Solving Large-scale Stochastic Orienteering Problems with Aggregation

Thomas C. Thayer Stefano Carpin

***Abstract*— In this paper we consider the stochastic cost orienteering problem, i.e., a version of the classic orienteering problem where the cost associated with each edge is a random variable with known distribution. Such a model is relevant when travel costs are variable, e.g., when a robot moves in uncertain terrain conditions. We model this problem using a composite state space tracking both how much progress the robot has made towards the goal and how much time it has left. On top of this state space, we compute a time-aware policy that allows the robot to dynamically adjust its path and avoid missing the temporal deadline. This policy is determined using a Constrained Markov Decision Process that allows tuning the accepted failure probability upfront. This approach suffers from a significant growth in the composite state space, and to mitigate this problem we introduce an aggregation technique where nearby vertices are compounded together, effectively reducing the original routing problem to an instance with a smaller state space. We then analyze this approach over large scale problem instances associated with robotic irrigation on a commercial grade vineyard.**

I. Introduction

Orienteering is a combinatorial optimization problem defined on undirected graphs where each vertex has a reward and each edge has a cost. The objective is to determine a path collecting the maximum possible reward whose cost does not exceed a preassigned budget B. There exists multiple variants of the problem that make different assumptions. For example, the start and final vertices may be assigned, the path may be constrained to be a circuit, there may be a team of agents, and many more. Most of these problems are known to be NP-hard (see Section II for more details). Many problems arising in robotics can be conveniently studied as instances of the orienteering problem, where the budget models the constraint requiring a robot to periodically stop and recharge its batteries, and it is therefore important that it reaches a suitable location where recharging is possible before it runs out of energy.

In our recent works [16], [17], [18], [19] we presented our ongoing RAPID project where robots are deployed in a vineyard to make adjustments to variable rate emitters, accurately regulating the amount of water delivered to each vine (see figure 1). Given the large scale of commercial vineyards, the robot cannot exhaustively visit all emitters, but rather should focus on adjusting those where the mismatch

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

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

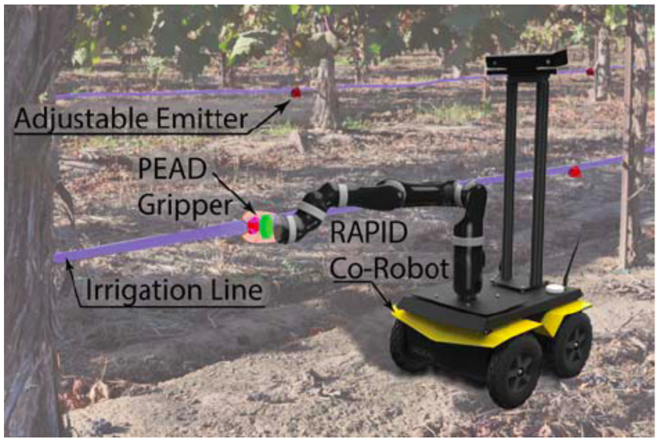

Fig. 1. A robot adjusting water emitters placed on irrigation lines (from our former paper [17]).

between the current and desired setting is large. Moreover, the robot needs to carefully plan its path to make sure it can return to the deployment site or recharging station before its battery is completely depleted. For this problem the orienteering framework offers a convenient formalism, but its inherent computational complexity calls for the development of domain-specific heuristics that can solve instances with graphs containing large numbers of vertices (more than $50,000$).

Most classic literature on the orienteering problem considers the deterministic case where costs and rewards are known constants. However, when the orienteering framework is used to study robotics problems, a stochastic model for these quantities is often more appropriate. For example, the time it takes to move between two locations is in general variable, and so may be the reward collected when reaching a certain place of interest. In our project, the largest uncertainty is in the time it takes to move from one emitter to the next, or equivalently, the travel time between two vertices in the graph. Additionally, when a robot physically reaches an emitter, through visual seroving [2] it must latch its gripper to the emitter and perform the desired adjustment. This phase may require more or less time, as the robot may need to repeatedly reposition itself to successfully turn the emitter. In this case, as the robot progresses through its path, it should consider that the remaining time is a random variable and therefore should adjust its path *on the fly*. This adjustment, while possibly eliminating visits to some areas of interest, helps to contain the risk of the robot emptying its battery before it returns to a location where it can recharge.

