Multi-Agent Task Allocation using Cross-Entropy Temporal Logic Optimization

Christopher Banks, Sean Wilson, Samuel Coogan and Magnus Egerstedt

***Abstract*— In this paper, we propose a graph-based search method to optimally allocate tasks to a team of robots given a global task specification. In particular, we define these agents as discrete transition systems. In order to allocate tasks to the team of robots, we decompose finite linear temporal logic (LTL) specifications and consider agent specific cost functions. We propose to use the stochastic optimization technique, cross entropy, to optimize over this cost function. The multi-agent task allocation cross-entropy (MTAC-E) algorithm is developed to determine both when it is optimal to switch to a new agent to complete a task and minimize the costs associated with individual agent trajectories. The proposed algorithm is verified in simulation and experimental results are included.**

I. INTRODUCTION

Multi-agent task allocation facilitates the ability for groups of robots to execute complex tasks over a limited time frame [1], [2]. For example, autonomous surveillance, search and rescue and environment monitoring all require the delegation of multiple tasks for each agent [3]. In addition to this, multi-agent systems enable redundancies in task allocation, fault tolerance [4] and faster execution of time based tasks [5] unlike task handling done by single agents. Naturally, the question of how to allocate these tasks effectively arises. Unfortunately, delegating tasks to individual agents to produce global behavior becomes difficult for human operators as the swarm size increases [6]. One solution to this problem is to design desired global task specifications for the entire swarm and enable the collective to autonomously allocate tasks among themselves to achieve the provided global specifications. One way of providing global task specifications to a team of robots is through a linear temporal logic formulation [7].

Linear temporal logic (LTL) formulas provide formal, mathematical guarantees for the performance of a system and are relatively close to natural language syntax which aids users in easier control of robotic systems that satisfy these specifications. In [8], an optimal graph search algorithm was developed to assign multiple agents tasks based on the environment transition system and LTL specification given. Often in works that utilize temporal logic for high-level task specification for multiple agents, a product automaton of all agents must be created which can be computationally demanding [9].

This material is partially based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1650044 and in part through the grant ARL DCIST CRA W911NF-17-2-0181.

All authors are affiliated with Georgia Institute of Technology, Atlanta, GA 30332, USA, {cbanks34, sean.t.wilson, sam.coogan, magnus}@gatech.edu

To avoid product automaton construction, tasks can be decomposed, essentially assigning agents a subset of the provided global specification. Algorithms that give finite time horizon plans for each agent [10] decompose tasks in an adaptive manner, yet, suffer from synchronization constraints between agents. Approaches that use MILP for multi-agent planning [11] are limited to linear constraints and utilize fixed costs for agent transitions. Work in [12] utilizes a global specification as well as individual specifications that the agents must specify. This is in contrast to [13] where a global specification is given and is decomposed for any number of agents in the composition of the problem. However, like previously mentioned work, solutions are limited to fixed, assigned costs between actions for each agent.

In this paper, synchronization requirements are avoided and we leverage the decomposition framework of [14] as a viable alternative for developing multi-robot task allocation and consider each agent when delegating tasks, switching between agents based on an assigned cost function. We also propose a framework for systems where action costs are not explicitly known a priori (e.g. "the control input cost of performing a rescue operation") and instead are generated via a cost function. To achieve this individual allocation strategy from the high level objectives given by LTL, cross-entropy optimization, an importance sampling technique, is used to determine when to switch tasks between agents.

Generally, robot tasks should not be assigned to robots uniformly as some robots may be better suited to perform the task due to their proximity, voltage levels, previous actions, or other factors. This can be considered as a set of system cost constraints that must be conformed to over time. Cross entropy optimization, as described in [15], is a form of importance sampling that estimates rare-event probabilities, which we leverage to design cost functions to optimize over as "rare-event" distributions (i.e. characterized as events that occur infrequently). In this paper, we develop a novel algorithm designed to sample trajectories and converge to a desired cost function to determine the best robot for continued satisfaction of a goal specification. This algorithm advances the state-of-the art by introducing a formulation that allows users to design agent specific cost functions – for a homogeneous team of robots with equivalent dynamics – and dynamically allocate tasks over time while satisfying a global specification in addition to constraints of the environment or individual agent dynamics.

This paper proceeds in the following manner. We develop the mathematical foundation for defining discrete transition systems and finite LTL in Section II. A formal problem

Manuscript 1244 submitted to 2020 International Conference on Robotics and Automation (ICRA). Received September 15, 2019.

formulation is given in Section III. We present the MTAC-E algorithm in Section IV. Finally, we present a case study in simulation and provide experimental results in Section V and conclude in Section VI.

II. Finite LTL for Global Task Specification

To encode global task specifications we use LTL, allowing users to design global goals for swarm execution. In addition to this, global goals enable scalability (i.e. goals are independent of the swarm size) and reduce cognitive load on the designer as they do not have to assign each agent a specification. This type of interaction modality is easily adapted from temporal logic formula, in addition to providing formal guarantees for global specification satisfaction.

