LTL Cross Entropy Optimization for Quadcopter Task Orchestration

Christopher Banks [a] , Samuel Coogan [b] and Magnus Egerstedt [c]
[a] School of Interactive Computing, Georgia Institute of Technology, Atlanta, GA 30332, USA; [b] School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA; [c] Samueli School of Engineering, University of California-Irvine, Irvine, CA 92697, USA

ARTICLE HISTORY
Compiled July 28, 2022

ABSTRACT
This paper presents a task orchestration framework for multi-agent systems utilizing linear temporal logic (LTL) and cross entropy optimization, a stochastic optimization technique used for rare-event sampling. We define task orchestration as a combination of task decomposition, allocation and planning for a quadcopter or team of quadcopters given a high-level specification. Specifically, we consider tasks that are complex and consist of environment constraints, system constraints, or both, that must be satisfied. We first approach motion planning for the single agent case where transition systems for the environment allow tasks to be developed as linear temporal logic (LTL) specifications. Trajectories are then generated via motion primitives for a single quadcopter and optimized via cross entropy to ensure optimal satisfaction of a cost function. We extend this work to the multi-agent case where a team of homogeneous quadcopters are considered to satisfy an LTL specification. In order to provide faster computations and initial cost-agnostic sampling, we formulate the online version of multi-agent task allocation via cross entropy for tasks specified in LTL specifications. The results of this framework are verified in simulation and experimentally with a team of quadcopters.

KEYWORDS
cross-entropy optimization; multi-agent systems; linear temporal logic; motion planning; task allocation

1. Introduction

Interaction with multi-agent systems often involves users requiring the satisfaction of a set of complex tasks. These tasks are delegated to the multi-agent system for a wide variety of reasons including: autonomous surveillance [1], search and rescue tasks [2] and environmental monitoring [3]. Often these tasks are defined as individual and environment constraints imposed on the system [4], [5]. These constraints must then be satisfied by a single agent or a multi-agent system while a main objective is reached according to an objective function or performance index. In this paper, we formulate a system called *task orchestration*, defined as a composition of task decomposition, allocation and planning to provide an end-to-end framework for a team of quadcopters given constraints and an objective function. Specifically, users provide

CONTACT: Christopher Banks. Email: cbanks34@gatech.edu

system constraints and objectives as linear temporal logic (LTL) specifications [6] and tasks are decomposed using a task decomposition framework and allocated according to an objective function for each agent in the multi-agent system. Finally, trajectories associated with a set of assigned tasks are generated for each agent.

We experimentally validate the application of the task orchestration framework through a fire-fighting quadcopters scenario. Consider a set of N quadcopters, capable of carrying water, surveying various locations and identifying resources within a predefined area. How does one dynamically allocate these quadcopters to different regions, extinguish fires and monitor their internal states in an efficient manner? Using the task orchestration framework, users give a desired global goal for the team of quadcopters to satisfy; the framework then dynamically allocates tasks to each agent based on input cost and trajectory length, environment and agent constraints and plans trajectories for each agent. Experimentally, we define desired regions as hoops in the work space and generate an environment transition system to indicate how regions are connected. The internal state of a quadcopter is represented as the robot transition system. With these formulations, we consider LTL as the global specification syntax users provide the task orchestration framework.

Formal logic specifications, like LTL, provide an efficient and concise method for specifying and verifying correct behavior in dynamical systems and are well suited to human level interpretation and development due to its expressive syntax. High-level motion planning and task allocation using LTL allows for a diverse set of problems to be solved. In prior works, product automata consisting of environmental and agent systems are composed and satisfying sequences of states are found for robotic systems, defined as tasks, that satisfy that specification. These tasks are usually formulated as either motion primitives consisting of Lyapunov-based controllers [7], potential vector fields [8] or through symbolic control approaches like state space partitioning [9]. A work that is similar to ours, in the single agent case, is [10] where the authors proposed a sampling-based technique utilizing modified RRT* for agents to satisfy a global linear temporal logic specification. However, this work like many others relies on computationally expensive product automatons for large environments which are time consuming to generate [11] and dynamical constraints for physical systems are not considered. In addition, each transition between tasks in these transition systems is often associated with action costs that are defined based on expert information about the system dynamics and environment.