In this paper we solve the problem using an approach in three stages. First, an initial path solving the orienteering problem utilizing the expected travel time of edges is determined using one of the heuristic algorithms we formerly proposed in [17]. Then, this path is used to produce a routing policy π that, based on where the robot is along the path and how much time is left, may decide to execute a *shortcut* and skip some vertices, ensuring that the final location is reached before the deadline expires. This policy is formulated on a composite state space featuring both position and time, and computed using a framework based on *Constrained Markov Decision Processes* (CMDPs). This last step, however, does not allow solving very large problems because explicitly tracking time causes the number of states to quickly grow leading to instances that cannot be solved quickly (or at all because they require too much memory). To address this issue, in between the two steps we introduce an aggregation stage where successive vertices in the path are logically combined into a single vertex, allowing control over the number of states. To the best of our knowledge this is the first paper providing an aggregation solution the orienteering problem with stochastic costs that allows solving arbitrarily large instances. Our findings are validated on real world problems using data sets collected in a commercial vineyard.

The remainder of this manuscript is organized as follows. Related work is presented in Section II, while the problem formulation is given in Section III. Next, in Section IV we introduce suitable models for computing time aware policies and in Section V we develop our overall strategy including the aggregation step. Our methods are validated on large data sets in section VI, and conclusions are given in Section VII.

II. Related Work

The *Orienteering Problem* (OP) was first formally introduced in [20], shown to be NP-hard in [11], and demonstrated as APX-hard in [5]. With a given graph, a reward function and a cost function on the graph's vertices and edges, the goal of the OP is to compute a path that maximizes the sum of rewards collected for visiting individual vertices while constrained to a limited travel budget. Rewards for each vertex can only be collected once, while costs are accrued during every traversal of an edge. As mentioned in Section I, there are many variants of this problem. A very common formulation fixes both the start and end vertices of the path, which is practical for real world applications, but in general is not necessary. Others may include time windows, multiple objectives (with multiple reward functions), time dependent travel costs and rewards, a team of agents, etc. For a more in-depth discussion of the OP, the reader is encouraged to review [12] and [21].

While most OP variants have exact solution methods available, due to the intrinsic computational complexity these can only be used on graphs with a restricted number of vertices (with the most efficient methods being unable to solve instances with more than $1,000$ vertices). A plethora of heuristic methods exist to solve larger problems, but many of these also run into scalability issues. Our previous work [17] considered the OP on a special class of graphs called *Irrigation Graphs* (see figure 4 for an example). Although this variant of the problem is still NP-hard, we proposed domain specific heuristics that are able to efficiently solve instances with over $100,000$ vertices.

This paper investigates the *Stochastic Cost Orienteering Problem* (SCOP), whereby the edge costs are unknown at the time of computation and are modeled as random variables following some known distribution. The SCOP is a special case of the *Stochastic Orienteering Problem* (SOP) - which can have both stochastic costs and rewards. In most cases, the stochastic variables all have the same type of random distribution (normal, exponential, etc.) but usually have different distribution parameters and are assumed to be independent. A literature review revealed a limited body of research on the SOP but exact, approximation, and heuristic methods do exist, usually for special cases or variants of the SOP. For example, [6] provides a general heuristic and exact methods for some specific cases, considering the case where travel times and service times (reward collection cost at vertex) are stochastic and penalties are incurred when a job (vertex) is committed to the path but goes unserviced. The methods given in [13] and [14] provide approximation algorithms for different methods of solving the SOP and prove the existence of an $\Omega(\log\log B)$-approximation algorithm with $\mathcal{O}(\log\log B)$ run time. In [8] the SOP is solved while considering the adaptivity of tours with precomputed policies and online processing of newly revealed edge costs. Finally, [9] provides a solution method for smaller problems that converges to the optimum, and also a heuristic method that is appropriately applied on larger instances.

In this work, we consider large problem sizes with tens of thousands of vertices, which are too large to solve using previous methods, and thus we are motivated to find a more scalable solution approach combining heuristics and time sensitive policies.

III. Problem Formulation

Let $G = (V, E)$ be an undirected graph, where V is the set of vertices and E is the set of edges. Without loss of generality let us assume G is a complete graph, i.e., $E = V \times V$. Let $r : V \to \mathbb{R}^+$ be the function associating every vertex to a reward. For every edge $e_i \in E$, let d_i be a probability density function with non-negative support and finite expectation $\mathbb{E}[d_i]$. The distribution describes the random variable for the stochastic time it takes to traverse the edge in either direction. We assume that traversal times are independent random variables, and we do not require them to be identically distributed. Finally, let B be a positive constant travel budget. A path $\mathcal{P} = v_1, v_2, \ldots, v_m$ is a sequence of vertices in V, and the reward obtained traversing the entire path is $r(\mathcal{P}) = \sum_{i=1}^{m} r(v_i)$ with the additional constraint that if a vertex is visited more than once its reward is collected only once. The traversal time for the path is a random variable $T(\mathcal{P}) = D_1 + D_2 + \cdots + D_{M-1}$ where each of the D_i is a random variable for the time it takes to go from vertex v_i to vertex v_{i+1} along the path. Moreover, for $i \geq 2$, let $T_i = D_1 + D_2 + \cdots + D_{i-1}$ be the random variable for the time spent to reach vertex v_i along the path. While the statistical properties for $T(\mathcal{P})$ and T_i can be studied off-line, it is only while the path is executed that specific realizations for those variables are obtained. For the time being we assume that the path $\mathcal{P}$ is given and that $\mathbb{E}[T(\mathcal{P})] \leq B$, i.e., the expected time to traverse the path is not larger than the given budget for the SCOP.