In this section, we provide a brief background on finite LTL, a class of LTL specifications well-suited for formally representing planning problems [16] and interpreting finite sequences [17]. For a more in-depth reading on finite LTL we refer the reader to [18]. We also provide definitions for generating the discrete transition system of an autonomous agent, the transition system of a team of robots and the decomposition framework, all of which will be utilized in Section V. There, as an example, the robots will execute a fire-fighting scenario in which each agent will be given tasks (e.g. carry water, visit regions of interest, etc.) expressed through a global LTL specification.

LTL specifications ϕ are defined as logic formalisms suited for specifying linear time properties [19], [20]. In order to create a formulation for decomposing, we consider finite LTL specifications. Finite LTL specifications are insensitive to infiniteness [18] and include classes of LTL like co-safe LTL. These specifications are defined over finite sequences of observations and the notation $\sigma \models \phi$ indicates that the finite sequence σ satisfies ϕ. The sequences are generated from the set of propositions, defined in the following.

Definition 1: Let $\Pi = \{\pi_0, \ldots, \pi_k\}$ be the finite set of atomic propositions. Each proposition π_i maps from system state to true $(\top)$ or false $(\bot)$ and enables us to define a Boolean property of the state space (e.g. "Is the robot in area G?").

Any finite LTL specifications can be represented via a constructed non-deterministic finite automaton (NFA) [14], which we define below.

Definition 2: A non-deterministic finite automaton (NFA) is given as the tuple $\mathcal{F} = (Q, Q_0, \beta, \delta, F)$ such that:

- Q is a set of states
- Q_0 is a set of initial states
- β is the set of Boolean formulas defined over the proposition set (Π)
- δ is a set of transition conditions such that $\delta : Q \times Q \to \beta$
- F is a set of accepting final states

Given finite runs $q = q(0) \ldots q(T) \to Q$ a sequence σ – defined as a sequence of propositions π_i from Π – satisfies q if it enables a transition from $q(0)$, an initial state, to $q(T) \in F$. These transistions, generated from $\delta(q, q') = \{\beta_i\}$, map onto a subset of the Boolean formulas, β, which evaluate to true if the proposition, $\pi(t)$, from σ satisfies it. Moreover, the NFA can be constructed from a finite LTL formula ϕ where a finite sequence $\sigma \models \phi$ if and only if σ successfully produces a run q such that $q(T) \in F$. Throughout this paper, finite LTL will be referred to as LTL.

A. Defining Transition Systems

The framework for defining task decomposition [14] involves creating several state transition systems for a robotic system. From this discrete planning framework, we are able to decompose a product automaton containing multiple agents into independent tasks that can be handled by each agent, while also satisfying a given goal specification. The definition of the robot transition system, $\mathcal{R}$, follows.

Definition 3: The robot transition system is defined as a tuple $\mathcal{R} = (S_\mathcal{R}, S_{\mathcal{R},0}, A_\mathcal{R}, \Pi_\mathcal{R}, \Lambda_\mathcal{R})$ such that:

- $S_\mathcal{R}$ is a set of robot states
- $S_{\mathcal{R},0} \subset S_\mathcal{R}$ is the set of initial robot states
- $A_\mathcal{R}$ is a set of available robot actions
- $\Pi_\mathcal{R}$ is the set of robot propositions
- $\Lambda_\mathcal{R} : S_\mathcal{R} \to 2^{\Pi_\mathcal{R}}$ is a labeling function that assigns atomic propositions to states.

The robot transition system captures the entire internal state of the robot and transitions are based on the actions, $A_\mathcal{R}$ available to the robot at each state. We next define the environment transition system $\mathcal{E}$ to capture the properties of the regions of interest for the agents.

Definition 4: The environment transition system is defined as a tuple $\mathcal{E} = (V_\mathcal{E}, E_\mathcal{E}, \Pi_\mathcal{E}, \Lambda_\mathcal{E})$ such that:

- $V_\mathcal{E}$ is a set of environment vertices
- $E_\mathcal{E}$ is a set of edges between vertices where $E_\mathcal{E} \subseteq V_\mathcal{E} \times V_\mathcal{E}$
- $\Pi_\mathcal{E}$ is the set of environment propositions
- $\Lambda_\mathcal{E} : S_\mathcal{V} \to 2^{\Pi_\mathcal{E}}$ is a labeling function that assigns atomic propositions to locations

The product automaton $\mathcal{A}$ is used to define the internal state and external location of the agent throughout the planning space.

