Track, stop, and eliminate: an algorithm to solve stochastic orienteering problems using MCTS

Carlos Diaz Alvarenga Stefano Carpin

***Abstract*— We present a novel algorithm to solve the stochastic orienteering problem with chance constraints that combines Monte Carlo Tree Search (MCTS) with a best arm identification (BAI) algorithm. This method extends our recently proposed solution that builds a search planning tree considering both an objective function to maximize, as well as a chance constraint on the failure probability, i.e., the probability of violating the assigned budget constraint. By combining these two approaches, we obtain a new planner that tunes the amount of tree search at run time. Extensive simulation results on our benchmark problems show that the new approach is significantly faster than the previous one, while incurring in just marginal decrements in terms of performance.**

I. Introduction

The orienteering problem is a combinatorial optimization problem whose input is a graph G with weights on edges and rewards on vertices, and a budget B. The objective is to find a path in the graph between preassigned start and end vertices that collects the maximum sum of rewards, while ensuring that the path length does not exceed the budget B. Orienteering offers a convenient abstraction for many robotic problems where a robot has to select a subset of locations to perform certain services. Examples include logistics [14], environmental monitoring [23], surveillance [8], and precision agriculture [19], just to name a few. Applications in precision agriculture motivate our ongoing research in this problem (see Figure 1).

In *stochastic* orienteering, the costs of the edges are random variables whose realizations are obtained at run time. This extension is suitable to model scenarios in which the time or energy spent to move between two locations can vary due to environmental conditions and is not known upfront. In this case the cost of the path is a random variable and the constraint on the budget can be expressed as a chance constraint, i.e., as a bound on the probability that the cost of the path exceeds the budget B (a formal problem statement will be given in section III).

In our recent work [20] we introduced a new algorithm using Monte Carlo Tree Search (MCTS) to solve the stochastic orienteering problem with chance constraints. MCTS algorithms are extensively used to solve planning problems using generative models, and use a receding horizon approach to explore a set of alternatives to identify the best next action. As common in these types of planning problems, the objective is to identify an action that accumulates high reward in an additive setting. However, in some problem instances each action is not only associated with a reward, but also with a failure probability. For example, in the stochastic orienteering problem the choice of a path is not only associated with the reward collected along the path, but also with the probability that the stochastic length of the path violates a preassigned budget (an event that we deem a *failure*). Starting from this observation, to use MCTS for the stochastic orienteering problem with chance constraints, in [20] we introduced a modified criterion for node selection that modifies the widely used Upper Confidence Bounds (UCB) heuristic by considering not only the accrued reward, but also the estimated failure probability. This method turns out to beat our previously proposed solutions, but is not immune from drawbacks. In particular, it is not adaptive. This means that, as is typical for MCTS, a certain amount of computational effort (e.g., number of different actions to try or rollouts to perform) is predetermined and executed at each stage. In some instances this number may be too high, for example when after few steps it is already evident which option is the best. But in some instances the opposite may

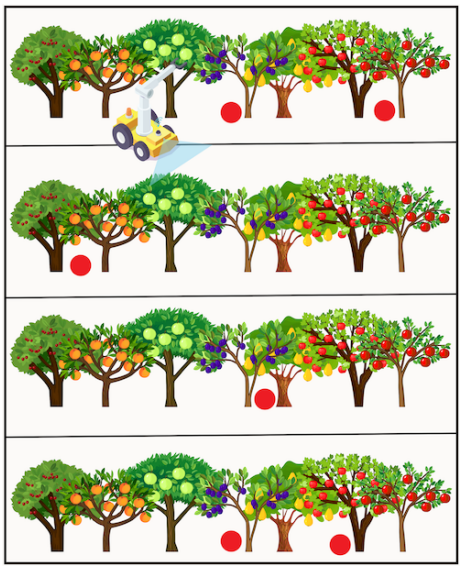

Fig. 1: In many precision agriculture applications, a robot is tasked with performing a set of tasks (with different rewards) situated at different locations (red dots). Due to limited onboard energy, the robot has to select which subset of tasks it can execute. Moreover, in unstructured conditions the energy needed to move between two locations can be subject to significant random variations.

The authors are with the Department of Computer Science and Engineering, of the University of California, Merced, USA.This work is partially supported by USDA-NIFA under award # 2021-67022-33452 and by the National Science Foundation (NSF) under Cooperative Agreement EEC-1941529. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the U.S. Department of Agriculture or the NSF.

be true as well, i.e., after the search budget is exhausted it may still be unclear which option is the best. In an effort to overcome this limit, in this paper we explore a new alternative, i.e., the use of pure exploration algorithms to solve the *best arm identification* problem. These algorithms are adaptive, i.e., keep exploring as long as needed, but stop as soon as the probability of having identified the best arm exceeds a given target threshold. There is a well known connection between MCTS and bandit problems, as the action selection in MCTS is equivalent to the arm selection in bandit problems. In a pure exploration bandit problem the objective is to identify the best arm without worrying about the regret accrued during exploration. Rather, the objective is to identify the best arm in the least number of attempts subject to a fixed confidence bound. Pure exploration algorithms generally consider bandits with a single reward, however as pointed out above, we are interested in situations where each action is associated with both a stochastic reward and a failure probability. Hence, in this paper we define a new model suitable to solve this type of problems and propose a new algorithm called *track, stop and eliminate* that extends the formerly proposed "track and stop" algorithm [5] introduced to solve pure exploration problems with a single objective function. Afterwards, we use it to derive a heuristic for the stochastic orienteering problem with chance constraints that overcomes our previous solutions.