A path can be seen as a policy $\pi : V \to V$ that, for each vertex, determines where to go next. Without additional information, such a policy is *open loop*, i.e., $\pi(v_i)$ is always v_{i+1} and is not informed by the value T_i. Therefore, it is unable to adjust the path on the fly and cannot, for example, bypass some vertices in the path if there is a high probability that $T(\mathcal{P}) > B$ with the current value of T_i. In the following we study different time-aware policies that will explicitly consider this time dependency and take actions in a principled way. The general idea is depicted in figure 2. At each vertex along the path the robot can decide to move towards any of the following vertices (but not the previous), possibly reducing the time necessary to reach the last vertex v_m. Of course, by skipping vertices along the path the robot will also miss some of the rewards, so it is necessary to strike a balance between the reduction in collected reward and reduced probability of missing the deadline.

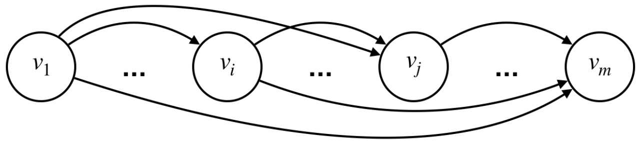

Fig. 2. The connectivity of vertices in the path. For $1 < i < j < m$, a robot at v_1 may travel to any arbitrary vertex in $\mathcal{P}$, a robot at v_i may travel to any vertex v_j in $\mathcal{P}$ occurring later than v_i, and eventually $\mathcal{P}$ terminates at v_m.

IV. Time Aware Policies

To account for the stochastic nature of the travel time between vertices and to decrease the chances of violating the temporal deadline B, we consider a model based on CMDPs with composite states tracking both the vertex where the robot is positioned and the time spent since starting the path. The goal is to maximize in expectation a primary objective function (collected reward) while ensuring that a second objective function (travel cost) is bound in expectation.

A. CMDP Background

A finite *Markov Decision Process* (MDP) $\mathcal{M} = (S, A, \Pr, r)$ features the following elements (see [3] for a comprehensive introduction):

- S is a finite state space
- A is a finite set of actions. In general different actions are available in different states, and therefore $A(s)$ indicates the set of actions available at $s \in S$. In the following it will be convenient to use the *state/action* space defined as $K = \{(s, a) : s \in S, a \in A(s)\}$
- $\Pr : K \times S \to [0, 1]$ is a *transition kernel* for the transition probabilities, i.e. $\Pr(s, a, s')$ is the probability that the next state is s' when action a is executed from state s
- $r : K \to \mathbb{R}^+$ is the reward function, i.e. $r(s, a)$ is the reward obtained when action a is executed from state s

Extending that, a CMDP [1] is defined as $\mathcal{CM} = (S, A, \Pr, r, \beta, r, c_j, U_j)$ where $S, A, \Pr, r$ are defined as above and

- β is a mass distribution over S describing the probability distribution of the initial state [1]
- c_j are J cost functions $c_j : K \to \mathcal{R}^+$
- U_j are J given positive constants

Under this CMDP formulation, a policy is a function $\pi : S \to \mathbb{P}(A)$ that associates to each state s a probability mass distribution over the set of actions. Indeed, one notable difference between MDPs and CMDPs is that the optimal policy for a CMDP is in general randomized and so in general the policy maps states into random actions. Executing a policy over a CMDP results in the collection of a reward $r(s, a)$ and the accumulation of J costs $c_1(s, a), c_2(s, a), \ldots, c_J(s, a)$. Given a start state $s \in S$, a policy π induces a trajectory, i.e., a sequence of states defined by the policy and the transition kernel.