Definition 5: The agent transition system is given as a product transition system $\mathcal{A} = \mathcal{E} \otimes \mathcal{R} = (S_\mathcal{A}, S_{\mathcal{A},0}, A_\mathcal{A}, \Pi_\mathcal{A}, \Lambda_\mathcal{A})$ such that:

- $S_\mathcal{A} = V_\mathcal{E} \times S_\mathcal{R}$ are the combined location and internal states of the agent
- $S_{\mathcal{A},0} = \{(v, s_0) \in S_\mathcal{A} : s_0 \in S_{\mathcal{R},0}\}$ is the set of initial agent states
- $A_\mathcal{A} \subseteq S_\mathcal{A} \times S_\mathcal{A}$ are the actions available to the agent
- $\Pi_\mathcal{A} \subseteq \Pi_\mathcal{E} \times \Pi_\mathcal{R}$ is the set of agent propositions
- $\Lambda_\mathcal{A} : S_\mathcal{A} \to 2^{\Pi_\mathcal{A}}$ is a labeling function that assigns atomic propositions to agent states

In this definition, the set of actions $A_\mathcal{A}$ are available to a robot based on both its internal state and location in the environment. Additionally, the actions are restricted in that only actions that are available at states which satisfy the Boolean transition formula, $\xi : A_\mathcal{A} \to \psi$, are included. More formally,

$$A_\mathcal{A} = \{a = ((v, s), (v', s')) \in S_\mathcal{A} \times S_\mathcal{A} : (v, v') \in E_\mathcal{E} \wedge (s, s') \in A_\mathcal{R} \wedge \Lambda_\mathcal{A}((v, s)) \models \xi(a)\}$$

Now that we have the agent automata defined for all agents, we can define the planning automaton $\mathcal{P}$ for the entire system.

Manuscript 1244 submitted to 2020 International Conference on Robotics and Automation (ICRA). Received September 15, 2019.

Definition 6: The planning automaton $\mathcal{P}$ is a product automaton of the NFA and agent transition system where $\mathcal{P} = \mathcal{F} \otimes \mathcal{A} = (S_\mathcal{P}, S_{0,\mathcal{P}}, A_\mathcal{P})$ such that:

- $S_\mathcal{P} = Q \times S_\mathcal{A}$ is the set of states
- $S_{0,\mathcal{P}} = \{(q_0, s) \in S_\mathcal{P} : q_0 \in Q_0 \wedge s \in S_{\mathcal{A},0}\}$ is the set of initial states
- $A_\mathcal{P} = \{((q,s),(q',s')) \in S_\mathcal{P} \times S_\mathcal{P} : (s,s') \in A_\mathcal{A} \wedge \beta(s) \models \delta(q,q')\}$ is the set of actions

With the planning automaton $\mathcal{P}$, only sequences, σ – with propositions $\Pi_\mathcal{A}$ – that satisfy the LTL specification ϕ are accepted.

B. Decomposition Set

Given a multi-agent system with N agents, each represented according to the automata $\mathcal{P}^i$, defined previously, we seek to decompose the global LTL specification ϕ such that parts of it can be assigned to the set of agents based on their cost functions. Moreover, using task decomposition, we wish to generate independent sequences of action/state pairs from $\mathcal{P}^i$ to satisfy ϕ where sequences are $s^i = s_0 a_0, \ldots, a_n s_n$. We give the following definition of finite LTL task decomposition.

Definition 7: [14] Let $\mathcal{T}_i$ with $i \in \{1, \ldots, n\}$ be a set of finite LTL task specifications and σ_i denote any sequence such that $\sigma_i \models \mathcal{T}_i$. These tasks are called a decomposition of the finite LTL mission specification ϕ if and only if:

$$\sigma_{j_1} \ldots \sigma_{j_i} \ldots \sigma_{j_n} \models \phi \tag{1}$$

for all permutations of $j_i \in \{1, \ldots, n\}$ and all respective sequences σ_i.

From this definition of decomposition we can create the decomposition set $\mathcal{D} \subseteq \mathcal{Q}$ of the NFA $\mathcal{F}$ developed from ϕ. This set contains all states q for which the pair of tasks $\mathcal{T}_1^q, \mathcal{T}_2^q$, where q is a state in the decomposition set $\mathcal{D}$, define a valid decomposition. For a proof of this property, we refer the reader to [14].

We use this to avoid generating a large product automaton of the transition system of agents and automaton representation of the finite LTL specification. This greatly reduces the computational complexity usually encountered with systems involving a large number of agents. We define team product automata, $\mathcal{T}$, with the following definition.

Definition 8: The team model automaton $\mathcal{T}$ is a union of the N local product automatons $\mathcal{P}^i$ with $i \in \{1, \ldots, N\}$ where the tuple is $\mathcal{T} = (S_\mathcal{T}, S_{0,\mathcal{T}}, A_\mathcal{T}, F_\mathcal{T})$ such that:

- $S_\mathcal{T} = \{(r,q,s) : r \in \{1, \ldots, N\}, (q,s) \in S_\mathcal{P}^i\}$ is the set of states
- $S_{0,\mathcal{T}} = \{(r,q,s) \in S_\mathcal{T} : r = 1\}$ is the set of initial states, with r being a randomly assigned initial agent
- $A_\mathcal{T} = \bigcup_i A_\mathcal{P}^i \cup \zeta$ is the set of actions, including the switch transitions Z
- $F_\mathcal{T}$ is the set of accepting final states

Switch transitions, Z, allow our algorithm to select a new agent within the product automaton to complete the satisfaction of the specification.