The rest of this paper is organized as follows. Related literature is presented in Section II and relevant theoretical background is provided in Section III. Our algorithmic contributions are given in Section IV and validated in Section V, where we compare them with various alternatives, including an optimal planner using a mixed integer program formulation. Finally, in Section VI we conclude the paper and outline venues for future work.

II. Related work

The deterministic orienteering problem is known to be NP-hard and was first formalized in [6]. Exact solutions can be found formulating this problem as an integer program [4], and therefore heuristics are mostly used when solving large problem instances [7]. Approximation algorithms [3] have appeared in literature but their use is scarce. The stochastic orienteering problem has received less attention and was first introduced in [2], though this initial formulation did not consider chance constraints. As this problem is also NP-hard, most solutions for this problem rely on heuristics, too. In our previous works, we have studied the problem using a chance constrained formulation [16]–[18] and introduced the concept of *path policy*, i.e., a family of sub-paths was computed off-line, and the decision of which one to follow was taken online based on the actual incurred costs. In [22] the authors propose an algorithm to solve orienteering based on local search, while in [21] the authors propose a mixed integer program based on sample average approximation. These solutions are fundamentally different from the one we propose because they are formulated offline, and not updated as the mission unfolds based on the travel costs experienced during the mission. In [20] we proposed an alternative method to solve the stochastic orienteering problem with chance constraints that uses MCTS. MCTS includes a family of any-time methods to solve sequential stochastic decision problems using a generative model. These methods have been used for quite some time but have seen a resurgence in popularity due to their successful use in combination with reinforcement learning (see e.g., [12]).

There is a rich literature about bandit algorithms and the reader is referred to [11] for a comprehensive introduction to the problem. Bandit algorithms are connected to MCTS algorithms because at each level in the tree one is tasked with selecting the best action to execute, and this is equivalent to picking a bandit arm. In fact, the UCT criterion widely used in MCTS is derived by the UCB rule derived for bandits [10]. The pure exploration, or best arm identification (BAI) algorithms for multi-armed bandits (MAB) generally fall into two categories: fixed confidence and fixed budget settings. The focus of this work is the former case, fixed confidence settings, where the goal of the algorithm is to identify the optimal arm (highest mean reward) within some confidence threshold in the minimum number of pulls or samples. The assumption is that there is a single optimal arm to identify. In [5], the authors present both a lower bound on the number of pulls needed in the fixed confidence formulation and an algorithm that is (asymptotically) optimal called "Track and Stop". We decided to build our solution on top of the track and stop algorithm because of its optimality and guarantees, however, we consider its application to a constrained version of the MAB-BAI problem. In our formulation, the goal is not only to identify the best arm within a minimum number of pulls but to also to make sure that the optimal arm is feasible as well.

The application of these fixed confidence best arm identification algorithms to MCTS (MCTS-BAI) has been studied before in [15] and [9], among others. The potential is to have tools for analyzing the sample complexity of MCTS methods, and algorithms for MCTS that are adaptive with respect to the difficulty of determining high reward actions. [9] presents a general algorithmic framework for MCTS by best arm identification and analyzes some instances such as the Lower Upper Confidence Bounds-MCTS (LUCB-MCTS); furthermore, the authors conclude by presenting a lower bound on the samples needed for MCTS-BAI which requires the solution to a generally difficult optimization problem. The MCTS-BAI method presented in this work differs with respect to the previously mentioned works in many regards. As mentioned before, we consider a constrained version of the BAI problem and the integration of its solution into the MCTS framework. Moreover, we break down the so-called selection phase (tree policy) [1] into two distinct parts. At the root of the entire tree our proposed method uses the track and stop algorithm to decide which sub-tree to explore next, then it applies our UCTF [20] heuristic, discussed in depth later, when traversing from the root of the selected sub-tree to a leaf. Finally, it performs a rollout and updates the statistics in the tree.

III. Background and problem definition

A. Stochastic Orienteering with Chance Constraints

The stochastic orienteering problem with chance constraints (SOPCC) can be defined as follows. Let $G = (V, E)$ be a directed graph, and $s, g \in V$ be the start and goal vertices. Let $r : V \to \mathbb{R}^+$ be a positive reward function defined over the vertices. For each edge $e \in E$, let c_e be a random variable with non-negative support modeling the cost incurred when traversing edge e. Let $B > 0$ be a travel budget and $0 < P_f < 1$ be a given failure bound. For a path p in G from s to g, its reward $R(p)$ is the sum[1] of the rewards of the vertices appearing along p. Its cost $C(p)$ is a random variable given by the sum of the random variables associated with all edges appearing along p. Let Π be the set of all paths in G from s to g. The stochastic orienteering problem with chance constraints asks to solve the following optimization problem:

$$p^* = \arg\max_{p \in \Pi} R(p)$$
$$\text{s.t.}\, \Pr[C(p) > B] \leq P_f$$

As the deterministic orienteering problem is a special case of the stochastic orienteering problem, it follows that this problem is also NP-hard.