In the associated literature, one normally finds two types of (C)MDPs. The first one is the finite horizon (C)MDP where it is a-priori established that the (C)MDP will execute only N transitions, where N is given constant. In this case, the reward associated with a trajectory is $\sum_{i=1}^{N} r(s_i, \pi(s_i))$. The second one is the discounted infinite horizon (C)MDP where an infinite number of transitions are executed, but rewards are scaled by a discount factor $\gamma < 1$. In this case the reward associated with a trajectory is $\sum_{i=1}^{+\infty} \gamma^n r(s(i), \pi(s_i))$. Both choices ensure that the sum of all rewards is bounded under any possible trajectory. On the contrary, we do not make these assumptions because they are inappropriate for our problem, and instead impose that the CMDP is *absorbing*. This means that there exists a special state $s_A \in S$ with a single action a_A such that:

1) for each policy π, the induced trajectory eventually enters s_A with probability 1
2) $r(s_A, a_A) = 0$
3) $\Pr(s_a, a_A, s_A) = 1$
4) $c_j(s_A, a_A) = 0$ for $1 \leq j \leq J$

The absorbing property ensures that under every policy the absorbing state s_A is eventually entered, where it does not accumulate any more reward, and from which it cannot leave or accrue any additional cost. Therefore, the reward $\sum_{i=1}^{+\infty} r(s(i), \pi(s_i))$ and J costs $\sum_{i=1}^{+\infty} c_j(s(i), \pi(s_i))$ are still bounded for each policy, even though it is neither discounted nor based on a finite horizon assumption.

For a given CMDP $\mathcal{CM}$, the reward and costs are random variables indicated as $r(\pi)$ and $c_j(\pi)$ to outline their dependency on the policy π. The CMDP problem, therefore, is to

[1]For CMDPs the optimal policy in general depends from the distribution β. However, in our problem the start state will be known with certainty and therefore the dependency on β is dropped. The reader is referred to [1] for details about how β influences the policy.

compute

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

i.e., we want to find a policy that maximizes in expectation the accumulated reward $r(\pi)$ while at the same time ensuring that each of the J additional costs does not exceed the associated upper bounds U_j in expectation.

While MDPs and CMDPs share many traits, there are a couple of differences with regard to policies. First, for MDPs the optimal policy can be determined by writing the Bellman equation for the value function and then solving iteratively using the Value Iteration algorithm. To determine the optimal policy for a CMDP problem, one needs to write a constrained linear optimization problem with a number of variables equal to $|K|$, i.e., one for each state/action pair. Second, it is well known that one can determine the optimal policy for an MDP by considering only deterministic time-invariant policies, however the optimal policy for CMDPs is generally stochastic. Details about the CMDP linear program formulation can be found in [1], [10].

B. Stochastic Orienteering using CMDPs

Starting from what we discussed so far, we now introduce a CMDP built upon a common bi-dimensional state space S where the first component is a vertex along the path and the second component is a temporal interval. More specifically, let N_s be a discretization step for the temporal deadline B and let $\Delta = B/N_s$. States in S are of the type (v, n) where $v \in V$ and $n \in \{1, \ldots, N_s\}$. State (v_j, i) models the fact that the robot is at vertex v_j and the random variable D_j is in the interval $[(i-1)\Delta, i\Delta)$. In addition, we add an *absorbing state* s_A to make the CMDP absorbing, as per our previous definition. Finally, we add a *failure state* s_F modeling the event that the robot has violated the temporal deadline B.

Next, we introduce the action set A. For each state (v_j, i) with $j < m$ the action set is v_k for $j + 1 \leq k \leq m$ (recall that the path has m vertices). This means that at every vertex along the path, the policy may select the next vertex as one of any of the remaining vertices, i.e., it can execute a shortcut like the one illustrated in Fig. 2. States of the type (v_m, i), i.e., states associated with the last vertex in the path, have a single terminal action a_T that will deterministicly go to the absorbing state s_A. States s_F and s_A, too, have a single action (a_F and a_A, respectively). For the state (v_j, n) with $j < m$ and action v_k the transition kernel depends on the probability density d associated with the edge (v_j, v_k). Specifically, the transition probability between states (v_j, n) and (v_k, l) for $l \geq n$ is

$$\Pr((v_j, n), v_k, (v_k, l)) = \int_{(l-1)\Delta}^{l\Delta} d(\psi - n\Delta) d\psi$$

Conversely, the transition probability between (v_j, n) and (v_k, l) for $l < n$ is 0 because the robot cannot move backwards in time. For action v_k, the transition probability between (v_j, n) and s_F is given by the probability that the temporal deadline B expires before the robot reaches v_k and is given by the following expression:

$$\Pr((v_j, n), v_k, s_F) = \int_{N_s\Delta}^{+\infty} d(\psi - n\Delta) d\psi$$

Finally $\Pr((v_m, i), a_T, s_A) = \Pr(s_F, a_F, s_A) = 1$, i.e., when the robot reaches the last vertex along the path or enters the failure state s_F, it deterministicly moves to the absorbing state s_A. For the absorbing state s_A, as per its definition, $\Pr(s_A, a_A, s_A) = 1$, i.e., the state cannot be left once it is entered.