We address the issue of quadcopter high-level motion planning by creating motion primitives based on hoop/direction pairs [12]. These motion primitives leverage hand-picked control points around desired hoop locations in the state space and we compose trajectories from them. To avoid discretizing the state space and provide optimal trajectories we utilize cross-entropy optimization [13,14]. Cross-entropy optimization provides a method for stochastic optimization of trajectories by estimating rare-event probabilities (characterized as events that occur infrequently), which we associate with a desired cost and cost function. An alternative optimal sampling-based technique proposed in [15] leverages a probabilistically complete method of motion planning; however, it is not applied to multiple goal satisfaction nor does it address how it can be used for multiple agents. However, many prior works exist that utilize cross-entropy optimization for multi-agent task allocation applications. It has been used in solving problems like the max-cut problem [16], vehicle routing problem [17] and other problems for which dynamic allocation of resources can be formulated as minimizing according to a known desired cost and resources are drawn from a known distribution in order to estimate an unknown distribution [18],[19].

In addition to providing a framework for high-level motion planning in the single agent case, we also reduce the time consuming and resource intensive process of product automaton generation for the multi-agent case. We do this in two ways: we leverage an LTL decomposition framework and introduce an online method for cross-entropy optimization on trajectories sampled from multivariate distributions. The decomposition framework is from [20] and provides a theoretically sound method for decomposing a global LTL specification for an arbitrary number of agents. This bounds the size of the system product automaton to grow linearly with the number of agents as opposed to exponentially, as in most other works. We expand this work by introducing cross-entropy optimization which allows task delegation and switching to be based on the cost function of an individual agent. This not only removes the need for an expert dependent action cost assignment but also allows general agent cost constraints to be defined within each agent that can leverage cross-entropy optimization for multi-agent task allocation. In addition to this, the online multi-agent task allocation framework we propose provides fast calculations of optimal agent assignments and greatly reduces the task allocation computation time. One work that is close to ours is that of [21], where the authors use adaptive cross-entropy for task assignment for UAV formations, optimized over a global cost function. However, these authors do not consider path planning in their problem formulation and only consider task allocation for simulated vehicles.

Contributions

In this paper, we develop a framework for task orchestration in the single agent and multi-agent case and present three contributions. In the first contribution, we develop a hoop-sequencer algorithm that ensures a quadcopter flies to complete complex tasks where explicit sequences may be difficult to create. This is solved through the generation of LTL specifications and optimized via cross-entropy optimization.

The second contribution of this work presents a novel algorithm designed to sample trajectories and converge to a desired cost to determine the best robot from a team of robots for continued satisfaction of a goal specification and objective function. 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.

Our third contribution improves on the multi-agent task allocation algorithm by modifying our framework to consider online task allocation, providing fast updates to desired trajectories allocated to a team of robots. To the best of the authors' knowledge, this paper is the first to utilize cross-entropy optimization for both UAV task allocation and planning via the use of multivariate sampling distributions while leveraging high-level task assignments through linear temporal logic and presents experimental results validating the use of cross-entropy for UAV task allocation and trajectory planning. This novel approach to task allocation provides online trajectory sampling from a known distribution and iteratively updates based on the desired quantile of the multivariate distribution associated with a desired trajectory. By providing this online methodology we combine the reduction in product automata size from task decomposition with an expert independent framework for choosing optimal costs and a fast way to dynamically allocate tasks to a set of agents which scales linearly.

