2017 American Control Conference
Sheraton Seattle Hotel
May 24–26, 2017, Seattle, USA

Informative Path Planning under Temporal Logic Constraints with Performance Guarantees

Kevin J. Leahy[1], Derya Aksaray[2], and Calin Belta[1]

Abstract— **In this work we consider an agent trying to maximize a submodular reward function while moving in a graph environment. Such reward functions can be used to capture a variety of crucial sensing objectives in robotics including, but not limited to, mutual information and entropy. Furthermore, the agent must satisfy a mission specified by temporal logic constraints, which can encode many rich and complex missions such as "visit regions A or B, then visit C, infinitely often. Never visit D before visiting C." We present an algorithm to maximize a submodular reward function under these constraints and provide an approximation for the performance of the proposed algorithm. The results are validated via simulation.**

I. Introduction

In a surveillance mission an agent visits a set of locations to increase situational awareness. A major goal in such missions is to plan the optimal path of the agent with respect to a performance metric. Mutual information or entropy are some commonly used metrics in surveillance scenarios (e.g., [1], [2], [3], [4], [5], [6]), and they exhibit a property called *submodularity*. Intuitively, submodular functions exhibit the property of diminishing returns (i.e., adding a new observation increases the function's value more if a few observations are made so far than if many observations have already been made). Greedy algorithms have proven performance bounds for the maximization of submodular functions [7], making such algorithms an attractive and convenient tool when sensing goals are expressed as submodular functions.

Recently, there has been an increased interest in developing efficient algorithms optimizing submodular objective functions (e.g., [8], [9], [10]). The authors in [11] addressed the problem of finding an optimal path for a single agent maximizing a submodular function (over the visited regions) and developed a *recursive-greedy algorithm* with theoretical approximation guarantees. This *recursive-greedy* approach was extended to multi-agent scenarios with resource constraints [10] and single-agent scenarios with multiple tours [9]. Moreover, a recent study [12] proposed a cost-benefit algorithm optimizing submodular functions while considering other costs (e.g., coverage cost, visit cost).

In some surveillance missions, while an agent is optimizing a performance measure, it may also be subject to trajectory constraints such as avoiding some regions or visiting regions in a specific order. As the complexity of these constraints increases, it is hard to formulate them in a classical optimization setup. *Temporal logics* (TL), which are rich and expressive specification languages, can capture such complex constraints. For example, Linear Temporal Logic (LTL) can express a persistent surveillance task as follows: "visit regions A or B, then C, infinitely often. Never visit D before visiting C." Motion planning subject to TL formulas has been extensively studied in the literature (e.g., [13], [14], [15], [16], [17], [18]). Typically, model checking algorithms [19] are used to find a high-level plan, which is then implemented by a low-level controller.

Recent work using TL planning for information gathering includes [20], [21], [22]. These works in general do not produce an optimal solution due to the limited lookahead horizons. Specifically, they use closed-loop planning in the belief space online, but provide no performance guarantees. In this work we compute the entire path offline and provide an approximate solution with performance guarantees. Moreover, task planning and sequencing under some set, counting, and ordering constraints was discussed in [23], which differs from our work by considering a brute-force approach to find a path satisfying the constraints.

The problem we consider is to satisfy a surveillance mission in a discretized environment while maximizing a submodular set function. The mission is specified using TL constraints, and we seek a principled manner of searching over only those paths which satisfy the specification. By optimizing submodular functions, our solution is applicable to a broad class of sensing and information gathering scenarios, and by considering TL constraints, the proposed method can deal with numerous rich and complex mission specifications. This method mitigates the complexity of such problems and has a guaranteed optimality bound on the performance. To the best of our knowledge, the work presented in this paper is the first effort of optimizing submodular functions subject to TL constraints with theoretical bounds.

II. Preliminaries

A. Notation

For two sets A and B, we denote their Cartesian product as $A \times B$. Let $A^n = A \times A \times \ldots \times A$, where the Cartesian product $\times$ is taken n times. The cardinality of a set A is written $|A|$. For $v \in A$ and $t \in \mathbb{N}$, $v^{0:t}$ is a sequence of length $t+1$ from A^{t+1}. For two sequences $\chi_1 \in A^{t_1}$ and $\chi_2 \in A^{t_2}$, we denote their concatenation as $\chi_1 \cup \chi_2$. We denote the concatenation of $k > 2$ such sequences as $\chi_{1:k}$.

[1]Department of Mechanical Engineering, Boston University, Boston, MA 02215 USA (e-mail: {kjleahy,cbelta}@bu.edu).

[2]Computer Science and Artificial Intelligence Laboratory at Massachusetts Institute of Technology, Cambridge MA 02139 USA (e-mail: {daksaray}@mit.edu).

This work was partially supported at Boston University by the NSF under grants CMMI-1400167 and NRI-1426907, and by the ONR under grant MURI N00014-09-1051.

We denote a directed graph $\mathcal{G} = (V, E)$, which consists of a set of nodes V and a set of directed edges $E \subseteq V \times V$. For two nodes $v_1, v_2 \in V$, let $d(v_1, v_2)$ be the graph distance between those nodes, which is defined as the shortest path between them.

B. Temporal logic (TL) constraints

We consider the agent's mission specified as a fragment of syntactically co-safe LTL (scLTL) specification ϕ. Given a set of atomic propositions AP, scLTL formulas are inductively defined as [24]:

$$\phi = p|\neg\phi|\phi_1 \wedge \phi_2|\phi_1 \mathcal{U} \phi_2|\bigcirc \phi_1|\Diamond\phi_1 \,, \tag{1}$$

where $p \in AP$ is an atomic proposition, and ϕ, ϕ_1, and ϕ_2 are scLTL formulas. The boolean operators $\neg$ and $\wedge$ are negation and conjunction, respectively. The temporal operators $\mathcal{U}$, $\bigcirc$, and $\Diamond$ denote until, next, and eventually, respectively. The interested reader is referred to [24] for a full description of the syntax and semantics of this logic. Given an scLTL formula ϕ, we can convert it to a finite state automaton (FSA) using off-the-shelf tools [25].