A CMDP also requires the reward and cost functions to be defined. For the state s_F we set $r(s_F, a_F) = 0$ and for s_A we set $r(s_A, a_A) = 0$. For all other states $s(v_j, i)$ we define $r((v_j, i), v_k) = r(v_k)$, i.e., the reward obtained when v_k is reached. We next introduce a the cost c_1 as $c_1((v_j, n), v_k) = 0$ and $c_1(s_F, a_F) = 1$, i.e., a unit cost is incurred only when the state enters s_F. For this CMDP model, the significance of introducing the failure state s_F and the cost c_1 that is 0 everywhere except for (s_F, a_F) is given by the following theorem:

Theorem 1. *Let $\mathcal{CM}$ be a CMDP defined as above. Then, for any policy π, $\mathbb{E}[c_1(\pi)]$ is the expected probability that a trajectory induced by π goes through state s_F, i.e., it is the probability that by time B the robot has not yet reached the final vertex of the path v_m.*

The detailed proof of this theorem is omitted for lack of space, but it follows the same steps for the proof of Theorem 1 found in our former work [7]. Therefore, in the context of stochastic orienteering, the CMDP formulation can be used to determine policies that will miss the deadline with a probability no larger than the constant U_1.

There is an obvious tradeoff between the number of time steps and the computational time. A finer grain discretization is of course preferable, but as it causes a growth in the size of the composite state space. In section VI we will discuss this with more details.

V. Methods

A. Initial Path Creation

In general, the initial path $\mathcal{P}$ can be created using any method that solves a deterministic OP. When computing the initial path, the traversal time between two arbitrary vertices v_x and v_y is $t(v_x, v_y) = \mathbb{E}[D(v_x, v_y)]$, where $D(v_x, v_y)$ is the distribution of travel times between the two vertices. If this sequence of vertices is included in the path, then $v_i \to v_x$, $v_{i+1} \to v_y$, and $D_i \to D(v_x, v_y)$. The total traversal time $\mathbb{E}[T(\mathcal{P})] = \sum_{i=1}^{M} \mathbb{E}[D_i]$ must not exceed the budget B set for the original SCOP.

B. Vertex Aggregation

While CMDPs work well for computing optimal policies, they are limited in the sizes of problems that can be solved due to computation time and memory restrictions. In particular, the SCOP we wish to solve may have paths containing arbitrarily long sequences of vertices. To overcome this issue, we introduce the concept of vertex aggregation, where sequential vertices in a path $v_i \dots v_j$ for $1 < i < j < m$ can be aggregated into a single compound vertex $v_{i,j}$ that represents a section of the path containing the aggregated vertices (see figure 3 for connectivity). Here, the reward for the compound vertex is $r(v_{i,j}) = \sum_{k=i}^{j} v_k$ and the travel time to the compound vertex from any vertex v_x outside of the compound is $D(v_x, v_{i,j}) = D(v_x, v_i) + \sum_{k=i}^{j-1} D(v_k, v_{k+1})$.

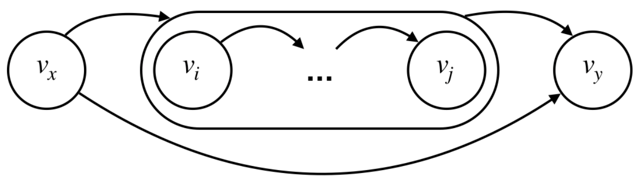

Fig. 3. An example of the connectivity of a compound vertex. v_x can be any compound vertex or non-aggregated vertex occurring in the path before v_i. v_y can be any compound or non-aggregated vertex in the path after v_j

Aggregation techniques for reducing the state space of (C)MDPs have been used in the past (see [4] for an overview). However, these techniques usually work by compressing the state space, meaning similar states are grouped together and special care must be taken when computing the transition kernel since the states aggregated together in general have unique transitions. On the contrary, we propose aggregating vertices before building the state space, explicitly defining the entry/exit vertices and the path between them such that the travel time to compound vertices is merely a sum of random variables.