This paper extends the work from [12] and [19] by significantly expanding the capabilities of the previously developed multi-agent task allocation algorithm via the online cross-entropy optimization method developed in this paper. In particular, we present a novel characterization of online cross-entropy over multivariate distributions and develop a general, task orchestration framework that utilizes this stochastic optimization methodology for online task assignments to individual agents in a multi-agent system. We compare this online result to the prior offline work, and empirically show a significant decrease in overall cost and execution time. In addition to this, we validate the online approach for task orchestration of multi-agent systems on a particular fire-fighting quadcopters scenario.

The paper proceeds as follows. In Section 2 we review preliminaries and notation used throughout the paper. In Section 3 we introduce the hoop-sequencer planner and optimization framework for the planner. In Section 4 and 5 we introduce the MTAC-E algorithm and provide a case study. In Section 6 we improve on the MTAC-E framework and provide an online version of the stochastic optimization technique followed by experimental results comparing the online and offline versions. We provide experimental results for the online MTAC-E framework in Section 7 followed by the conclusion in Section 8.

2. Preliminaries

In this section we introduce the preliminaries that will be used throughout the paper.

First, we give a brief introduction of finite LTL, which we use as a method of generating high-level specifications for discrete autonomous systems. We optimize our frameworks throughout this paper with the stochastic optimization technique, cross-entropy, allowing us to sample from a subset of nominal trajectories that satisfy an optimal cost for a system of agents. Following this, we show how we utilize quadcopters as a highly maneuverable and versatile robotic platform to use for task orchestration frameworks and introduce quadcopter dynamics and the controller used throughout this paper.

2.1. Finite LTL for Global Task Specification

In this section, we provide a brief background on finite LTL, a class of LTL specifications well-suited for formally representing planning problems [22] and interpreting finite sequences [23]. LTL specifications ϕ are defined as logic formalisms suited for specifying linear time properties [24,25]. LTL specifications are defined over traces and indicate repeated satisfaction of a set of propositions Π defined in the following definition.

Definition 2.1. 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?").

In order to create a formulation for decomposing, we consider finite LTL specifications. Finite LTL specifications are insensitive to infiniteness [26]. This restriction applied to LTL specifications means we can reduce LTL formulas to finite LTL formulas by including a unique *end* proposition into the set of all propositions (2^π) for the formula. In order for a formula to be insensitive to infiniteness the following can be

shown: i) inclusion of this *end* proposition, ii) *end* must hold eventually, iii) *end* must remain true for all time, and iv) when *end* is true, all other propositions are trivially set to false. Specifications are defined over sequences of observations and the notation $\sigma \models \phi$ indicates that the sequence σ satisfies ϕ. Finite LTL includes the co-safe LTL [27] class where a sequence σ contains "good" prefixes such that $\sigma_f = w_0, w_1, \ldots, w_n$ is a satisfying truncated finite sequence and an infinite sequence σ_ω with propositions in the set $\sigma_\omega \in 2^\Pi$ where every proposition in σ_ω evaluates to true ($\top$) for the specification. The entire satisfying sequence is then a composition of σ_f and σ_ω where $\sigma = \sigma_f \sigma_\omega$. Another class of finite LTL specifications that is utilized contains a similar prefix-suffix structure for sequences except the sequence σ is defined as $\sigma = \sigma_f \sigma_N$ where σ_N is a N repeating sequence of propositions that satisfies the LTL specification for the duration of the algorithm. Any finite LTL specification can be represented via a non-deterministic finite automaton (NFA) [20], which we define below.

Definition 2.2. 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 $q = q(0) \ldots q(T) \in Q$ a sequence σ – defined as a sequence of propositions π_i from Π – satisfies ϕ if it enables a transition from $q(0)$, an initial state, to $q(T) \in F$. These transitions, 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.