B. Best Arm Identification (BAI) with Failures

We consider the following bandit exploration problem (see [11], ch. 33 for more details about pure exploration algorithms for bandits). We have a bandit $\mathcal{A}$ with k arms. Each arm $a_i \in \mathcal{A}$ is associated with two distributions, i.e., when we *pull* arm a_i we obtain two random returns. The first return X comes from a distribution P_{a_i} with finite mean μ_i. The second return Y comes from a Binomial distribution with parameter b_i. As usual in bandit algorithms, we assume that for each arm the values μ_i and b_i are unknown. We consider two confidence levels δ and ε, as well as a probability threshold ζ. A policy $\pi = (\pi_t)_{t=1}^{\infty}$ selects the arms of $\mathcal{A}$ in sequence as well as a stopping time τ. Paralleling the notation in [11], let τ be a stopping time, and $\psi \in \{1, \ldots, k\}$ be the recommendation by policy π for the arm to pull at time $\tau + 1$. The rules of the game are the same, i.e., π (the learner) repeatedly pulls arms from $\mathcal{A}$ until it eventually chooses as stopping time τ to stop exploring, and then makes a recommendation ψ for the *best* arm identified. In the following definition, $\Delta_\psi(\nu)$ is the *immediate regret*, i.e., the difference between the expected return of the optimal arm and the expected return of arm ψ. Note that for a suboptimal arm the immediate regret is strictly positive.

Definition 1: A triple (π, τ, ψ) is sound at confidence levels δ, ε if for all bandits

$$\mathbb{P}_\pi(\tau < \infty \text{ and } \Delta_\psi(\nu) > 0) \leq \delta$$

and

$$\mathbb{P}_\pi(b_\psi > \zeta) \leq \varepsilon.$$

In plain words, we look for an algorithm π (policy) that through exploration determines a stopping time and then selects an arm that with high confidence is the best arm and has a parameter b_ψ not exceeding the threshold value ζ. In [11] the same problem is presented, but without considering the second return Y distributed according to a binomial distribution.

Remark: we introduce the problem BAI with failures to use it for action selection in MCTS to solve SOPCC, as we will describe shortly. Consider an instance of SOPCC and consider a partial path p_1 from s to an intermediate vertex $v \neq g$ (see Figure 2). This path has cost $C(p_1)$. Path p_1 can be extended into a full path from s to g by considering all subpaths starting at v and ending at g. Each of these paths p_j will have a reward and a random cost that may or may not exceed the residual budget $B - C(p_1)$. In this case we use BAI with constraints to identify with high confidence the path from v to s_g that has the highest reward and does not exceed the residual budget. In this case each path p_j from v to g can be seen as an arm whose reward X is $C(p_j)$ and whose variable Y is a Bernoulli with parameter $b_j = \Pr[C(p_j) > B - C(p_1)]$.

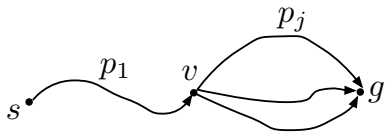

Fig. 2: A partial path p_1 from s to v can be extended into a full path from s to g considering multiple subpaths from v to g. Each will have a reward X and a failure probability Y with parameter $\Pr[C(p_j) > B - C(p_1)]$. Each such path can be seen as an arm in a BAI problem with failure probability.

IV. Track, Stop and Eliminate

A. Solving SOPCC with Constrained MCTS

We start sketching the basic algorithm we developed in [20] to solve the SOPCC problem, as this is the foundation for the algorithm we develop later on. The idea is to alternate planning and execution where at each iteration we build a MCTS rooted at the current vertex in the graph and parametrized by the remaining budget B. In the tree, each node is associated with a vertex in the graph, and each edge in the tree is associated with an action corresponding to moving between the two corresponding vertices in the graph. For each action a the MCTS produces two estimates $Q(a)$ and $F(a)$. $Q(a)$ is the estimated return of the path going from the root to the end vertex and using action a, and $F(a)$ is the probability that the path will violate the budget constraint (details on how these are computed are given below). After the tree is built, the action a to execute is the one at the root that has the highest $Q(a)$ value subject to the constraint that its $F(a)$ value does not exceed P_f. The action is then executed, and the budget is updated based on the time spent to complete the action. The process iterates

[1] If a vertex v appears multiple times along a path, its reward is collected only once.

until either the final vertex g is reached, or all budget is consumed. Algorithm 1 sketches this approach.