Definition II.1 (Finite State Automaton)**.** A *finite state automaton* is a tuple $\mathcal{A} = (Q, q_0, \Sigma, \Delta_A, F_\mathcal{A})$, where

- Q is a finite set of states;
- $q_0 \in Q$ is an initial state;
- Σ is a finite input alphabet;
- $\Delta_A : Q \times \Sigma \to Q$ is a deterministic transition function;
- $F_\mathcal{A} \subseteq Q$ is a set of accepting states.

A *run* of an FSA for an input word $\sigma_0, \sigma_1, \ldots \sigma_{n-1} \in \Sigma^n$ is a sequence of states $q_0, q_1, \ldots q_n \in Q^{n+1}$, where $q_{i+1} = \Delta_A(q_i, \sigma_i)$ for all $i < n$ in the run. An FSA accepts an input word of length n if and only if the final state of the run is in its set of accepting states, i.e., $q_n \in F_\mathcal{A}$.

C. Submodular functions

In this work we consider a global reward function $f(\cdot) : 2^V \to \mathbb{R}$ that an agent is trying to optimize with respect to a set V, such as mutual information, entropy, or duration of time since last visit [26]. Such reward functions exhibit a property called *submodularity*, i.e., $\forall A \subseteq B \subseteq V$ and $v \in V \setminus B$:

$$f(A \cup \{v\}) - f(A) \geq f(B \cup \{v\}) - f(B) \,. \tag{2}$$

This property can be understood intuitively as a type of diminishing returns. More importantly, these functions have provable optimality bounds for greedy approaches [8]. To evaluate the incremental benefit of adding some element v to a set $\mathcal{R}$, we define the *residual reward function*, $f_\mathcal{R}(\cdot)$, where

$$f_\mathcal{R}(v) = f(\mathcal{R} \cup \{v\}) - f(\mathcal{R}) \,. \tag{3}$$

III. Problem Formulation

A. Agent motion model

For our problem, we consider an environment modeled as a graph $\mathcal{G} = (V, E)$, with V representing a set of regions of interest and E representing the feasible travel between those regions. The agent's objective is to maximize a submodular reward function while moving through its environment, without violating its TL constraints.

Definition III.1 (Deterministic Transition System)**.** A *deterministic transition system* (TS) is a tuple $\mathcal{T} = (V, v_0, Act, \Delta)$, where

- V is a finite set of states from $\mathcal{G}$;
- $v_0 \in V$ is the initial state;
- Act is a finite set of actions;
- $\Delta : V \times Act \to V$ is a deterministic transition function.

The motion of the agent is modeled using a deterministic transition system whose states are the nodes in the environment graph. At each time step, the agent selects an action and moves in the environment. For simplicity, we assume that each action takes one time step to complete. The methods presented in this paper also extend to the case in which the transition system is weighted, with the weights representing the relative travel time or cost for the different transitions. Such weights are omitted to keep the exposition and notation as simple as possible. Thus, the agent's movement through the environment via a set of actions $a_1, a_2 \ldots a_n \in Act^n$ corresponds to a sequence of states $v_0, v_1, v_2, \ldots v_n \in V^{n+1}$.

B. Problem statement

For a specificaton ϕ over V, we assume a deadline on satisfaction $\mathcal{B}$, which corresponds to an energy budget for the agent's motion. That is, the agent may take $\mathcal{B}$ transitions to satisfy ϕ[1]. The specification is given over the set of nodes in the environment V, including the base $v_b \in V$, at which the agent starts and finishes each tour of the environment. Thus, the specification enforces an ordering over a subset of nodes in V, while the agent is permitted to visit the other nodes regardless of order.

Problem III.1. Given an environment graph $G = (V, E)$, an agent operating in G and modeled as a transition system $\mathcal{T} = (V, v_0, Act, \Delta)$, a specification ϕ over V, a budget $\mathcal{B}$ on satisfaction of ϕ, and a submodular reward function $f : 2^V \to \mathbb{R}$, find a sequence of agent states $v^{0:\mathcal{B}}$ that solves the following optimization problem:

$$\max_{v^{0:\mathcal{B}}} f\left(v^{0:\mathcal{B}}\right) \tag{4}$$

and satisfies specification ϕ.

Example 1. Consider the environment in Fig. 1, which is abstracted as a graph. An agent begins at a base located at

[1]For a weighted transition system, this requirement corresponds to the sum of the weights of a path being less than or equal to $\mathcal{B}$.

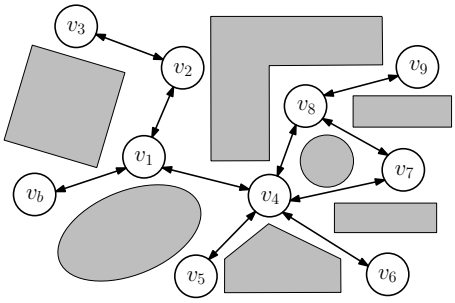

Fig. 1: Example environment abstracted as a graph. Obstacles are indicated in gray.

v_b and must visit nodes v_3 and v_4 before returning to the base. Then its specification ϕ is written

$$\Diamond v_3 \wedge \Diamond v_4 \wedge \Diamond v_b \wedge (v_b \implies v_b \mathcal{U} (\neg v_b \mathcal{U} v_3)) \wedge (v_b \implies v_b \mathcal{U} (\neg v_b \mathcal{U} v_4)) \ . \tag{5}$$

