lambda = 1$ the reward r^λ gives an MDP that minimizes the additional cost c without considering r. Moreover, note that $r^\lambda(s, a)$ is strictly decreasing when λ increases, and consequently the total cost for the associated MDP is also strictly decreasing as a function of λ. As we will show in Algorithm 2, the immediate cost function in Eq. (1) allows for determining a λ value for which the constraint $\mathbb{E}[c(\pi)] \leq P_f$ is active, consistent with the theory of constrained optimization.

Algorithm 1 computes the maximum reward policy according to the r^λ while also determining the value of $R^\lambda(s), R(s), C(s)$ for each state according to the policy using $r^\lambda(s, a), r(s, a), c(s, a)$. This allows us to find a policy for the Lagrangian MDP while simultaneously evaluating the policy for both reward and cost on the original CMDP.

Algorithm 1: Value Iteration

Input : P, r^λ, r, c
Output : R^λ, R, C, π

1 $R^\lambda(s) \leftarrow 0, R(s) \leftarrow 0,\ C(s) \leftarrow 0\ \ \forall s \in S$
2 $\delta \leftarrow \infty$
3 **while** $\delta \geq \theta$ **do**
4 **forall** $s \in S$ **do**
5 $r \leftarrow R^\lambda(s)$
6 $\pi(s) \leftarrow$ $\arg\max_a \sum_{s',r^\lambda} p(s', r^\lambda|s, a)[r^\lambda + R^\lambda(s')]$
7 $R^\lambda(s) \leftarrow \sum_{s',r^\lambda} p(s', r^\lambda|s, \pi(s))[r^\lambda + R^\lambda(s')]$
8 $R(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + R(s')]$
9 $C(s) \leftarrow \sum_{s',c} p(s', c|s, \pi(s))[c + C(s')]$
10 $\delta \leftarrow \max(\delta, |r - R^\lambda(s)|)$

Generally, the while loop at line 3 will execute as long as the max difference between state's value after successive iterations δ is larger than a preassigned constant θ. However, we can exploit the topological ordering of our CMDP to reduce computation down to a single iteration. Consequently, value iteration can be modified to converge in a single iteration through the state space. This is established by the following theorem.

Theorem 2. *Given an MDP where no state can appear twice in a realization and with a single sink state that is reached with probability* 1*, value iteration can converge to the optimum value function in a single iteration.*

Proof. The value function in this case can be computed with a single *backward pass* of the topological ordering starting from the absorbing state s_A. We begin with setting $R^\lambda(s_A) = 0$, a value which will never change regardless of the number of iterations performed, as it has no transitions to other states. Next we consider states that can only proceed to the absorbing state, which must exist because of the topological ordering. Since $R^\lambda(s_A)$ is unchanging, so too is the value of these states. We continue with states that have not been evaluated and can only transition to states that have (following reverse topological ordering) until eventually all states have been evaluated. □

Theorem 2 allows us to remove the while loop in line 3 of Algorithm 1 by strategically arranging states in S according to the reverse topological ordering, as the second iteration of the algorithm will always result in $\delta = 0$. For state spaces with a temporal component (that is, the current state always moves in the positive direction of one component during every transition), this simply means sorting the states according to their temporal component. Because of this, on the Lagrangian MDP for the SOPCC presented in the previous section, Algorithm 1 runs in $\mathcal{O}(S^2)$ time, as we can pre-sort the states in $\mathcal{O}(S \log S)$.

The Lagrangian instantaneous cost in Eq. (1) is a weighted sum of the two cost functions parameterized by λ. However, the optimal λ value must be determined in order to obtain a policy satisfying the constraint P_f, and in particular to make the constraint active. As λ is defined within an interval, we can use any bracketing root finding algorithm to close in on it's appropriate value. Algorithm 2 uses the bisection method to iteratively determine an interval in which the optimal value of λ lays for the given P_f on C, such that the ends of the interval have a difference in reward R no greater than ε. This algorithm allows us to control the precision of the approximation in two ways. The first is

Algorithm 2: Lagrange Bisection

Input : $P, r, c, P_f, \varepsilon, \theta$
Output : $R_{hi}, R_{lo}, C_{hi}, C_{lo}, \pi_{hi}, \pi_{lo}$