Algorithm 1 Solving SOPCC with MCTS

Data: Graph $G = (V, E)$, vertices $s, g \in V$, budget B
$v \leftarrow s$
while $B > 0$ **and** $v \neq g$ **do**
  $next_v \leftarrow \text{MCTS}(v, B)$
  move to vertex $next_v$ and let ξ_v be the incurred cost
  $B \leftarrow B - \xi_v$
  $v \leftarrow next_v$

The process to build the MCTS extends the classic MCTS approach in two ways. First, for each action a we include the probability $F(a)$ that a path using a will violate the constraint P_f. This is obtained by executing multiple rollouts at the leaves and using the sample average to estimate $F(a)$. Second, rather than using the well known UCB rule [10] as the tree policy to balance exploration and exploitation, we introduce a new criterion called UCTF that evaluates each action a using both $Q(a)$ and $F(a)$. The formula for UCTF is

$$UCTF(a) = Q(a)(1 - F(a)) + c\sqrt{\frac{\log t}{N(a)}}$$

By introducing the term $(1-F(a))$ the exploitation term puts less weight on actions that have a high failure probability, even if they are associated with high returns. The other quantities in the formula are the same as in UCB. Algorithm 2 sketches this approach.

Algorithm 2 MCTS

Data: start vertex v, budget B
Initialize tree $\mathcal{T}$ with root equal to v
for K *iterations* **do**
  $v_j \leftarrow UCTF(\mathcal{T})$
  Perform multiple rollouts at v_j and estimate Q and F
  Backup(Q, F)
return ActionSelection$(\mathcal{T})$

For lack of space we omit details like the heuristic used to execute the rollouts, and how the backup step is performed to propagate the values Q and F from the leaf v_j back to the root of the tree. The reader is referred to [20] for the specifics, and in the experimental section we use exactly the same implementation described therein. While this approach is competitive when compared with former solutions we proposed, we identify two aspects that can be improved. First, the number of iterations K is a preassigned parameter that requires manual tuning. It would be desirable to remove this dependency and let the algorithm determine on the fly when enough exploration happened. Second, as the number of paths generated increases, the estimates for F could be used to eliminate from the search space actions that are very likely to violate the failure probability P_f. This can be done noting that F is the experimental estimate of the success/failure probability of a Bernoulli random variable. These two aspects are tackled in the following subsections.

B. Solving BAI with constraints

In [5] the authors propose an algorithm for the best arm identification problem that stops the exploration step after a predetermined level of confidence δ is achieved. This is similar to the problem we introduced in section III but it considers a single reward. Algorithm 3 sketches their proposed solution and the reader is referred to [5] for more information and a thorough mathematical analysis.

Algorithm 3 Best Arm Identification (Track and Stop)

Data: confidence level δ, k arms $a_1, \ldots, a_k$
Try all k arms once and set $t \leftarrow k$
while $Z_t < \beta_t(\delta)$ **do**
  if $\arg\min_i T_i(t) \leq \sqrt{t}$ **then**
    Choose $A_{t+1} = \arg\min_i T_i(t)$
  else
    Choose $A_{t+1} = \arg\max_i (t\hat{\alpha}_i^*(t) - T_i(t))$
  Observe return X_{t+1}, update statistics, increase t
return $\tau = t$, $\psi = i^*$

After having tried all arms at least once, the algorithm enters a loop where it considers $T_i(t)$, the number of times that the i-th arm has been tried up to time t. If there is at least one arm tried less than $\sqrt{t}$ times, it selects that arm (with ties randomly broken), otherwise it picks an arm solving an optimization problem whose solution depends on the number of times each arm has been tried so far, as well as the cumulative reward obtained up to that point (quantities called *statistics* in the paper). Critical to this approach is the test governing the main loop. Given the statistics and an assigned confidence level δ, the authors identify two functions Z_t (a function of the statistics) and $\beta_t(\delta)$ such that as soon as Z_t exceeds $\beta_t(\delta)$ one can conclude that with probability at least δ the arm with the highest cumulative reward is the best one. The index of this arm is then returned at the end. The theoretical guarantees studied in the paper hold under the assumption that the returns of each arm a_i are sub-Gaussian random variables. The authors name the algorithm track and stop and prove that it is asymptotically optimal with respect to the lower bound on the number of pulls needed to provide statistical certainty.

To solve the Best Arm Identification with Failures problem, we extend algorithm 3 by tracking the statistics not only for the return X but also for the Bernoulli variable Y. In particular, for each variable Y_i we estimate its parameter b_i using the sample mean. Moreover, using the the Wald test [13], one can estimate lower and upper bounds for the confidence interval for b_i with the formulas:

$$L = b_i - z_{\varepsilon/2}\sqrt{\frac{b_i(1-b_i)}{T_i}} \quad U = b_i + z_{\varepsilon/2}\sqrt{\frac{b_i(1-b_i)}{T_i}}$$

where T_i is the number of times arm a_i has been tried and $z_{\varepsilon/2}$ is the $1-\varepsilon/2$ quantile of a standard normal distribution.