If its budget $\mathcal{B}$ were 8 time units, the agent can take 8 transitions while trying to ensure that ϕ is satisfied and $f\left(v^{0:\mathcal{B}}\right)$ is maximized. One $f(\cdot)$ of interest could be time since last visit, which is a modular function[2]. The time since the agent's last visit to node i as of time t is written $\alpha_i(t)$, and it evolves according to

$$\alpha_i(t) = \begin{cases} 0 & \text{if the agent visits node } v_i \text{ at time } t, \\ \alpha_i(t-1) + 1 & \text{otherwise.} \end{cases} \tag{6}$$

The objective function then becomes

$$f\left(v^{0:\mathcal{B}}\right) = -\sum_{t=1}^{\mathcal{B}} \sum_{i=1}^{|V|} \alpha_i(t) \ , \tag{7}$$

a modular function that we wish to maximize. The residual reward with respect to no surveillance is

$$f_{\mathcal{R}}\left(v^{0:\mathcal{B}}\right) = f\left(\emptyset \cup v^{0:\mathcal{B}}\right) - f(\emptyset) \ , \tag{8}$$

where $f(\cdot)$ is given by (7). It is (8) that our algorithm will use in its maximization. This function quantifies the improvement in (7) that a path yields in comparison to no surveillance.

IV. Solution

A. Solution overview

The solution we propose is inspired by the algorithm in Meliou et al. [9], which builds on work by Chekuri and Pal [11]. In general, planning under TL constraints uses graph algorithms on the product of the automaton $\mathcal{A}$ and the transition system $\mathcal{T}$, which has $|V||Q|$ nodes. Rather than planning over a product of $\mathcal{A}$ and $\mathcal{T}$, we plan over a subset of $\mathcal{T}$ by looking at the appropriate edges in $\mathcal{A}$, reducing the complexity of our proposed solution. In [9], the authors presented a nonmyopic solution to a persistent mission on an environment graph. They searched for a solution over multiple tours on an environment graph, thereby creating a larger graph out of multiple copies of the environment graph. We use a similar approach, but rather than copying the environment graph, we examine reachable states in $\mathcal{T}$ in a product automaton that corresponds to the same state in the FSA $\mathcal{A}$. Thus, rather than searching over multiple tours, we search over the edges of $\mathcal{A}$.

Assumption 1. We assume the following about the FSA $\mathcal{A}$:

- $\mathcal{A}$ is deterministic;
- $\mathcal{A}$ is acyclic (except for self-loops for individual states);
- transitions between states in $\mathcal{A}$ have only one label from Σ.

Assumption 1 implies the following: first, we consider a deterministic FSA for which each transition is uniquely defined by its source state and input so the expressed specification does not contain any uncertainty. For example, a surveillance mission such as "eventually visit region a and eventually visit region b" can actually be expressed by a deterministic FSA as in Fig. 2(a). If the specification results in a nondeterministic FSA, well-known algorithms can be used to make them deterministic, at the cost of some computational complexity [28].[3]Second, we consider an FSA that does not contain any cycles other than the self-loops for individual states. The proposed algorithm can in principle handle such loops, but we avoid them here to simplify the complexity analysis.[4] Finally, we assume that the transitions between the states in $\mathcal{A}$ have only one label because our TL formula is given over the nodes in the environment graph and the edges in $\mathcal{A}$ are labeled with *precisely one node* in the environment graph. In other words, the transitions in $\mathcal{A}$ are uniquely defined by a node in $\mathcal{T}$ (see Sec. III-A for more details).

Overall, the proposed method, which is planning over the edges of $\mathcal{A}$, becomes possible because of Assumption 1 because it allows us to enforce forward progress in $\mathcal{A}$. In particular, this assumption allows us to consider a subgraph of $\mathcal{T}$ located at each node in $\mathcal{A}$, with transitions to other nodes in $\mathcal{A}$ occurring for visiting specific nodes in $\mathcal{T}$. Our planning takes place in these subgraphs of $\mathcal{T}$, using the algorithm from [11]. Our solution also exploits the fact that greedy approximations have provable bounds to allow us to perform a forward search type of dynamic program along the edges of $\mathcal{A}$. This approach allows us to calculate a theoretical bound for our solution (Sec. IV-F). We provide a simple example below to illustrate the main idea of our algorithm.

Example 2. Consider an agent operating on the graph environment in Fig. 2(b). Let the agent's specification be

$$\Diamond a \wedge \Diamond b \, , \tag{9}$$

[2]This function can be formulated as a linear function over a set of states (see [13]), which is a type of modular function [27].

[3]Note, this discussion of determinism pertains to the *specification*, not the robot motion. Addressing stochasticity in robot motion is outside the scope of this work.

[4]In the worst case, consider such loops should increase the complexity by a linear factor of $\mathcal{B}$, the maximum number of possible transitions in $\mathcal{A}$.

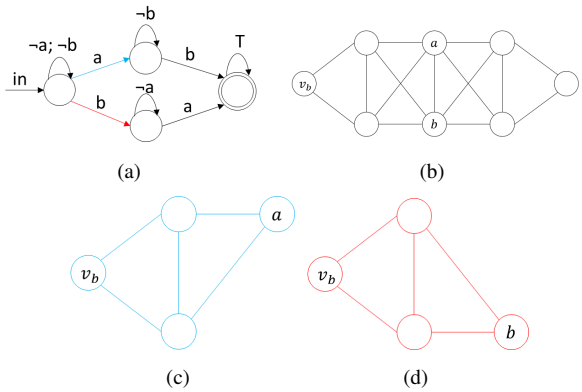

Fig. 2: 2(a) FSA $\mathcal{A}$ encoding $\Diamond a \wedge \Diamond b$; 2(b) simple labeled TS $\mathcal{T}$; 2(c) subset of $\mathcal{T}$ corresponding to taking edge with label a (in blue) from the in state in $\mathcal{A}$; 2(d) subset of $\mathcal{T}$ corrseponding to taking edge with label b (in red) from the in state in $\mathcal{A}$. Each time the recursive greedy algorithm is called, it operates a subset of $\mathcal{T}$ such as those in 2(c) or 2(d).