1 $\lambda_{hi} \leftarrow 1\ \ \lambda_{lo} \leftarrow 0$
2 $R^\lambda, R, C, \pi_{hi} = ValueIteration(P, c, r, c)$
3 $R_{hi} \leftarrow R(s_0)$
4 $R^\lambda, R, C, \pi_{lo} = ValueIteration(P, r, r, c)$
5 $R_{lo} \leftarrow R(s_0)$
6 $\lambda = 0.5$
7 **while** $(|R_{hi} - R_{lo}| > \varepsilon)$ **and** $(|\lambda_{hi} - \lambda_{lo}| > \theta)$ **do**
8 $r^\lambda(s, a) \leftarrow (1 - \lambda)r(s, a) - \lambda c(s, a); \forall (s, a) \in S \times A$
9 $R^\lambda, R, C, \pi = ValueIteration(P, r^\lambda, r, c)$
10 **if** $C(s_0) > P_f$ **then**
11 $\lambda_{lo} \leftarrow \lambda$
12 $R_{lo} \leftarrow R(s_0)$
13 $C_{lo} \leftarrow C(s_0)$
14 $\pi_{lo} \leftarrow \pi$
15 **else**
16 $\lambda_{hi} \leftarrow \lambda$
17 $R_{hi} \leftarrow R(s_0)$
18 $C_{hi} \leftarrow C(s_0)$
19 $\pi_{hi} \leftarrow \pi$
20 $\lambda \leftarrow (\lambda_{lo} + \lambda_{hi})/2$

with the parameter ε, which provides a maximum bound for the difference in expected total reward ($R(s_0)$) of both policies π_{lo} and π_{hi}. Since we know the expected total cost is monotonically decreasing as λ increases, we also know that all policies between the two computed policies have expected total rewards between them, and thus the maximum deviation from the optimum policy will be no more than ε. The second way to control the precision is with the parameter θ, which provides a stopping condition for Algorithm 2 so that it will terminate execution when the difference between the two λ at the ends of the interval is sufficiently small (and makes the constraint active). This is necessary because the first stopping condition may never be satisfied, as the expected total reward $R(s_0)$ is a step function over λ since MDPs have a finite number of deterministic policies. The optimal value for λ may lay on the edge of a step, and λ_{hi} and λ_{lo} will never converge since they will approach from opposite sides on different steps.

Algorithm 2 as presented stops after $\mathcal{O}(\log(\frac{1}{\varepsilon}))$ iterations in the worst case and in each iteration it executes Algorithm 1. Together with Algorithm 1, the worst case complexity is $\mathcal{O}(\log_2(\frac{1}{\varepsilon})|S|^2)$.

Instead of using the bisection method to find the optimal value for λ, we could use the Illinois False Position method [8] by swapping the formula in line 20 for the following:

$$\lambda \leftarrow \frac{0.5(\lambda_{lo}(C_{lo} - P_f)) - \lambda_{hi}(C_{hi} - P_f)}{0.5(C_{hi} - P_f) - (C_{lo} - P_f)}$$

The Illinois method usually converges faster than the bisection method, however it is not guaranteed to do so. Nevertheless, it is included here as an alternative because it does provide a significant speed improvement on average.

Theorem 1 states that the optimal stochastic policy for the CMDP can be obtained as a mixture of two deterministic policies for the Lagrangian MDP for different values of λ_i, but it does not specify how to find the mixing parameter σ. However, there exists a closed formula for σ (see [16], page 139). The key observation is that for the CMDP to be well posed, the constraint $E[c(\pi^*)] = P_f$ must be active for the optimal policy π^*, and this leads to the following formula.

$$\sigma = \frac{P_f - C_{lo}}{C_{hi} - C_{lo}}$$

For mixing policies, $\pi_1 = \pi_{lo}$ and $\pi_2 = \pi_{hi}$. This algorithm runs in $\mathcal{O}(S^2)$ time. The end results of using Algorithms

Algorithm 3: Policy Mixture

Input : π_1, π_2, σ
Output : π
1 **forall** $s \in S$ **do**
2 **forall** $a \in A(s)$ **do**
3 $\pi(s,a) \leftarrow \sigma\pi_1(s,a) + (1-\sigma)\pi_2(s,a)$

1, 2, and 3 together is a policy for the acyclic absorbing CMDP, computed in $\mathcal{O}(\log_2(\frac{1}{\epsilon})|S|^2)$ time, that satisfies the constraint $V_2(s_0) \leq P_f$ and minimizes the expected total cost $V_1(s_0)$ within ϵ of the optimum.

VI. Comparison to conventional LP methods

To asses the efficiency of our new methods, this section compares the test results for our methods presented in Section V to LP based solvers, using the original SOPCC formulation with a single path p as well as the extended version with a path tree, both of which are described in Section IV.