In algorithm 4 we start as in algorithm 3, but after the best arm with respect to the return X has been identified and we exit the inner while loop, the Wald criterion is used to test the

Algorithm 4 Best Arm Identification with Failures

Data: confidence levels δ and ε, k arms $a_1, \ldots, a_k$
Try all k arms once and set $t \leftarrow k$
$done \leftarrow$ **false**
while not *done* **do**
  while $Z_t < \beta_t(\delta)$ **do**
    if $\arg\min_i T_i(t) \leq \sqrt{t}$ **then**
      Choose $A_{t+1} = \arg\min_i T_i(t)$
    else
      Choose $A_{t+1} = \arg\max_i (t\hat{\alpha}_i^*(t) - T_i(t))$
    Observe return X_{t+1}, update statistics, increase t
  Compute L, U for best arm i^*
  if $U \leq P_f$ **then**
    $done \leftarrow$ **true**
  else if $L > P_f$ **then**
    eliminate arm i^*
  else
    keep pulling arm i^* until $U < P_f$ or $L > P_f$
return $\tau = t$, $\psi = i^*$

confidence interval for the estimate for b_{i^*}. If $U \leq P_f$, then the best arm satisfies the criterion and is returned. If instead $L > P_f$, then the arm is eliminated because it violates the failure constraint. In such case, the algorithm goes back to the inner loop and keeps exploring. If neither case holds, the arm is repeatedly pulled until one of the two cases apply and we behave accordingly, either terminating or eliminating the arm (this is guaranteed to happen as T_i grows).

C. Solving SOPCC with MCTS and Constrained BAI

Starting from the previous algorithms we can now combine algorithms 2 and 4 to expedite the solution of SOPCC. The idea is to use the algorithm to solve the constrained BAI problem to decide when to stop the exploration while building the MCTS. More specifically, algorithm 4 is used at the root of the tree to identify the arms to explore and when to stop. When an arm at the root is selected for exploration, we use UCTF as the tree policy to traverse from that arm to a leaf. This way, we combine the advantages of both approaches. The pseudocode is given in algorithm 5.

We see that BAI with failure is used at the root to select one arm, i.e., the first edge on the path from the current vertex v (root of the node) to the goal vertex g. When the arm (i.e., edge towards another node) is selected, we then use UCTF to expand the tree, as in the original MCTS algorithm updating Q and F for all involved edges/actions. When the inner loop exits, the confidence levels L, U are computed using $F(i^*)$, i.e., the estimated failure probability of the tentative best arm. At that point we follow the same strategy as before, i.e., we may terminate, eliminate the arm, or pull it again until needed.

The practical implementation of this algorithm may have one limitation. In the case of a graph with symmetries or similar rewards, there may be multiple arms at the root that have exactly the same expected reward (or very similar). In this case, there is no best arm (if they have the same expected reward) or it may take an extremely large number of

Algorithm 5 MCTS with Constrained BAI (MCTS-BAI)

Data: start vertex v, budget B,confidence levels δ and ε
Initialize tree $\mathcal{T}$ with root equal to v
Try all k actions as the root once using UCTF to determine Q and$F$
$t \leftarrow k$
$done \leftarrow$ **false**
while not *done* **do**
  while $Z_t < \beta_t(\delta)$ **do**
    if $\arg\min_i T_i(t) \leq \sqrt{t}$ **then**
      Choose $A_{t+1} = \arg\min_i T_i(t)$
    else
      Choose $A_{t+1} = \arg\max_i (t\hat{\alpha}_i^*(t) - T_i(t))$
    Use UCTF(A_{t+1}) expand the tree, updating Q and F for all involved actions, and performing Backups
    Update statistics, increase t
  Compute L, U for best arm i^*
  if $U \leq P_f$ **then**
    $done \leftarrow$ **true**
  else if $L > P_f$ **then**
    eliminate arm i^*
  else
    keep pulling arm i^* until $U < P_f$ or $L > P_f$
return $\tau = t$, $\psi = i^*$

attempts to disambiguate (if they have similar rewards). We actually observed this in some of the benchmark problems we studied in Section V. If this is the case, this problem can be mitigated by clustering the arms in three groups, based on the cumulative reward accrued during the exploration. Then the criterion $Z_t < \beta(\delta)$ used to terminate the exploration is computed using representatives from each cluster (i.e., we reduce the problem to the case of a bandit with three arms). The number three is of course arbitrary and could be refined based on the specific application.

Remark: one could observe that the theoretical results supporting algorithm 3 hold under precise assumptions for the returns, i.e., that the random variables are sub-Gaussian, but that in general there is no guarantee that the returns in the MCTS algorithm satisfy this hypothesis. This is correct. However, it should be pointed out that even when using UCB for tree exploration one is de-facto using an heuristic that is guaranteed to provide optimal results only under specific hypothesis that are most often not verified. In fact, many successful strategies for MCTS implementation rely on insightful heuristics that cannot be theoretically characterized. Our work falls into this category.