which corresponds to "visit nodes a and b." The specification is encoded in the FSA $\mathcal{A}$ (Fig. 2(a)). For example, traversing the edge shown in blue in Fig. 2(a) corresponds to traveling from v_b to the node labeled a and hence, planning on the subset of $\mathcal{T}$ shown in Fig. 2(c). Likewise, traversing the red edge in Fig. 2(a) corresponds to finding a path from v_b to the node labeled b in Fig. 2(d). In this way, we can repeatedly call the recursive planner of Chekuri and Pal [11] on the subsets of $\mathcal{T}$, like those in Figs. 2(c) and 2(d), instead of using it on the product of $\mathcal{A}$ and $\mathcal{T}$.

B. Construction of product automaton

Although we plan over the subsets of $\mathcal{T}$, the first step in our solution is the construction of a product automaton for the agent. This allows us to determine which subsets of $\mathcal{T}$ are relevant for each edge in the FSA $\mathcal{A}$. Given a specification ϕ, we construct an FSA $\mathcal{A}$, and takes it product with the TS $\mathcal{T}$.

Definition IV.1 (Product Automaton)**.** Given a TS T and an FSA A, we can define a *product automaton* $P = T \times A$ as a tuple $P = (S, s_0, Act_P, \Delta_P, F_P)$, where

- $S \subseteq V \times Q$ is a finite set of states;
- $s_0 = (v_0, q_0)$ is an initial state;
- $Act_P \subseteq Act$ is a set of input actions;
- $\Delta_P : S \times Act_P \rightarrow S$ is a deterministic transition function;
- $F_P \subseteq V \times F_{\mathcal{A}}$ is a set of accepting states.

The transition function is defined as $s' = \Delta_P(s, a)$ for $s = (v, q), s' = (v', q') \in S$ and $a \in Act_P$ if and only if $v' = \Delta(v, a)$ and $q' = \Delta_A(q, v)$.

After constructing the product automaton, we compute the *distance to satisfaction* $D(s)$ [14] for all nodes in P. This distance is equal to the shortest path $d(s, s')$ for $s' \in F_P$ through the product automaton. Computing $D(\cdot)$ for all nodes allows us to keep track of the necessary budget to complete the mission specified by ϕ.

C. Main algorithm

We now introduce the main algorithm for optimizing a submodular reward function (Alg. 1), which we call the Single Agent Constrained Path Planner (SACPP). This algorithm proceeds from the initial node of the FSA and uses the recursive greedy algorithm of Chekuri and Pal [11] to traverse each edge in the FSA, checking among feasible budgets. When two or more paths converge, the path with higher reward per unit budget expended is chosen.

In lines 1-11, the algorithm is initialized. During those steps, the product automaton is constructed and pruned, and the distance to acceptance is computed for each state in the product. If the distance to acceptance from the initial state is greater than the allotted budget $\mathcal{B}$, there is no solution. Otherwise, a copy of the states in $\mathcal{A}$ is created, called $ToCheck$, that keeps track of states in the FSA that have not been checked. Likewise, a set called $Next$ is initialized to contain node q_0, the initial FSA state. Finally, three sets are built, $B(q)$, $f_{\mathcal{R}}(q)$, and $\mathcal{R}(q)$. For each node $q \in Q$, these sets store information for each node in the FSA that the algorithm visits, namely the budget expended to reach that node, the reward collected to reach it, and the path taken to get there, respectively.

The main while loop (lines 12-45) executes until entries in $B(q)$, $f_{\mathcal{R}}(q)$, and $\mathcal{R}(q)$ have been computed for all $q \in Q$, at which point, the path with maximum reward among all paths reaching an accepting state is returned. The loop executes by creating the set of nodes $current$ from the previously constructed set $Next$, and removing those same nodes added to $current$ from the set to be checked (lines 13-15). For each node in $q \in current$, all possible transitions are checked by iterating over all symbols in Σ (line 16). If a transition to a state q' is possible, since transitions in the FSA correspond to visiting one and only one state in $\mathcal{T}$, the corresponding product states s and s' are known (17-20). The minimum feasible budget $bMin$ is $D(s) - D(s')$ while the maximum feasible budget $bMax$ is the difference between the total budget $\mathcal{B}$ and $B(q)$ (the budget expended to reach q and therefore to reach s) and $D(s')$ (lines 21-22).

For each edge in the automaton $\mathcal{A}$, we find the range of feasible budgets $b \in [bMin, bMax]$ that can be subtracted from the overall budget $\mathcal{B}$ to traverse that edge. Each call to the recursive planner builds a list $\mathcal{M}$ of paths for each corresponding feasible budget b. The parameter corresponding to the depth of the recursion, $iter$, is set to $\lceil 1 + \log b \rceil$ (line 24) to maintain our performance guarantee (see Sec. IV-F). If q' is not an accepting state (i.e., $q \notin F_{\mathcal{A}}$), the path that has the maximum ratio of reward to budget is chosen (line 29). The path yielding the maximum reward is not chosen because the reward functions under consideration are monotonic, so the maximum reward path would almost certainly expend the entire feasible budget, potentially yielding less optimal

results for the other edges to be checked[5]. If, however, $q' \in F_{\mathcal{A}}$, the path with maximum reward is chosen, since there are no further edges in the FSA to check (line 27)[6].