A compound vertex has special connectivity requirements. A compound vertex must be entered at the first vertex in the aggregation v_i and exited at the last vertex in the aggregation v_j. The aggregated vertices within a compound vertex are connected sequentially as in $\mathcal{P}$ but all costs and rewards are incurred at the same time. This means that the travel times for traversing the vertices within the compound are pushed in front of the compound, such that all vertices aggregated together must be visited before a robot can be considered at $v_{i,j}$. This also has the effect of masking the true collected reward until the robot arrives at v_j, the last vertex in $v_{i,j}$, because the rewards are pushed towards the end of the compound. Thus, going over the temporal budget B sometime before reaching v_j (and thus $v_{i,j}$) means not collecting any rewards in $v_{i,j}$. A path may contain any number of vertex compounds of various sizes as long as v_1 and v_n are not included in any compound. This is to ensure that the computed policy π is not unnecessarily forced to visit vertices that are not v_1 and v_n.

C. SCOP on Irrigation Graphs

Keeping with the focus of the RAPID project, we direct our attention to building solutions for the SCOP on instances where a robot must traverse a vineyard. That is, the graph is of the form $IG(h, w)$, which is an *Irrigation Graph* (IG) with h rows and w columns. More information about IGs can be found in [17]. The IG has an associated reward function $r(v_x);\ v_x \in V$ and time distribution $D(e) = t_{min} + Exp(1 - t_{min});\ e \in E$, where t_{min} is a constant minimum bound from 0 to 1 and is the same for all edges. Note that this time function gives in expectation a value of 1 for all edges in the IG, making it applicable to the deterministic IG Constant Cost OP discussed in [17] (The expected time for each edge can be any real non-negative value, here it is defined as 1 only for convenience). The time distribution was chosen as a shifted exponential due to the physical nature of movement, i.e. we do not expect the robot to ever finish a movement faster than t_{min} but it may take an unexpectedly long time to finish.

First we compute an initial tour using the Greedy Partial Row (GPR) heuristic given in [17]. Due to the design of GPR, the output will be a path that contains a series of full-rows (total traversal across a row in the IG from one outside column to the other) and partial-rows (traversal partway into a row and back to the same outside column). Logically, the sequence of vertices visited by a full-row or partial-row will always be the same regardless of π, and they therefore can be aggregated into a set of compound vertices (as described in section V-B) with one full-row or partial-row per compound. Because our travel time function is a shifted exponential distribution, we can easily model the new travel times between compound vertices using a shifted gamma distribution $D(v_x, v_{i,j}) = k \cdot t_{min} + \Gamma(k, 1 - t_{min})$, where k is the number of vertices passed in the IG along the path from v_x to $v_{i,j}$.

Next, these compound vertices can be compounded again to reduce the number of vertices and therefore states for our CMDP. One special consideration that must be made for paths over IGs is that the next location (full or partial-row) in a path must enter from the same side of the IG that the previous location exited on. That is, the robot cannot service a row from the left side of a vineyard if it is on the right side of the vineyard. It must first traverse a full row before it can service from the other side. The connectivity between compound vertices must enforce that constraint, where compounds that end on one side of the IG are only connected to future compound vertices that begin on the same side. Figure 4 shows this connectivity restriction.

Finally, a CMDP used to solve the SCOP on IGs can be constructed from the path with compound vertices without additional modification. The two variables that effect computation time are the number of compound vertices in the path and the granularity of time discretization, since these determine the total number of state action pairs.

VI. Results

To test the effectiveness of our technique, we applied it to multiple SCOPs of large size built from 9 different IGs based on a real-world commercial vineyard in California, USA. These IGs contain $h = 275$ rows and $w = 214$ columns (vines per row) totaling $58,850$ vines and therefore vertices in the graph. Data collected for our previous work [18] was used to determine the reward for each vertex. The vineyard was sampled for soil moisture using a Hydrosense HS2P

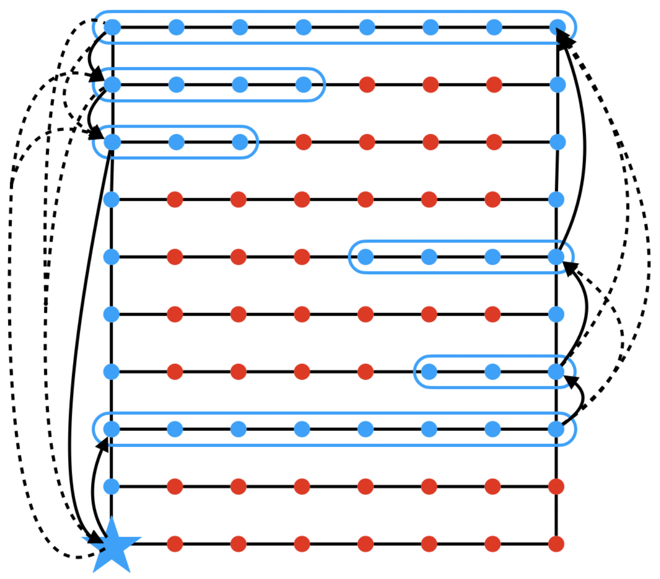

Fig. 4. An example showing the connectivity of compound vertices on an IG. The blue dots represent vertices in the original path, with each full row or partial row grouped into a single compound vertex, and the blue star represents the start and end vertex of the path. Dashed arrows represent taking shortcuts and skipping one or more compound vertices.