V. Results

In this section we start comparing the performance of our proposed new algorithm with the original MCTS version we proposed in [20]. Note that therein we already showed that the MCTS approach outperformed our former solutions that produced path policies using Constrained Markov Decision Processes. To ensure a fair evaluation, we use the same set of benchmark problems. These consists of four different graphs whose number of vertices vary between 10 and 40 (with

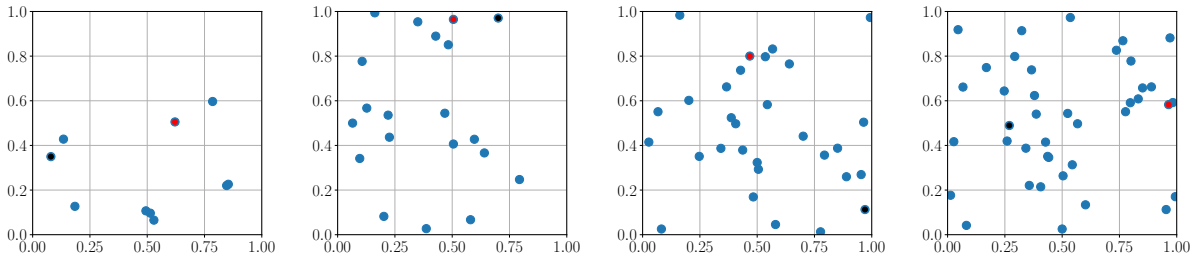

Fig. 3: The four test graphs used for benchmarking. In all graphs the start vertex is marked in red and the end vertex is marked in black.

increments of 10) and we consider two different budgets B, namely 2 and 3. In all instances, the locations of the points were randomly selected in the unit square, and the reward of each vertex was generated sampling from a uniform distribution in $[0, 1]$. The graphs are shown in figure 3.

Each edge (v_i, v_j) between vertices v_i and v_j has a random travel cost given by the formula

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

where $d_{i,j}$ is the Euclidean distance between v_i and v_j and $\mathcal{E}(\lambda)$ is random sample from the exponential distribution with parameter λ. With this choice we ensure that the travel cost is a random variable with expectation $d_{i,j}$. As in our previous works, throughout the experiments we set $\alpha = 0.5$. All graphs are complete, i.e., they include all possible edges. This setup is the most challenging for the MCTS and MCTS-BAI algorithms, because every node in the tree has highest possible branching factor and and for n vertices there are $\mathcal{O}(n!)$ possible paths. In all experiments we set the confidence levels δ and ε to 0.1. All algorithms were coded in Python and executed on an Apple M1 Max with 32 GB of memory. For the implementation of the BAI algorithm in Algorithm 3, we relied on a publicly available implementation provided by the authors.[2]

We start comparing the performance of the algorithm we presented in this paper with the MCTS implementation we discussed in [20]. Specifically, we compare three quantities, i.e., the collected reward, the the time spent for planning, and the number of rollouts executed. All instances are obtained averaging the results of 100 independent runs for all cases. Figure 4 shows the ratio between the average reward collected by the MCTS algorithm (Algorithm 2) and the MCTS-BAI algorithm (Algorithm 5). As the latter is designed to use less resources, it is expected that it will perform worse, i.e., that it will collect less reward. However, the figure shows that the decrement in performance is modest, and in most cases is below 10%.

Figures 5 and 6 contrast the two algorithms with respect to the number of rollouts performed and the time spent for planning. As we can see, the MCTS-BAI algorithm consistently performs less rollouts and spends less time, and the gap between the two widens as the number of

[2]The implementation is coded in Julia and we translated it in Python.

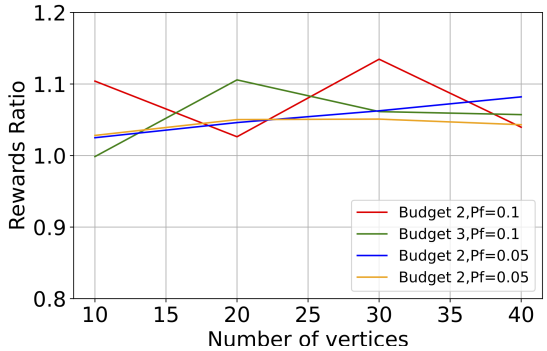

Fig. 4: Rewards ratio between the MCTS and the MCTS-BAI algorithms for different graph sizes (averages over 100 independent runs) and different budgets and failure probabilities.

vertices increases. This outlines the ability of MCTS-BAI to scale to larger problem instances. Overall, MCTS-BAI collects rewards similar to MCTS, while being much faster by performing far less rollouts.