If q' has not already been visited by the algorithm, then the budget, reward and path chosen in line 27 or 29 is added to $B(q')$, $f_{\mathcal{R}}(q')$, and $\mathcal{R}(q')$, respectively (lines 30-32). If q' has already been visited, then the path with higher value of $f_{\mathcal{R}}(q')$ is chosen if $q' \in F_{\mathcal{A}}$ (lines 33-35), otherwise the path with the higher value of $\frac{f_{\mathcal{R}}(q')}{B(q')}$ is chosen (lines 37-38). Finally, if all incoming edges to q' have been checked, it is added to $Next$, the set of nodes to check for the next iteration of the while loop.

Algorithm 1 Single Agent Constrained Path Planner.

1: **function** SAPP
Input: $\mathcal{T}$, $\mathcal{A}$, $\mathcal{B}$
2: Construct product automaton $P = \mathcal{T} \times \mathcal{A}$;
3: Compute distance to acceptance $D(s)\, \forall s \in S$;
4: Prune unreachable states from $\mathcal{P}$;
5: **if** $D(s_0) > \mathcal{B}$ **then return** no solution;
6: $ToCheck \leftarrow Q$;
7: $Next \leftarrow q_0$;
8: $B(q_0) \leftarrow 0$; ▷ used budget
9: $stateV(q_0) \leftarrow v_0$; ▷ initial state
10: $f_{\mathcal{R}}(q_0) \leftarrow 0$; ▷ collected reward
11: $\mathcal{R}(q_0) \leftarrow \emptyset$; ▷ for residual reward calculation
12: **while** $ToCheck$ is not empty **do**
13: $current \leftarrow Next$;
14: $ToCheck \leftarrow ToCheck \setminus Next$;
15: $Next \leftarrow \emptyset$;
16: **for** $q, \sigma \in current \times \Sigma$ **do**
17: $q' \leftarrow \Delta_A(q, \sigma)$;
18: $s \leftarrow \{(q, stateV(q))\}$;
19: $stateV(q') \leftarrow \sigma$;
20: $s' \leftarrow \{(q', stateV(q'))\}$;
21: $bMin \leftarrow D(s) - D(s')$;
22: $bMax \leftarrow \leq \mathcal{B} - B(q) - D(s')$;
23: **for** $bMin \leq b \leq bMax$ **do**
24: $iter = \lceil 1 + \log b \rceil$;
25: $\mathcal{M}(q', b) \leftarrow RP(s, s', b, \mathcal{R}(q), iter, P)$;
26: **if** $q' \in F_{\mathcal{A}}$ **then**
27: $\chi \leftarrow \arg\max \{f_{\mathcal{R}}(m) \,|m \in \mathcal{M}\}$;
28: **else**
29: $\chi \leftarrow \arg\max \{f_{\mathcal{R}}(m)/b(m) \,|m \in \mathcal{M}\}$;
30: **if** $B(q')$ is empty **then**
31: Update $B(q')$, $f_{\mathcal{R}}(q')$, $\mathcal{R}(q')$;
32: **else**
33: **if** $q' \in F_{\mathcal{A}}$ **then**
34: **if** $f_{\mathcal{R}}(q') < f_{\mathcal{R}}(\mathcal{R}(q) \cup \chi)$ **then**
35: Update $B(q')$, $f_{\mathcal{R}}(q')$, $\mathcal{R}(q')$;
36: **else**
37: **if** $\frac{f_{\mathcal{R}}(q')}{B(q')} < \frac{f_{\mathcal{R}}(\mathcal{R}(q) \cup \chi)}{B(q) \cup c(\chi)}$ **then**
38: Update $B(q')$, $f_{\mathcal{R}}(q')$, $\mathcal{R}(q')$;
39: **if** All incoming edges for q' have been checked **then**
40: $Next \leftarrow Next \cup q'$;
return $\arg\max_{q \in F_{\mathcal{A}}} f_{\mathcal{R}}(\mathcal{R}(q))$;

[5]The choice to use the path with the best ratio instead of best absolute cost is not necessary for our algorithm's theoretical performance, but improves results in practice.

[6]Note that the path χ is a sequence of $b+1$ states using budget b.

D. Recursive Planner

The recursive planner is presented in Alg. 2. It is based on the recursive greedy algorithm in [11]. For considerations of length, we curtail our discussion of this algorithm to the difference from their algorithm. The original paper [11] should be consulted for further details. The main difference is that we prune the set of states we are considering as candidates for our path. The set $cands$ (line 6 in Alg. 2) contains our initial and final nodes s_i and s_j, as well as any nodes that share the same automaton state as s_i. Intuitively, this is the subgraph of P corresponding to one state in FSA $\mathcal{A}$. Note that it is, at largest, a copy of the entire graph of $\mathcal{T}$. Thus, our planning is at most carried out over $\mathcal{T}$, rather than over the entire product automaton.

Algorithm 2 Single Agent Recursive Planner. Used for planning portion of Alg. 1. Adapted from Chekuri and Pal [11].

1: **function** RP
Input: s_i, s_j, b, $\mathcal{R}$, $iter$, P
Output: $path$
2: **if** $d(s_i, s_j) > b$ **then return** no solution;
3: $path \leftarrow greedyPath(s_i, s_j)$;
4: Base case: $iter = 0$ **return** $path$;
5: $cands \leftarrow \{\{s_i, s_j\} \cup \{s_k = (v_k, q_k)\} \,|q_k = q_i\}$;
6: **for** $s_k \in cands$ **do**
7: **for** $1 \leq b_1 \leq b$ **do**
8: $path_1 \leftarrow RR(s_i, s_k, b_1, \mathcal{R}, iter - 1)$;
9: $b_2 \leftarrow b - b_1$;
10: $\mathcal{R}_2 \leftarrow \mathcal{R} \cup path_1$;
11: $path_2 \leftarrow RR(s_k, s_j, b_2, \mathcal{R}_2, iter - 1)$;
12: **if** $f_{\mathcal{R}}(path_1 \cup path_2) > f_{\mathcal{R}}(path)$ **then**
13: $path \leftarrow path_1 \cup path_2$;
return $path$