Definition 9: The switch transitions in $\mathcal{T}$ are given by $Z \subset S_\mathcal{T} \times S_\mathcal{T}$. A transition $\zeta = ((r_s, q_s, s_s), (r_t, q_t, s_t)) \in Z$ if and only if [14]:

- $r_s \neq r_t$: the agents are different
- $q_s = q_t$: the progress of the NFA is preserved
- $r_t = r_s + 1$: A new agent is selected
- $s_t = s_{0,\mathcal{A}}^{r_t}$: The new state is the initial state of a new agent
- $q_s \in \mathcal{D}$: the state is in the decomposition set of the NFA

III. Problem Formulation

With discrete transititon systems defined for a homogenous team of agents and a decomposition framework, we turn to our problem formulation.

Problem: For a given set of homogenous agents, distribute tasks among these agents considering discrete agent transition systems with unknown action costs. Distribute these tasks while minimizing individual agent cost functions $f_i(\cdot)$, given by the operator before execution, for agents $i, \ldots, N$.

We demonstrate this problem as a firefighting quadcopter scenario in Section V. In this problem, we designate N quadcopters, each defined by discrete product automata as our set of homogeneous agents, with no action costs to transition between states. The swarm of robots is given the global task of surveying goal locations within the state space, acquiring water and transporting it to the desired location while obeying the constraints of the environment. We solve this problem using the MTAC-E algorithm proposed below.

IV. MTAC-E Algorithm

Solution: We propose the Multi-Agent Task Allocation Cross-Entropy (MTAC-E) Algorithm to delegate tasks to a set of agents. Previously, we defined a decomposition framework in Section II; given we have designed cost functions for each agent in the problem, we need a way to find optimal trajectories by minimizing these cost functions. To find the associated minimized costs, we propose using cross-entropy optimization. In this framework, we use cost functions optimized via cross-entropy as opposed to static actions costs defined at discrete state transitions. This additional flexibility in problem design allows operators to minimize over individual agent cost functions and use generalized functions for entire agent trajectories when the cost to perform an action is not known. We present a brief overview of cross entropy and our algorithm in the following sections.

A. Cross-Entropy Optimization

Cross-entropy optimization is a method of importance sampling for probablistically rare events. The algorithm design for using cross-entropy with motion planning [15] can be generalized as the following:

1) Generate a set of sample trajectories (J) from a distribution $p(\cdot, x)$ and calculate cost $\mathcal{J}(\cdot)$ for each trajectory
2) Update the distribution, p, using a subset of samples (κ), until the sampling distribution converges to a desired cost (λ) and delta function over the optimal trajectory

The subset of sampled trajectories with the lowest cost (i.e. $\kappa \subset J$) is defined such that $|\kappa| = \rho|J|$, where typically $10^{-1} \leq \rho < 0.3$. This subset is known as an "elite set" and provides a new sampling space to generate the

Manuscript 1244 submitted to 2020 International Conference on Robotics and Automation (ICRA). Received September 15, 2019.

distribution p. In this work, we sample trajectories according to a multi-variate Gaussian distribution $\mathcal{N}(\mu, \mathbf{\Sigma})$ such that $\mu = [\mu_0, \ldots, \mu_n]^T$ for n equally spaced points along the set of sampled trajectory. The co-variance matrices, $\Sigma = [\Sigma_0, \ldots, \Sigma_n]^T$ form an $np \times p$ matrix with Σ_i initially set to the identity matrix, I. Expectation-Maximization [21] is used to update the means and co-variances for the newly sampled trajectories. In the next section, we describe the multi-agent task allocation cross-entropy (MTAC-E) algorithm.

B. Algorithm

The algorithm developed in this paper, provided in pseudocode format in Algorithm 1, can be described in four steps: (1) given the initial state of the agent product automaton, find the cost of transitioning to the next state using cross-entropy and the cost function assigned to the agent, (2) if this state is contained in the decomposition set, check all other agent cost functions and, (3) if an agent has a lower cost, switch to this agent for the remainder of the algorithm or until a new switch is determined, (4) this process in continued until the end state is found and corresponding trajectories are returned to all agents for execution.

The algorithm receives as input the product automaton, $\mathcal{T}$, the decomposition set, $\mathcal{D}$, an optimal cost for each agent to minimize towards, λ, the elite set modifier, ρ, an initial sampling distribution, p and the number of times to iterate the sampling procedure, K. In Line 1, the initial state, p_i, the agent of p_i ,α_i, and the current sequence of states visited by agent i, $sequences_i$, are initialized. We recall Definition 8 of the team product automaton in this framework such that via a standard BFS search, once the state $p \in final\_states(\mathcal{T})$ is found and a sequence is generated that reaches this state, the LTL specification is satisfied.

The cross-entropy optimization technique in Line 7 is utilized in the function `cost_to_go`. An initial probability distribution is provided for each agent with initial means and variances. Also, elite set modifiers (ρ), an optimal cost (λ), and a bounding maximum iteration number (K) are supplied as input. The function samples from the given distribution and iterates until either the cost function has been met or the maximum iterations has been exceeded and returns the trajectories for each agent $(\eta_i(t), \ldots, \eta_n(t))$.