from Campbell Scientific at 72 equally spaced locations on 9 separate days throughout the growing season. Moisture values for locations between those sampled were estimated using Kriging interpolation [15]. Reward for each vertex was determined as the difference between a target soil moisture value and the interpolated value obtained at that vertex location.

The following results are averages compiled from the 9 IGs run with varying parameters, keeping the budget B and the minimum transition cost t_{min} fixed, and normalizing for the reward collected by the GPR when computing the initial deterministic path. This allows for a fair comparison across each IG. The minimum edge traversal time t_{min} was set to 0.75 and the failure probability $U_1 = \Pr[T(\mathcal{P}(\pi)) > B]$ was fixed at 5%, however results look similar across a range of different t_{min} and $\mathbb{E}[c_1(\pi)]$ values. For the value of B used (which was fixed at $1/4$ the traversable distance of the vineyard), the initial paths contained an average 14851 vertices, consisting of 76 full or partial rows. Therefore, these were aggregated into an average of 76 initial compound vertices (size 1 shown in the figures) representative of the original 14851 vertices. We limited the paths to this length due to time and memory constraints. When increasing the size of compound vertices (and thereby reducing the number of compound vertices), inevitably one compound vertex may be smaller than the others because the compound size does not divide evenly into the initial number of vertices, however this does not noticeably effect performance.

Figure 5 shows how varying the size of each compound vertex and the number of time steps N_s effects the expected reward relative to the initial (deterministic) path. With larger sizes of compound vertices, performance obviously degrades,

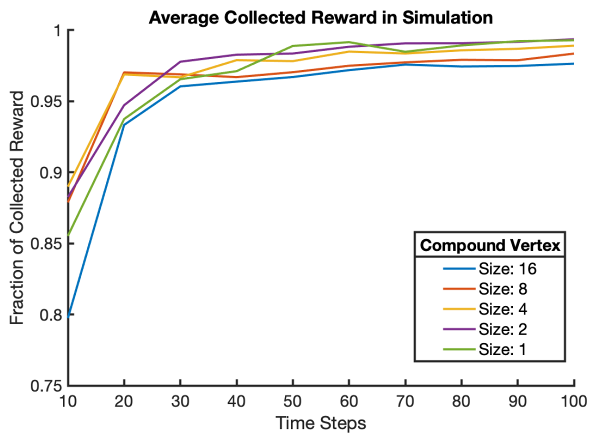

Fig. 5. A comparison of the average reward collected by compounding different numbers of vertices.

but not as significantly as one might think. The difference between expected rewards for size 16 and size 1 at $N_s = 100$ time steps is only 1.65% of the total reward on the deterministic path. This shows that the loss in performance when using more coarse compound vertices is not very significant. With small compound vertex sizes 1, 2, and 4, results are very comparable throughout the range of tested N_s. Interestingly, the most fine-grain tests with compound vertex sizes of 1 and 2 were not always the top performers. This is due to the fact that a larger N_s is needed for the CMDP to take advantage of the smaller granularity, which occurs in our tests at around $N_s = 30$. With smaller N_s values, the increased vertex resolution is actually a detriment, because a large portion of state transitions can occur where a new vertex is reached without crossing into new time interval. This can occur with any size compound vertices, and is related to the ratio of vertices vs the time steps used.

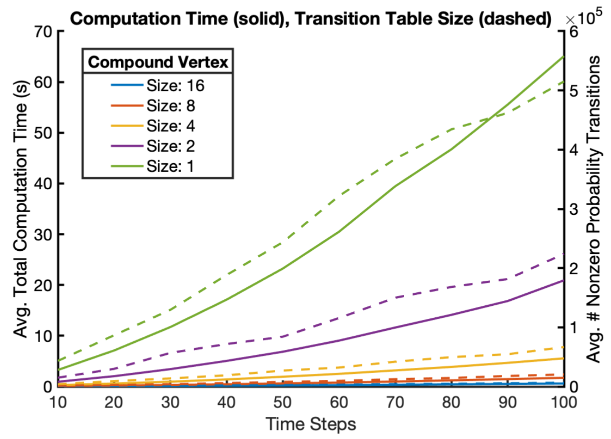

Fig. 6. Solid lines: a comparison of the average total computation time required to solve an instance of the SCOP on an IG. This includes time to build the state transition table in Matlab as well as solve the CMDP with CPLEX. These results were obtained using an Intel i7 6700k with 32gb ram. Dashed lines: the number of transitions where $\Pr((v_j, n), v_k, (v_k, l)) > 0$ in the state-action-state table of the CMDP.