Finally, for sake of completeness, we compare our solution with the approach presented in [21] which uses a mixed integer linear program (MILP) formulation to optimally solve the problem. One drawback of the approach presented in [21] is that it requires to manually tune various parameters. In particular, to meet the constraint that $\Pr[C(p) > B] < P_f$ it is necessary to carefully pick a number of samples that will influence the number of integer variables in the problem. We implemented the algorithm in [21] using the Python library PULP to generate the integer programs, and then used the commercial solver Gurobi to solve them. In all cases, if the solver had not found the optimal solution after 10 minutes, it stopped and produced the best solution found up to that point. Figure 7 parallels figure 4 and displays the ratio between the quality found by the MILP algorithm and the MCTS-BAI algorithm. As expected, the ratio is higher, but it is interesting to observe that it never exceeds 1.35. Figure 8 instead compares the time spent by the two algorithms. For small problem instances ($n = 10$) the time spent is comparable, but for larger sizes the time spent becomes much larger. The only exception is for $P_f = 0.05$ and $B = 2$ (where nevertheless MCTS-BAI is faster). This is explained by the fact that with a small budget and a small failure probability the number of paths satisfying the constraints

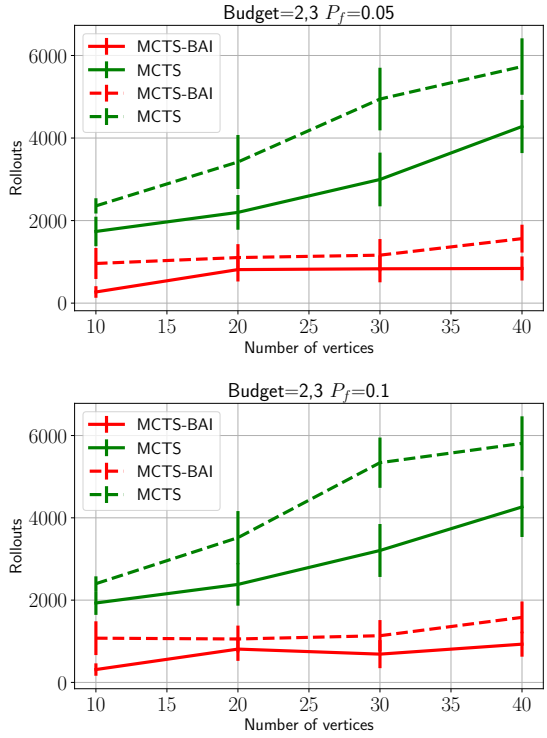

Fig. 5: Average number of rollouts performed by the original MCTS algorithm and the MCTS-BAI algorithm proposed in this paper as a function of the number of vertices when solving SOPCC. Solid lines correspond to the case with budget 2, while dashed lines are for the case with budget 3. The top figure is for failure probability $P_f = 0.05$, while the bottom figure is for failure probability $P_f = 0.1$.

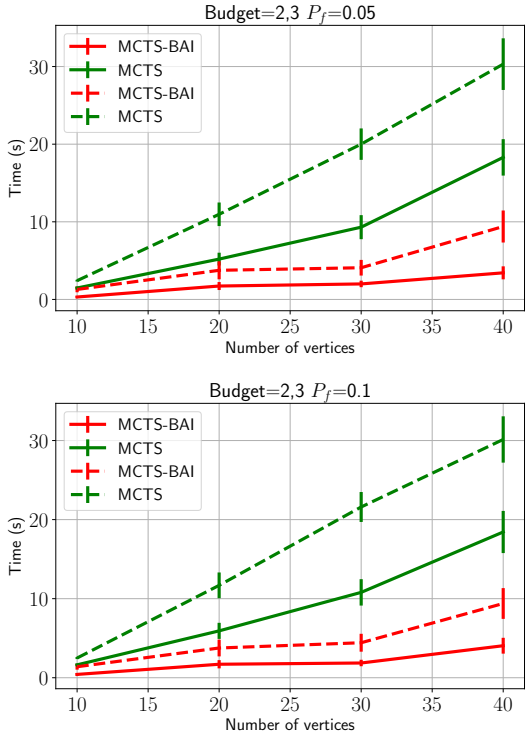

Fig. 6: Average time spent for planning by the original MCTS algorithm and the MCTS-BAI algorithm proposed in this paper as a function of the number of vertices when solving SOPCC. Solid lines correspond to the case with budget 2, while dashed lines are for the case with budget 3. The top figure is for failure probability $P_f = 0.05$, while the bottom figure is for failure probability $P_f = 0.1$.

is smaller, and therefore the MILP algorithm manages to efficiently prune the search space and reduce its computation time. In all other cases the gap is large.

VI. Conclusions

In this paper we presented a novel algorithm to solve the stochastic orienteering problem with chance constraints. The idea is to combine our recently introduced MCTS algorithm for this problem with the solution of the best arm identification with constraints problem. The combination of these algorithms gives a new method that is much faster than the previous one, while incurring in a marginal decrease in the performance. Interestingly, when compared with the optimal solution, we see that for hard problem instances the reward gap remains bounded, while our algorithm is orders of magnitude faster. In the future we will explore how these algorithmic ideas can be used to solve other constrained planning problems.

References

[1] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

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

[3] C. Chekuri, N. Korula, and M. Pál. Improved algorithms for orienteering and related problems. *ACM Transactions on Algorithms*, 8(3), July 2012.

[4] M. Fischetti, J. J. S. Gonzalez, and P. Toth. Solving the orienteering problem through branch-and-cut. *INFORMS Journal on Computing*, 10(2):133–148, 1998.

[5] A. Garivier and E. Kaufmann. Optimal best arm identification with fixed confidence. In *Proceedings of the 29th Conference on Learning Theory*, page 998–1027, 2016.