The vertices $v \in V$ for G are obtained sampling the unit square with uniform distribution, and the reward for each $r(v)$ is a random sample from the $[0, 1]$ uniform distribution. G was made complete by inserted edges for all pairs of vertices. Edge travel times are calculated as

$$\alpha d_{i,j} + \mathcal{E}\left(\frac{1}{(1-\alpha)d_{i,j}}\right) \tag{2}$$

where $d_{i,j}$ is the euclidean distance between v_i and v_j, $\mathcal{E}$ is the exponential distribution, and $0 < \alpha < 1$ is a parameter relating to the variance of edge travel time $((1-\alpha)d_{i,j})^2$. This gives each edge a random, non-negative cost with expectation equal to its euclidean distance. The initial path p is calculated by solving the OP with the S-Algorithm [25] using the expected value for each edge as its travel time.

Starting with the original SOPCC formulation, we varied the length of the path p, because this is a proxy for the size of the CMDP, and thus affects how long it takes to find a solution. The number of vertices in the graph was fixed to twice the length of the desired path, $|V| = 2|p|$. Because B can effect the length of p, we did not fix B, but instead allowed it to vary so that we could obtain consistent lengths of p. Other parameters were also fixed, with $\varepsilon = 0.1$ and $\theta = 0.0001$ to highlight the potential speed increases without giving up much optimality, and $\alpha = 0.5$ along with $P_f = 0.05$ to show the method works even in highly uncertain environments with tight failure constraints. For each length of p, we created 10 random graphs and solved the resulting CMDP using the Lagrangian method with Bisection search and Illinois False Position search from Section V, as well as using the LP formulation with the Interior Point algorithm [18] and the Dual Simplex algorithm [15], [18] . The Lagrangian method with both types of search were coded in Matlab and the built in Matlab solver was used for the LP method. While it is acknowledged that more efficient LP solvers exists, by implementing everything in Matlab we can establish a consistent timeline to evaluate the scalability of the various methods. The tests were run on a Linux computer with an Intel Core i7 6700k processor and 32GB of memory.

Figure 4 shows the average computation time for different lengths of p, as well as the minimum and maximum computation times for the 10 trials at each length. The trends for the different solvers are noticeable after the path reaches 50 vertices. Both the Interior Point method and the Dual Simplex method start to grow in computation time much faster than the methods we propose. They also develop a large variance in computation time while the variance for our proposed methods stays small. For a path p with 100 vertices, it is clear that the Illinois False Position search Lagrangian is the fastest (average time of $21.3s$), followed

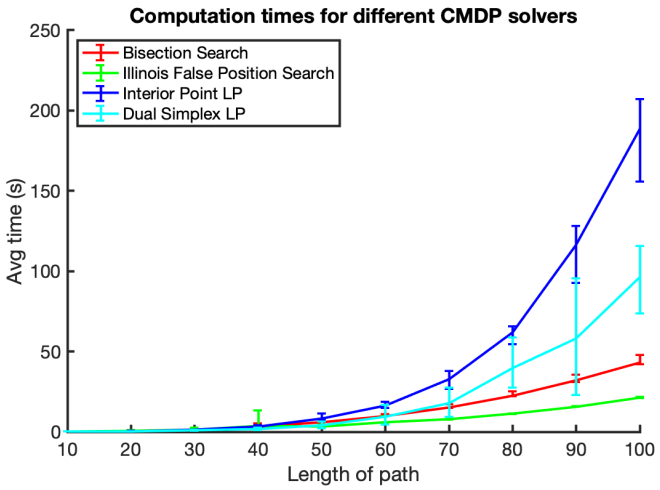

Fig. 4: A comparison of the time to solution for different methods of solving the SOPCC using a CMDP, on lengths of paths up to 100 vertices.

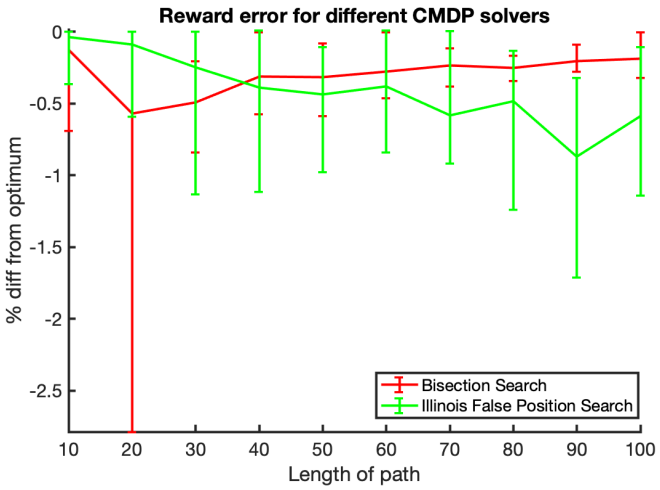

Fig. 5: A comparison of the total reward error for the proposed Lagrangian methods for solving the SOPCC using a CMDP, on lengths of paths up to 100 vertices.