For states in the decomposition set ($\mathcal{D}$), a cost is calculated from each in Line 9 and if one of the costs is less than the current agent's cost ($cost_i$) the agents are swapped and the new agent j continues the remainder of the sequence until the next potential switch transition occurs.

This algorithm will return a set of trajectories N with each agents individual trajectory $\eta_i(t)$.

C. Complexity

We give a brief overview of the complexity of the algorithm and compare it to other methods for task allocation using temporal logic. Size analysis to search through LTL automata for satisfying sequences is well-known [22]. Generally, a trajectory, η can be checked if it satisfies the automata $\mathcal{A}_\phi$ in $O(|\eta| \cdot |\mathcal{A}_\phi|)$, denoting a bilinear complexity in the length of the trajectory and in the size of the automata. Leveraging task decomposition, the size of our team automaton, $\mathcal{T}$ is much smaller than one created via a product automata (i.e. $\mathcal{A}_{prod} = \mathcal{P}_i \otimes \mathcal{P}_{i+1}, \ldots, \mathcal{P}_{N-1} \otimes \mathcal{P}_N$), where N is the number of agents. In our work, we check trajectories for membership in an agent planning automaton, $\mathcal{P}_i$, which is equivalent to the number of NFA states $\mathcal{F}$ times the number of agent states $\mathcal{A}$ or $|\mathcal{P}_i| = |\mathcal{F}| \cdot |S_\mathcal{A}|$ unlike automata produced by constructing a product where $|\mathcal{A}_{prod}| = |\mathcal{F}| \cdot |S_\mathcal{A}|^N$, thus $|\mathcal{P}_i| \ll |\mathcal{A}_{prod}|$. Due to the checking of N agents in our framework, our algorithm has a complexity of $O(N \cdot (|\eta| \cdot |\mathcal{P}_i|))$. Recall, that product automata have states that grow exponentially with the number of agents therefore, due to our algorithm being linear in the number of agents, N, we show our algorithm is far more scalable than other methods utilizing product automata for task allocation. In addition to this, the runtime of the MTAC-E Algorithm, while heavily dependent on cost function choice and size of planning automaton, is ~ 300 seconds for the task allocation of three agents.

Algorithm 1: MTAC-E Algorithm

input : product automaton $\mathcal{T}$, decomposition set $\mathcal{D}$, optimal cost λ, elite set modifier ρ, sampling distribution $p(\mu_0, v)$, iteration number K
output: set of trajectories $\mathcal{N}$

1 p_i := initial_state($\mathcal{T}$)
2 $\alpha_i \rightarrow p_i$:= agent i in initial state
3 $sequences_i$:= sequence of states visited by agent i
4 $p \rightarrow p_i$:= set p to initial state
5 **while** $p \notin$ *final_states*($\mathcal{T}$) **do**
6 **for** q *in neighbors(p)* **do**
7 $\eta_i(t)$, $cost_i \rightarrow$ `cost_to_go`(α_i, q, $sequences_i$, λ, $p(\cdot, v)$, K, ρ)
8 **if** $q \in \mathcal{D}$ **then**
9 $\eta_{j:n}(t)$, $cost_{j:n} \rightarrow$ `cost_to_go`($\alpha_{j:n}$, q, $sequences_{j:n}$,λ, $p(\cdot, v)$, K, ρ)
10 **end**
11 **if** $cost_{j:n} < cost_i$ **then**
12 $\alpha_i \rightarrow \alpha_j$
13 $p \rightarrow p_j$
14 $sequences_j = sequences_j + sequences_{j,p \rightarrow q}$
15 **end**
16 **else**
17 $sequences_i = sequences_i + sequences_{i,p \rightarrow q}$
18 **end**
19 **end**
20 **end**
21 $\mathcal{N} = \{\eta_i(t), \ldots, \eta_n(t)\}$
22 **return** $\mathcal{N}$

V. Case Study: Fire Fighting Drones

We motivate the application of Algorithm 1 with a fire-fighting UAVs scenario. For example, each agent may be a fire-fighting autonomous aircraft capable of collecting water, extinguishing fires and surveying goal locations. These UAVs are given the following global goal: "*eventually* visit LOC_1 and LOC_2 and *always* ensure visiting $SMOKE$ implies $CARRYING$". Using LTL, this specification can be

represented as $\phi = \Diamond LOC_1 \wedge \Diamond LOC_2 \wedge \Box(SMOKE \implies CARRYING)$.

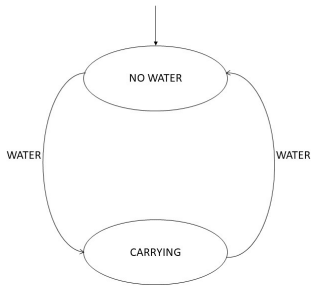

Fig. 1. A transition system for a single agent which describes the internal state of a robot ($\mathcal{R}_i$). All robots start at the initial state 'NO WATER' and a location-based transition is used to determine when to transition to the 'CARRYING' state. If a robot is in the environment state that satisfies the 'WATER' proposition, the robot can transition to the 'CARRYING' state.