E. Complexity

The initial solution in [11], a recursive greedy algorithm we will refer to as RG, had a complexity of $\mathcal{O}\left((n\mathcal{B})^{\log n}\right)$, and the nonmyopic solution presented in [9] had complexity of $\mathcal{O}\left(\mathcal{B}^2 T\,(n\mathcal{B})^{\log n}\right)$, where n is the number of nodes in the environment graph, T is the number of tours (i.e., copies of the environment graph), and $\mathcal{B}$ is the overall budget for the set of tours. Our proposed solution has complexity of $\mathcal{O}\left(\mathcal{B}|\Sigma||Q|\,(|V|\mathcal{B})^{\log|V|}\right)$, where Σ is the input to the FSA and Q are the states of the FSA. To compare this complexity with that of [11] and [9], it should be noted that $|V| = n$. In our algorithm, the RG algorithm is called at most for each element of Q, each element of Σ, and a budget from 1 to $\mathcal{B}$, hence our complexity is the complexity of RG multiplied by $\mathcal{B}|\Sigma||Q|$. If we were to use the RG algorithm to conduct a search over the entire product automaton, the complexity would instead be $\mathcal{O}\left((\mathcal{B}|V||Q|)^{(\log|V||Q|)}\right)$. Thus, the exponent in our complexity is reduced by a factor of $\log|Q|$.

F. Optimality

To present the theoretical bound of our approach with respect to the optimal solution, we must first introduce

Lemma 1, which gives us an approximation guarantee on Alg. 2.

Lemma 1. [11] Given a pair of initial and final nodes (s_0, s_f) in a graph, a search depth $iter$, a set $\mathcal{R}$ for computing the residual reward, and a budget $\mathcal{B}$, let χ^* be the optimal path between s_0 and s_f with a length of at most $\mathcal{B}$. Let χ be a $\mathcal{B}$-length path between s_0 and s_f returned by Alg. 2. If $iter \geq \lceil 1 + \log \mathcal{B} \rceil$, then $f_{\mathcal{R}}(\chi) \geq \frac{1}{\lceil 1+\log \mathcal{B} \rceil} f_{\mathcal{R}}(\chi*)$.

Lemma 1 follows directly from Lemma 3.2 in [11]. This lemma will allow us to prove our result below in Theorem 1, since we call Alg. 2 in Alg. 1 to compute our overall solution.

Theorem 1. Let $\chi_{1:k}$ be a path consisting of sub-paths $\chi_1, \ldots, \chi_k$ returned by Alg. 1, consisting of k edges in $\mathcal{A}$, and returning reward $f_{\mathcal{R}}(\chi_{1:k})$. Let $f_{\mathcal{R}}(\chi^*_{1:k})$ be the optimal reward for traversing the same k edges. Denote the depth of the recursion for the k^{th} call to Alg. 2 as $iter_k$. Then if $iter_k \geq \lceil 1 + \log \mathcal{B}_k \rceil$,

$$f_{\mathcal{R}}(\chi_{1:k}) \geq \frac{\lceil 1 + \log \mathcal{B}_k \rceil}{\lceil 2 + \log \mathcal{B}_k \rceil} \min\left(\beta_{1:k-1}, \frac{1}{\lceil 1 + \log \mathcal{B}_k \rceil}\right) f_{\mathcal{R}}(\chi^*_{1:k}) \ , \tag{10}$$

where $\beta_{1:k-1}$ is the approximation guarantee for the first $k-1$ edges in $\mathcal{A}$ and $\mathcal{B}_k$ is the budget allocated for traversing the k^{th} edge.

Proof. Given a budget $\mathcal{B}_1$, let χ_1 be the path returned by Alg. 1. Assume that $iter_1 \geq \lceil 1 + \mathcal{B}_1 \rceil$. In light of Lemma 1,

$$f_{\mathcal{R}}(\chi_1) \geq \frac{1}{\lceil 1 + \log \mathcal{B}_1 \rceil} f_{\mathcal{R}}(\chi^*_1) \ , \tag{11}$$

where χ^*_1 is the optimal path for the same starting and ending nodes as χ_1. Likewise, we can say that

$$f_{\mathcal{R}'}(\chi_2) \geq \frac{1}{\lceil 1 + \log \mathcal{B}_2 \rceil} f_{\mathcal{R}'}(\chi^*_2) \ , \tag{12}$$

where $\mathcal{R}' = \mathcal{R} \cup \chi_1$, and χ^*_2 is the optimal path for the same starting and ending nodes as χ_2, if $iter_2 > \lceil 1 + \log \mathcal{B}_2 \rceil$.

The remainder of our proof follows the same structure as the proof Lemma 3.2 in [11]. We will denote $\chi_1 \cup \chi_2$ as $\chi_{1:2}$ and $\chi^*_1 \cup \chi^*_2$ as $\chi^*_{1:2}$. Starting from (12), we write

$$f_{\mathcal{R}'}(\chi_2) \geq \frac{1}{\lceil 1 + \log \mathcal{B}_2 \rceil} \left(f_{\mathcal{R}}(\chi^*_2 \cup \chi_1) - f_{\mathcal{R}}(\chi_1)\right) \tag{13}$$

$$\geq \frac{1}{\lceil 1 + \log \mathcal{B}_2 \rceil} \left(f_{\mathcal{R}}(\chi^*_2) - f_{\mathcal{R}}(\chi_1 \cup \chi_2)\right) \ , \tag{14}$$

where the first inequality follows from the definition of $f_{\mathcal{R}'}(\chi^*_2)$, and the second inequality follows from monotonicity of f. Then we can add (11) and (14) to get