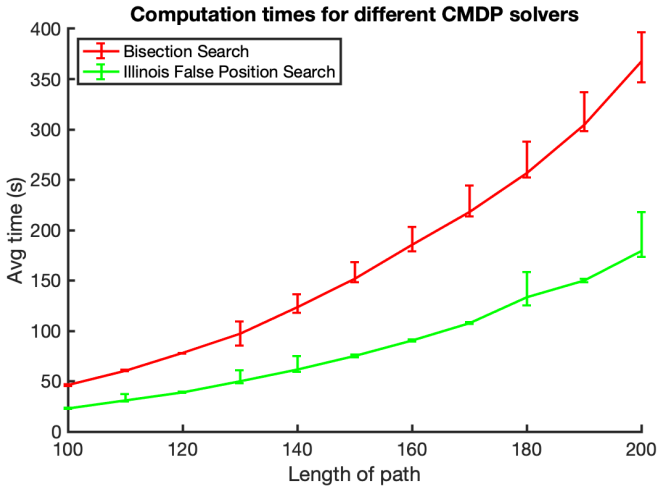

Fig. 6: A comparison of the time to solution for the proposed Lagrangian methods solving the SOPCC using a CMDP, on lengths of paths from 100 to 200 vertices.

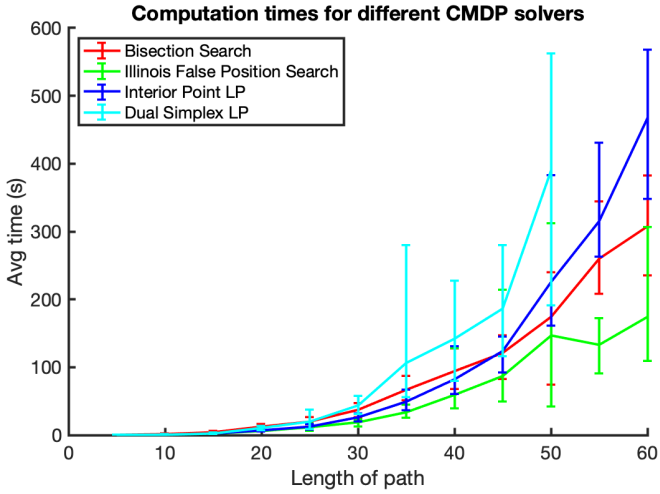

Fig. 7: A comparison of the time to solution for the proposed Lagrangian methods solving the SOPCC using the path tree method, on lengths of paths from 10 to 120 vertices.

by the Bisection search Lagrangian ($43.1s$), then the Dual Simplex LP ($96.3s$), and lastly the Interior Point LP ($188.7s$). These results are to be contrasted to those presented in [26], [27] where their mixed integer sample approximation algorithm gave solutions within $600s$ when $|V| = 63$.

Figure 5 shows the average reward error for the two Lagrangian methods. Since $\varepsilon = 0.1$, there is a maximum of 10% difference from the optimum computed with the LP formulation, however our trials show that in practice the actual error is much smaller than the bound we set. For the Bisection search, the largest error on the reward found was 2.79% when p had 20 vertices, and for every other trial the error was less than 1%. Conversely, a trial with error of 0% was found when p was of lengths 10, 20, and 40. This shows it is indeed possible to find the optimal solution of an CMDP using our methods. For the Illinois search, the largest error was 1.71% when p had 90 vertices, and errors of 0% were achieved when p had 10, 20, 30, 40 and 60 vertices.

For the Lagrangian methods presented in this paper, we ran them on randomized SOPs with path lengths up to 200 vertices. Results of these trials are shown in Figures 6. The LP methods were not included in these trials because time and memory constraints made them impractical. As seen in Figure 6, the trend for computation time continues, with both methods showing a quadratic increase in time for the path length. Every trial of the Illinois False Position Search method was faster than the Bisection Search method.