According to our discrete planning framework, we define the internal state of the robot using Definition 3 where our robot is represented by a two state transition system with a transition denoted by whether it has visited the water location in the environment. A robot transitioning from the 'NO WATER' state to the 'CARRYING' state indicates the 'WATER' proposition was true in the environment during that transition. In Fig. 1 we represent the discrete internal transition system of robot i as ($\mathcal{R}_i$).

The environment transition system, Fig. 2, is represented by a set of nodes corresponding to states with adjacent nodes in the graph representing neighbors for potential paths through the state space. In simulation and experiment, we represent each node as an ellipsoid in $\mathbb{R}^3$. Formally, these ellipsoids have the following form:

Definition 10: The environment proposition set $\Pi_{\mathcal{E}} = \{\mathcal{E}_i, \ldots, \mathcal{E}_n\}$ is defined as:

$$E_j(r) = \frac{(r_x - x_j)^2}{a^2} + \frac{(r_y - y_j)^2}{b^2} + \frac{(r_z - z_j)^2}{c^2} \quad (2)$$

$$\mathcal{E}_j = \{r \in \mathbb{R}^3 \quad | \quad E_j(r) \leq 1\} \quad (3)$$

where(r_x, r_y, r_z) is the pose of the quadcopter, (x_j, y_j, z_j) is the position of a region of interest ($\mathcal{E}_j$) with index j and a, b, and c are the x-radius, y-radius and z-radius of the regions, respectively. We define these three constant radii $(a, b, c) \in \mathbb{R}_{>0}$ for the regions to represent the volume covered by each ellipsoid in our experiments and note they are equivalent for all ellipsoids.

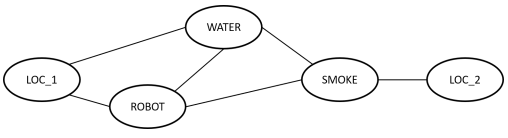

Fig. 2. The environment transition system where each state indicates a desired region of interest. The initial state of the environment is the 'ROBOT' state. In the fire fighting example, LOC_2, cannot be reached unless the quadcopter passes through the SMOKE region.

Using this definition, discrete transitions are identified when the relative position of a quadcopter transitions inside any of the regions of interest defined in the state space. In our case study, the environment proposition set is $\Pi_{\mathcal{E}} = \{WATER, SMOKE, LOC_1, LOC_2\}$. By taking the product we can generate the full agent automaton for each agent i such that $\mathcal{A}_i = \mathcal{E} \otimes \mathcal{R}_i$ shown in Fig. V. Following the standard procedure for developing automata for robotic systems we generate a NFA from the finite-LTL specification and take the product with $\mathcal{A}_i$ for each agent to get P, an automaton that only accepts runs that satisfy the LTL specification and agent transition system.

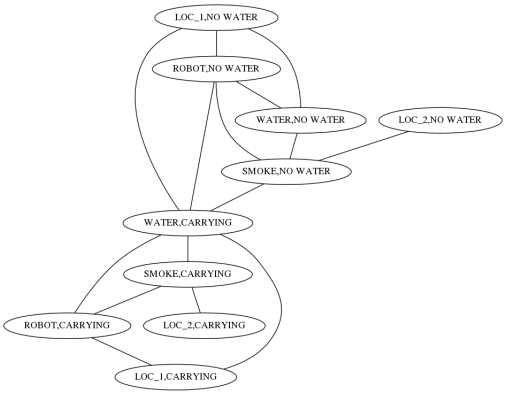

Fig. 3. The full agent transition system for a quadcopter. Transitions to the 'CARRYING' state can only be fulfilled once the agent has retrieved water from the environment node.

A. Deriving the Control Input

Utilizing the differentially flat dynamics [23] of the quadcopters, we represent the inputs and outputs of the system as algebraic function of chosen flat outputs and their derivatives. From this property, trajectories can be generated by leveraging the nonlinear dynamics of the quadcopters. This leads to the ability to plan smooth trajectories that are three-times continuously differentiable functions, $\eta(t) \in C^3$, in the output space that can be converted back analytically into feasible trajectories for the full state of the quadcopters. We utlilize a virtual input $u \in \mathbb{R}^3$ from [24] that controls a chain of integrator dynamics for the differentially flat outputs of the system.

B. Simulation

We apply Algorithm 1 to the disjoint product of the n agents P automaton, $\mathcal{T} = P_i \cup \cdots \cup P_n$. In order to generate trajectories from the given specifications we utilize a custom sequence planner that uses pre-selected trajectories based on a quadcopter's position and speed relative to a labeled location (e.g. an ellipsoid's location and generate splines between ellipsoids). After the initial trajectory for a given sequence is plotted, we use cross-entropy optimization to minimize that trajectory over the cost function $\mathcal{J} = \int_0^T \eta(\tau) + u(\tau) d\tau$.