Figure 6 shows how detrimental the size of compound vertices and the number of time steps are to the amount of time required to compute a solution for the SCOP. As the number of time steps increases, the computation time increases super-linearly. This is because more time steps means there is a larger number of possible state transitions with non-zero probability. But more significantly, larger compound vertices greatly reduce the computational burden compared to runs with non-aggregated vertices. As the number of vertices increases, there are more possible actions to take at each vertex, and the number of decision variables in the linear program increases, thus requiring more time to compute. Less compound vertices decreases this, and also makes for a smaller state-action-state table because of the reduced number of vertex transition pairs. A run with compound vertex size of 16 and $N_s = 100$ completes in 0.91% of the time needed to complete a run with compound vertex size of 1 and the same number of time steps. For the same number of time steps but a compound vertex size of 4, the percentage is 8.53%. Given that the computation time difference is so substantial compared to the reward difference, we feel this is a worthwhile trade-off when solving large problems.

VII. Conclusions

In this paper we studied a version of the OP where edge costs are stochastic and given as independent continuous random variables with positive support. The goal is to compute a policy with two objectives; maximize the expected reward collected and constrain the probability of exceeding a given budget. Our proposed solution works in three phases. First, we compute a path for the deterministic OP using the expected values of edge costs. Second, we aggregate a number of consecutive vertices into compound vertices that represent visiting all of its constituents, collecting reward and accruing cost along the way. This technique is useful for decreasing computation time on problems of large size where the initial path contains numerous vertices. Lastly, we formulate a CMDP that determines when to "take shortcuts" or to skip vertices in the path so that the probability of exceeding the budget does not exceed a given constraint. To test our solution method, we use a real world robotics problem where a robot with limited battery life must travel within a large vineyard to adjust irrigation emitters for optimizing water usage. Our tests indicate that we can efficiently compute policies maximizing expected reward and bounding failure probabilities on these robotic irrigation problems.

There are a number of potential avenues for future research with this approach. One such avenue is to thoroughly study how different parameters, such as the bound on failure probability and the variance of edge costs, effect the efficiency of this technique. Another is to consider adaptive replanning that is able to iteratively update how vertices are aggregated to compute policies which collect reward more efficiently. Another is to explore adjusting the time intervals for each vertex cluster to build more efficient policies. Finally, it would be useful to explore dynamically adjusting the initial path to visit different vertices when advantageous.

Acknowledgments

We gratefully acknowledge Luis Sanchez and Nick Dokoozlian from E&J Gallo Winery for having granted access to their vineyards for data collection.

References

[1] Eitan Altman. *Constrained Markov Decision Processes*. Stochastic modeling. Chapman & Hall/CRC, 1999.

[2] Ron Berenstein, Roy Fox, Stephen McKinley, Stefano Carpin, and Ken Goldberg. Robustly adjusting indoor drip irrigation emitters with the toyota hsr robot. In *IEEE International Conference on Robotics and Automation*, pages 2236–2243, 2018.

[3] Dimitri P. Bertsekas. *Dynamic Programming & Optimal Control*, volume 1 and 2. Athena Scientific, 2005.

[4] Dimitri P. Bertsekas. *Reinforcement Learning and Optimal Control*, chapter 6. Aggregation, pages 308–340. Athena Scientific, 2019.

[5] 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.

[6] 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.

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

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

[10] Seyedshams Feyzabadi and Stefano Carpin. Planning using hierarchical constrained markov decision processes. *Autonomous Robots*, 41:1589–1607, 2017.

[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*, pages 315–332, 2016.

[13] Anupam Gupta, Ravishankar Krishnaswamy, Viswanath Nagarajan, and R. Ravi. Approximation algorithms for stochastic orienteering. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1522–1538, 2012.

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

[15] Margaret A. Oliver and Richard Webster. Kriging: A method of interpolation for geographical information systems. *International Journal of Geographical Information Systems*, 4(3):313–332, 1990.

[16] Thomas C. Thayer, Stavros Vougioukas, Ken Goldberg, and Stefano Carpin. Multi-robot routing algorithms for robots operating in vineyards. In *IEEE International Conference on Automation Science and Engineering*, pages 14–21, 2018.

[17] 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.

[18] 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.

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

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

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