[6] B. L. Golden, L. Levy, and R. Vohra. The orienteering problem. *Naval Research Logistics*, 34:307–318, 1987.

[7] A. Gunawan, H. C. Lau, and P. Vansteenwegen. Orienteering problem: A survey of recent variants, solution approaches, and applications. *European Journal of Operational Research*, 255(2):315–332, 2016.

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

[9] E. Kaufmann and W. M. Koolen. Monte-carlo tree search by best arm identification. *Advances in Neural Information Processing Systems*, 30, 2017.

[10] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, pages 282–293, 2006.

[11] T. Lattimore and C. Szepesvári. *Bandit Algorithms*. Cambridge Press, 2020.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[13] A. Papoulis and S.U. Pillai. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill, 4th edition, 2002.

[14] F. Betti Sorbelli, S. Carpin, F. Coró, S.K. Das, A. Navarra, and C.M.

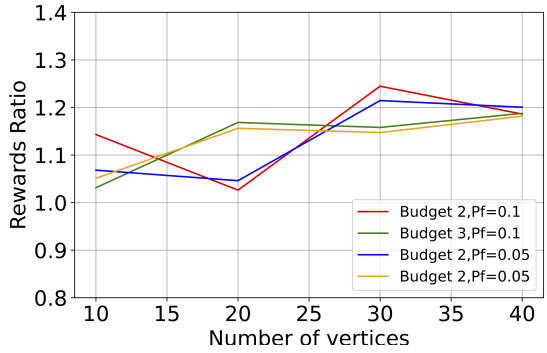

Fig. 7: Rewards ratio between the optimal algorithm and the MCTS-BAI algorithms for different graph sizes.

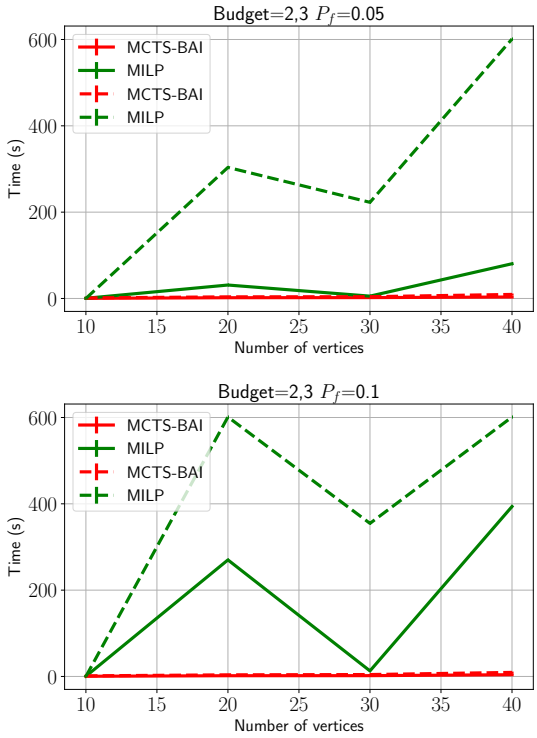

Fig. 8: Average time spent for planning by the original optimal algorithm and the MCTS-BAI algorithm proposed in this paper as a function of the number of vertices. Solid lines correspond to the case with budget 2, while dashed lines are for the case with budget 3. The top figure is for failure probability $P_f = 0.05$, while the bottom figure is for failure probability $P_f = 0.1$.

Pinotti. Speeding up routing schedules on aisle-graphs with single access. *IEEE Transactions on Robotics*, 38(1):433–447, 2022.

[15] K. Teraoka, K. Hatano, and E. Takimoto. Efficient sampling method for monte carlo tree search problem. *IEICE TRANSACTIONS on Information and Systems*, 97(3):392–398, 2014.

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

[17] T. Thayer and S. Carpin. A fast algorithm for stochastic orienteering with chance constraints. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 7398–7945, 2021.

[18] T. Thayer and S. Carpin. A resolution adaptive algorithm for the stochastic orienteering problem with chance constraints. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 6388–6395, 2021.

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

[20] T.C. Thayer and S. Carpin. Solving stochastic orienteering problems with chance constraints using monte carlo tree search. In *Proceedings of the IEEE International Conference on Automation Science and Engineering*, pages 1170–1177, 2022.

[21] P. Varakantham and A. Kumar. Optimization approaches for solving chance constrained stochastic orienteering problems. In *Algorithmic Decision Theory: Third International Conference, ADT 2013, Bruxelles, Belgium, November 12-14, 2013, Proceedings*, pages 387–398. Springer-Verlag, 2013.

[22] P. Varakantham, A. Kumar, H. C. Lau, and W. Yeoh. Risk-sensitive stochastic orienteering problems for trip optimization in urban environments. *Transactions on Intelligent Systems and Technology*, 9(3), 2018.

[23] J. Yu, M. Schwager, and D. Rus. Correlated orienteering problem and its application to persistent monitoring tasks. *IEEE Transactions on Robotics*, 32(5):1106–1118, 2016.