The MTAC-E Algorithm samples trajectories from an unknown distribution that minimizes the cost function, $\mathcal{J}$, which is a function of the path length, $\eta(t)$ and the control input, $u(t) = \dddot{r}$ where $r = [x, y, z]^T \in \mathbb{R}^3$, the position of the center of mass of the robot.

Manuscript 1244 submitted to 2020 International Conference on Robotics and Automation (ICRA). Received September 15, 2019.

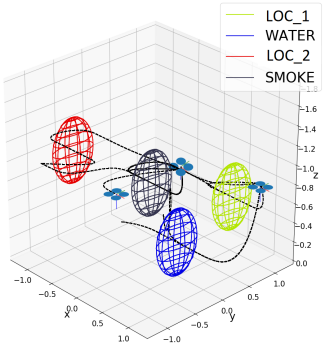

Fig. 4. Three quadcopters during a simulated fire fighting mission. The entire team is given the specification $\phi = \Diamond LOC_1 \wedge \Diamond LOC_2 \wedge \Box(SMOKE \implies CARRYING)$. Each quadcopter is considered during the iteration through the product automaton of the system, switches to another quadcopter are considered when the cost is beneficial for the team.

Results are shown in Fig. 4 where three quadcopters are shown satisfying the LTL formula ϕ. The results sequences are $quad_0 = \{WATER \quad SMOKE \quad LOC_2\}$, $quad_1 = \{SMOKE\}$ and $quad_2 = \{WATER \quad LOC_1\}$ which results in a satisfying sequence for the entire input specification.

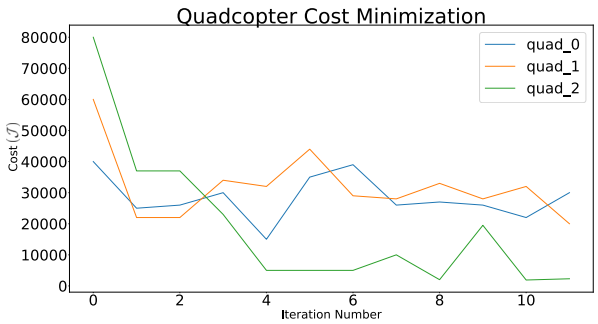

Fig. 5. The MTAC-E Algorithm iterates 12 times over a subset of trajectories and produces the trajectory with the lowest cost after all iterations. Here, we show the algorithm evaluating which quadcopter should transition to LOC_2. This calculation is formulated in our cost function where we minimize the distance traveled and input to system. Each quadcopter executes the MTAC-E optimization and after all quadcopters have completed the algorithm, the quadcopter with the lowest cost is selected to complete that task, in this example $quad_2$ is choosen.

C. Experimental Results

The MTAC-E Algorithm is implemented on the Robotarium at Georgia Tech where we use Crazyflie 2.0 quadcopters [25]. The Robotarium uses a Vicon Tracking system which records real-time position of robots with a 100 Hz update rate. The algorithm was created in Python and sends control inputs to a PID controller in C++. Commands are sent via ROS messages to Crazyflies and a radio operating in the 2400 MHz range with a data rate of 2 Mbit/s sends these commands to the quadcopters.

We use hoops with vertical stands to represent regions of interest, characterized by ellipsoids, as pictured in Fig. 6, and mark them with Vicon tracking points to record the center of the hoops and generate the proposition sets. In this experiment, we utilize three quadcopters and deploy them with the same LTL specification used in Section V-B. Based on the cost constraint $\mathcal{J}$, defined previously as functions of position and control input, tasks are assigned to agents based on proximity requirements to goal locations, control input constraints and prior tasks executed. Resulting from this problem, the quadcopters are allocated tasks in the following sequences $quad_0 = \{WATER \quad LOC_0 \quad SMOKE\}$, $quad_1 = \{SMOKE \quad WATER \quad LOC_2 \quad SMOKE\}$, and $quad_2 = \{WATER \quad SMOKE\}$. This experiment demonstrates the practical use of quadcopters in a real world scenario, delegating tasks to the agents in an optimal fashion using the MTAC-E algorithm.

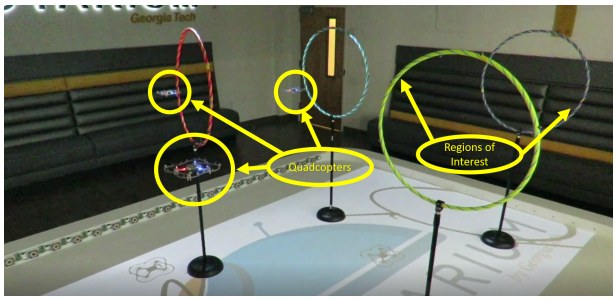

Fig. 6. In the experimental case study of the firefighting quadcopters, three quadcopters are chosen to execute the global task specification. The entire team is given the specification $\phi = \Diamond LOC_1 \wedge \Diamond LOC_2 \wedge \Box(SMOKE \implies CARRYING)$ and each are given a cost function to minimize. Regions of interest are represented as ellipsoids and hoops on stands are used in the experiment.