$$f_{\mathcal{R}}(\chi_{1:2}) \geq \frac{1}{\lceil 1 + \log \mathcal{B}_1 \rceil} f_{\mathcal{R}}(\chi^*_1) + \frac{1}{\lceil 1 + \log \mathcal{B}_2 \rceil} \left(f_{\mathcal{R}}(\chi^*_2) - f_{\mathcal{R}}(\chi_{1:2})\right) \ , \tag{15}$$

which, by rearranging terms, becomes

$$f_{\mathcal{R}}(\chi_{1:2}) \geq \frac{\lceil 1 + \log \mathcal{B}_2 \rceil}{\lceil 2 + \log \mathcal{B}_2 \rceil} \left(\frac{1}{\lceil 1 + \log \mathcal{B}_1 \rceil} f_{\mathcal{R}}(\chi^*_1) + \frac{1}{\lceil 1 + \log \mathcal{B}_2 \rceil} f_{\mathcal{R}}(\chi^*_2)\right) . \tag{16}$$

Finally, if we let $\frac{1}{\alpha_{12}}$ denote $\min\left(\frac{1}{\lceil 1+\log \mathcal{B}_1 \rceil}, \frac{1}{\lceil 1+\log \mathcal{B}_2 \rceil}\right)$ we get the result

$$f_{\mathcal{R}}(\chi_{1:2}) \geq \frac{\lceil 1 + \log \mathcal{B}_2 \rceil}{\lceil 2 + \log \mathcal{B}_2 \rceil} \left(\frac{1}{\alpha_{12}} f_{\mathcal{R}}(\chi^*_{1:2})\right) . \tag{17}$$

Following this same process recursively, for k copies of the recursive greedy algorithm, we find the bound

$$f_{\mathcal{R}}(\chi_{1:k}) \geq \frac{\lceil 1 + \log \mathcal{B}_k \rceil}{\lceil 2 + \log \mathcal{B}_k \rceil} \min\left(\beta_{1:k-1}, \frac{1}{\lceil 1 + \log \mathcal{B}_k \rceil}\right) f_{\mathcal{R}}(\chi^*_{1:k}) \ , \tag{18}$$

where $\chi_{1:k}$ is the path from the initial to the k^{th} node in the FSA, $\chi^*_{1:k}$ is the optimal such path, and $\beta_{1:k-1}$ is the optimality bound at iteration $k-1$. □

Theorem 1 provides a lower bound on the performance of Alg. 1. More specifically, it gives a guarantee on the performance for choosing the same path through $\mathcal{A}$, using the same budget allocation for each segment of the path. Note, this guarantee is not relative to the global optimal, but the optimal for the same choice of edges and budgets for traversing FSA $\mathcal{A}$. This is because we concatenate sequences that are themselves approximate optimal solutions between two nodes. The concatenation of these sequences introduces an additional sub-optimality that we quantify with Theorem 1.

Remark 1. The approximation guarantee in Theorem 1 depends on the fact that for the k^{th} call to Alg. 2, $iter_k \geq \lceil 1 + \mathcal{B}_k \rceil$. In Alg. 1, $iter$ is specified in line 24 as exactly $\lceil 1 + \mathcal{B}_1 \rceil$. This flexible assignment of $iter$ results in a significant speed-up of the algorithm, without sacrificing the performance bound. Alternately, the user could specify a fixed value of $iter$ to be used at every call to Alg. 2, which could be used to improve performance, at the cost of of computational speed.

V. Simulation and Results

To investigate the efficacy of our results, we ran simulations for minimizing the duration of time since last visit, given by (7). We considered a transition system $\mathcal{T}$ consisting of 9 nodes (Fig. 1). The agent began at the base v_b with a budget $\mathcal{B} = 8$. The mission ϕ was specified by (5), which is "eventually visit nodes v_3, v_4, and v_b, and don't return to v_b until visiting v_3 and v_4". The FSA encoding (5) contained 5 nodes. After pruning unreachable states, the resulting product automaton consisted of 33 nodes.

The results of two simulations are summarized in Table I. In the first, we considered the problem with the TL constraints mentioned above. We used Alg. 1, which terminated in about 30 seconds and returned a total *residual* reward of 88. Recall that the residual reward (8) is the improvement in the reward function (7) with respect to evaluating it for the empty set.

The budget $\mathcal{B}$ was split into three budgets $\mathcal{B}_1$, $\mathcal{B}_2$, and $\mathcal{B}_3$ since two intermediate nodes v_3 and v_4 need to be visited before visiting v_b. For example, one way of splitting the budget is allocating $\mathcal{B}_1$ for the sub-path from v_b to v_3, $\mathcal{B}_2$

	Alg. 1 (with TL)	RG [11] (without TL)
Run time (s)	$\sim$30	$\sim$2500
Residual reward $f_{\mathcal{R}}$	88	92
Fraction of optimal	$\geq \frac{1}{4}$	$\geq \frac{1}{4}$

TABLE I: Summary of simulation results.

for the sub-path from v_3 to v_4, and $\mathcal{B}_3$ for v_4 to v_b. This simulation results in a performance guarantee of $f_{\mathcal{R}}(\chi_{1:3}) \geq \frac{1}{4} f_{\mathcal{R}}(\chi^*_{1:3})$.

In the second simulation, we considered the problem without the TL constraints, using the same transition system for the agent, and only requiring it to start at the base and return to the base. This case is expected to return a better reward than the case with TL constraints (since Alg. 1 conducts a search over a restricted set of paths, which may eliminate the ones with higher rewards). We ran the RG algorithm [11], which terminated in approximately 42 minutes and returned a total residual reward of 92. In this case, the approximation guarantee was obtained as $f_{\mathcal{R}}(\chi) \geq \frac{1}{4} f_{\mathcal{R}}(\chi^*)$.