Finally, we show that our proposed Lagrangian approaches for solving CMDPs work just as well on the more complex path tree SOPCC method. Figure 7 shows how favorable they are compared to LP based solvers. Again we varied the length of the initial route as a proxy to state space size, however because the path tree method works by augmenting the path with branches to other vertices, this method produces CMDPs with much larger state and action spaces. Consequentially, all of the CMDP solvers required significantly

more time to produce a solution. In particular, the Dual Simplex LP took on average $391s$ to obtain a solution when the initial path contained 100 vertices, versus the Illinois search which took on average $147s$ for the same problems. Beyond 100 vertices, the Dual Simplex method took too long to complete and thus no results were obtained. Likewise, beyond 90 vertices in the initial path, the Bisection search Lagrangian performed better than the Interior Point LP, with results showing a diverging trend in favor of the Bisection search. Again in all cases, the Illinois False Position search Lagrangian method was the fastest.

VII. Conclusions

In this work we studied the Stochastic Orienteering Problem with Chance Constraints (SOPCC), using Constrained Markov Decision Processes (CMDP) as a solution method. We utilize our previous work that presents a CMDP formulation for the SOPCC by reducing the problem to a deterministic version of Orienteering and finding an initial optimistic path over which a path policy can be computed. The path policy utilizes a composite state space of both vertex and time, and allows for shortcuts to be taken in the path so that the time budget can be observed with respect to a given failure probability. Our previous method utilized linear programming (LP) to find the optimal path policy, however this method takes considerable time to compute. Here, we present a much quicker method to compute the path policy by taking a Lagrangian approach to solving the CMDP. We use a bracketed root finding algorithm to find two Lagrangian multipliers close to the optimal, which allow us to create a mixed policy that is within a given error bound of the optimal policy. Dynamic programming is used to compute each Lagrangian MDP, and we exploit the inherent topological ordering of our state space to speed up the Value Iteration algorithm to be quadratic on the size of the state space. Finally we experimentally show that our Lagrangian approach for solving the SOPCC is very quick to converge, beating two different LP based solvers and giving policies that are within acceptable bounds of the optimal.

References

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

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

[3] Graeme Best and Geoffrey A. Hollinger. Decentralised self-organizing maps for the online orienteering problem with neighbourhoods. In *IEEE International Symposium on Multi-Robot and Multi-Agent Systems*, pages 139–141, 2019.

[4] Avrim Blum, Shuchi Chawla, David R. Karger, Terran Lane, Adam Meyerson, and Maria Minkoff. Approximation algorithms for orienteering and discounted-reward tsp. *SIAM Journal on Computing*, 37(2):653–670, 2007.

[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] 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.

[7] Dejan V. Djonin and Vikram Krishnamurthy. Q-learning algorithms for constrained markov decision processes with randomized monotone policies: Application to mimo transmission control. *IEEE Transactions on Signal Processing*, 55(5):2170–2181, 2007.

[8] Mark Dowell and Peter Jarratt. A modified regula falsi method for computing the root of an equation. *BIT Numerical Mathematics*, 11:168–174, 1971.

[9] Matteo Fischetti, Juan José Salazar González, and Paolo Toth. Solving the orienteering problem through branch-and-cut. *INFORMS Journal on Computing*, 10(2), 1998.

[10] Bennett L. Fox and Dale M. Landi. Searching for the multiplier in one-constraint optimization problems. *Operations Research*, 18(2):193–373, 1970.

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

[12] 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.

[13] 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.

[14] 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.

[15] Achim Koberstein. Progress in the dual simplex algorithm for solving large scale lp problems: Techniques for a fast and stable implementation. *Computational Optimization and Applications*, 41:185–204, 2008.

[16] Vikram Krishnamurthy. *Partially Observed Markov Decision Processes*. Cambridge University Press, 2016.

[17] Moshe Mann, Boaz Zion, Dror Rubinstein, Rafi Linker, and Itzhak Shmulevich. The orienteering problem with time windows applied to robotic melon harvesting. *Journal Of Optimization Theory and Applicationos*, 168:246–267, 2016.

[18] Jorge Nocedal and Stephen Wright. *Numerical Optimization*. Springer, 2nd edition, 2006.

[19] Dinesh Thakur, Maxim Likhachev, James Keller, Vijay Kumar, Vladimir Dobrokhodov, Kevin Jones, Jeff Wurz, and Isaac Kaminer. Planning for opportunistic surveillance with multiple robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5750–5757, 2013.

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

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

[26] 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.

[27] 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.

[28] Jingjin Yu, Mac Schwager, and Daniela Rus. Correlated orienteering problem and its application to persistent monitoring tasks. *IEEE Transactions on Robotics*, 32(5):1106–1118, 2016.