VI. Conclusion

In conclusion, we have developed a novel method for multi-agent task allocation using cross-entropy motivated by task switching for decomposed sequences of tasks. This method allows users to design global specifications to multi-agent systems where exact action costs for agents are not known to the user a priori but a known distribution can be approximated through a cost function. In addition, cost functions can be defined for individual agents depending on agent specific constraints. We show that this algorithm is scalable and flexible in system and environment constraint satisfaction through an operator chosen cost function. We showcase the efficacy of the algorithm both in simulation and in experiment under a scenario that demands satisfaction of environmental constraints and system constraints while optimizing a cost function designed to minimize individual agent trajectories and inputs.

References

[1] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *The International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004.

[2] G. A. Korsah, A. Stentz, and M. B. Dias, "A comprehensive taxonomy for multi-robot task allocation," *The International Journal of Robotics Research*, vol. 32, no. 12, pp. 1495–1512, 2013.

[3] A. Khamis, A. Hussein, and A. Elmogy, "Multi-robot task allocation: A review of the state-of-the-art," in *Cooperative Robots and Sensor Networks 2015*. Springer, 2015, pp. 31–51.

[4] M. Franceschelli, M. Egerstedt, and A. Giua, "Motion probes for fault detection and recovery in networked control systems," in *2008 American Control Conference*. IEEE, 2008, pp. 4358–4363.

[5] L. Luo, N. Chakraborty, and K. Sycara, "Distributed algorithm design for multi-robot task assignment with deadlines for tasks," in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 3007–3013.

[6] A. Kolling, P. Walker, N. Chakraborty, K. Sycara, and M. Lewis, "Human interaction with robot swarms: A survey," *IEEE Transactions on Human-Machine Systems*, vol. 46, no. 1, pp. 9–26, 2016.

[7] S. G. Loizou and K. J. Kyriakopoulos, "Automatic synthesis of multi-agent motion tasks based on ltl specifications," in *2004 43rd IEEE Conference on Decision and Control (CDC)(IEEE Cat. No. 04CH37601)*, vol. 1. IEEE, 2004, pp. 153–158.

[8] S. L. Smith, J. Tůmová, C. Belta, and D. Rus, "Optimal path planning for surveillance with temporal-logic constraints," *The International Journal of Robotics Research*, vol. 30, no. 14, pp. 1695–1708, 2011.

[9] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

[10] J. Tůmová and D. V. Dimarogonas, "Multi-agent planning under local LTL specifications and event-based synchronization," *Automatica*, vol. 70, pp. 239–248, 2016.

[11] S. G. Loizou and K. J. Kyriakopoulos, "Automated planning of motion tasks for multi-robot systems," in *Proceedings of the 44th IEEE Conference on Decision and Control*. IEEE, 2005, pp. 78–83.

[12] J. Chen, S. Moarref, and H. Kress-Gazit, "Verifiable control of robotic swarm from high-level specifications," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2018, pp. 568–576.

[13] P. Schillinger, M. Bürger, and D. V. Dimarogonas, "Improving multi-robot behavior using learning-based receding horizon task allocation," in *Robotics: Science and Systems (RSS)*, 2018.

[14] P. Schillinger, M. Burger, and D. V. Dimarogonas, "Decomposition of finite LTL specifications for efficient multi-agent planning," in *Distributed Autonomous Robotic Systems*. Springer, 2018, pp. 253–267.

[15] M. Kobilarov, "Cross-entropy motion planning," *The International Journal of Robotics Research*, vol. 31, no. 7, pp. 855–871, 2012.

[16] A. E. Gerevini, P. Haslum, D. Long, A. Saetti, and Y. Dimopoulos, "Deterministic planning in the fifth international planning competition: Pddl3 and experimental evaluation of the planners," *Artificial Intelligence*, vol. 173, no. 5-6, pp. 619–668, 2009.

[17] G. De Giacomo and M. Y. Vardi, "Linear temporal logic and linear dynamic logic on finite traces," in *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

[18] G. De Giacomo, R. De Masellis, and M. Montali, "Reasoning on LTL on finite traces: Insensitivity to infiniteness," in *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[19] G. J. Holzmann, *The SPIN model checker: Primer and reference manual*. Addison-Wesley Reading, 2004, vol. 1003.

[20] A. Bauer, M. Leucker, and C. Schallhart, "Runtime verification for LTL and TLTL," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 20, no. 4, p. 14, 2011.

[21] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[22] C. Baier and J.-P. Kateon, *Principles of Model Checking*. Cambridge, Massachusetts: The MIT Press, 2008.

[23] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2520–2525.

[24] L. Wang, E. A. Theodorou, and M. Egerstedt, "Safe learning of quadrotor dynamics using barrier certificates," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 2460–2465.

[25] B. AB. (2018). [Online]. Available: https://www.bitcraze.io/