Running the RG algorithm on the complete product automaton would be computationally prohibitive, hence our comparison of the RG algorithm in the absence of TL constraints.

VI. Conclusion

In this paper we considered an agent moving in a discretized environment to maximize a submodular reward function while satisfying temporal logic constraints. Our main contribution was extending the recursive-greedy algorithm proposed in [11] for cases involving complex mission specifications expressed as temporal logics. Typically, the problems containing temporal logics are tackled by constructing a product automaton of the transition system (i.e., motion model) and the automaton encoding the desired specification. Using existing methods for maximizing submodular functions in conjunction with planning on the product automaton would be computationally infeasible. As opposed to the standard approach of solving over the entire product automaton, we proposed a solution conducted over a portion of the transition system with the relevant edges in the automaton. This approach exhibits a significantly lower complexity than a solution obtained from the overall product automaton. We presented a theoerical bound regarding the performance of the algorithm and illustrated the proposed approach via simulations. As future work, we plan to extend these results for multi-agent systems.

References

[1] J. Binney, A. Krause, and G. Sukhatme. Informative path planning for an autonomous underwater vehicle. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4791–4796. IEEE, 2010.

[2] F. Bourgault, A.A. Makarenko, S.B. Williams, B. Grocholsky, and H.F. Durrant-Whyte. Information based adaptive robotic exploration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 540–545. IEEE, 2002.

[3] B. Charrow, V. Kumar, and N. Michael. Approximate representations for multi-robot control policies that maximize mutual information. *Autonomous Robots*, 37(4):383–400, 2014.

[4] S.K. Gan, R. Fitch, and S. Sukkarieh. Online decentralized information gathering with spatial–temporal constraints. *Autonomous Robots*, 37(1):1–25, 2014.

[5] G.A. Hollinger and G.S. Sukhatme. Sampling-based robotic information gathering algorithms. *The International Journal of Robotics Research*, 33(9):1271–1287, 2014.

[6] M. Schwager, P. Dames, D. Rus, and V. Kumar. A multi-robot control policy for information gathering in the presence of unknown hazards. In *Proceedings of the International Symposium on Robotics Research (ISRR)*, 2011.

[7] G.L. Nemhauser, L.A. Wolsey, and M.L. Fisher. An analysis of approximations for maximizing submodular set functionsi. *Mathematical Programming*, 14(1):265–294, 1978.

[8] A. Krause and C. Guestrin. Submodularity and its applications in optimized information gathering. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(4):32, 2011.

[9] A. Meliou, A. Krause, C. Guestrin, and J.M. Hellerstein. Nonmyopic informative path planning in spatio-temporal models. 2007.

[10] A. Singh, A. Krause, C. Guestrin, and W.J. Kaiser. Efficient informative sensing using multiple robots. *Journal of Artificial Intelligence Research*, pages 707–755, 2009.

[11] C. Chekuri and M. Pal. A recursive greedy algorithm for walks in directed graphs. In *Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on*, pages 245–253. IEEE, 2005.

[12] H. Zhang and Y. Vorobeychik. Submodular optimization with routing constraints. In *AAAI*, pages 2346–2352. Citeseer, 2016.

[13] D. Aksaray, K. Leahy, and C. Belta. Distributed multi-agent persistent surveillance under temporal logic constraints. *IFAC-PapersOnLine*, 48(22):174–179, 2015.

[14] X.C. Ding, M. Lazar, and C. Belta. LTL receding horizon control for finite deterministic systems. *Automatica*, 50(2):399–408, 2014.

[15] S. Karaman and E. Frazzoli. Linear temporal logic vehicle routing with applications to multi-UAV mission planning. *Int. Journal of Robust and Nonlinear Control*, 21(12):1372–1395, 2011.

[16] H. Kress-Gazit, G.E. Fainekos, and G.J. Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE Trans. on Robotics*, 25(6):1370–1381, 2009.

[17] T. Wongpiromsarn, U. Topcu, and R.M. Murray. Receding horizon temporal logic planning. *IEEE Trans. on Automatic Control*, 57(11):2817–2830, Nov 2012.

[18] S. Smith, J. Tumova, C. Belta, and D. Rus. Optimal Path Planning for Surveillance with Temporal Logic Constraints. *Int. Journal of Robotics Research*, 30(14):1695–1708, 2011.

[19] C. Baier and J.P. Katoen. *Principles of model checking*, volume 26202649. MIT press Cambridge, 2008.

[20] A. Jones, M. Schwager, and C. Belta. A receding horizon algorithm for informative path planning with temporal logic constraints. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5019–5024. IEEE, 2013.

[21] A. Jones, M. Schwager, and C. Belta. Information-guided persistent monitoring under temporal logic constraints. In *American Control Conference (ACC), 2015*, pages 1911–1916. IEEE, 2015.

[22] K. Leahy, A. Jones, M. Schwager, and C. Belta. Distributed information gathering policies under temporal logic constraints. In *IEEE International Conference on Decision and Control (CDC)*, pages 6803–6808. IEEE, 2015.

[23] F. Imeson and S. Smith. Multi-robot task planning and sequencing using the sat-tsp language. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5397–5402. IEEE, 2015.

[24] O. Kupferman and M.Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.

[25] T. Latvala. Efficient model checking of safety properties. In *Model Checking Software*, pages 74–88. Springer, 2003.

[26] A. Krause and C.E. Guestrin. Near-optimal nonmyopic value of information in graphical models. *arXiv preprint arXiv:1207.1394*, 2012.

[27] A. Krause and D. Golovin. Submodular function maximization. In *Tractability: Practical Approaches to Hard Problems*, 2012.

[28] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages and Computation: For VTU, 3/e*. Pearson Education India, 1979.