2.2. Cross Entropy

Cross-entropy optimization is a method of importance sampling for probabilistically rare events. The algorithm design for using cross-entropy with motion planning [13] can be generalized as the following:

(1) Generate a set of sample trajectories ($\mathcal{X}$) 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 $0.1 \leq \rho < 0.3$. This subset is known as an *elite set* and provides a new sampling space to generate the distribution p. In this work, we sample trajectories according to a multivariate 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 covariance matrices, $\Sigma = [\Sigma_0, \ldots, \Sigma_n]^T$ form an $nm \times m$ matrix with Σ_i initially set to the identity matrix, $\mathcal{I}$. Expectation-Maximization [28] is used to update the means and covariances for the newly sampled trajectories.

2.3. Quadcopter Dynamics

We present here the dynamics considered for a quadcopter (shown in Fig. 1) and how trajectories are generated via spline interpolation to achieve proposition satisfaction. The rotation matrix, $R(\epsilon)$, that translates between the world frame (F_w) and body

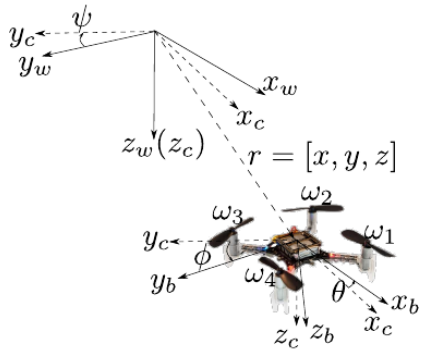


Figure 1. A quadcopter with respect to the world frame (F_w), intermediate frame (F_c) and body frame (F_b). Four motors ($\omega_{1:4}$) produce torques and thrust for the system.

frame (F_b) is given by

$$R(\epsilon) = \begin{bmatrix} c\theta c\psi & s\phi s\theta c\psi - c\phi s\psi & c\phi s\theta c\psi + s\phi s\psi \\ c\theta s\psi & s\phi s\theta s\psi + c\phi c\psi & c\phi s\theta s\psi - s\phi c\psi \\ -s\theta & s\phi c\theta & c\phi c\theta \end{bmatrix} \tag{1}$$

where $s\theta$ and $c\theta$ stand for $\sin(\theta)$ and $\cos(\theta)$, respectively. The angles θ, ψ, and ϕ are the angles between the axes of the quadcopter in the body frame and the axes of the world frame. The input (μ) to the system consists of $\mu = [f_z, \omega_{bw}^T]^T$ with f_z as the thrust and $\omega_{bw} = [\omega_x, \omega_y, \omega_z]^T$ as the body rotational rates of the quadcopter. We use the non-linear quadcopter model from [29] to describe the dynamics that generate trajectories for quadcopters:

$$\ddot{r} = g z_w + \frac{1}{m} R(\epsilon) z_w f_z \tag{2}$$

$$\dot{\epsilon} = \lambda(\epsilon)\omega_{bw} = \begin{bmatrix} 1 & s\phi t\theta & c\phi t\theta \\ 0 & c\phi & -s\phi \\ 0 & s\phi sc\theta & c\phi sc\theta \end{bmatrix} \omega_{bw}, \tag{3}$$

where $z_w = [0\,0\,1]^T$ is the z-direction vector for force in F_w and $sc\theta$ and $t\theta$ are $\sec(\theta)$ and $\tan(\theta)$, respectively. The position of the center of mass (r) in the world frame (F_w) is $r = [r_x, r_y, r_z]^T$; m, and g are the mass and acceleration of gravity, respectively. We represent the Euler angles as $\epsilon = [\phi, \theta, \psi]^T$, and $\Gamma(\epsilon)$ is the transformation matrix from body rotational rates in F_b to Euler angles in F_w. These dynamics give the 9-dimensional state (ξ) of the quadcopters where $\xi = [r^T, \dot{r}^T, \theta, \phi, \psi]^T$ and input (μ).

Utilizing the differentially flat dynamics [30] of the quadcopters, we plan trajectories in the flat output space 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.

For a multi-agent system $\mathcal{M} = \{1, \ldots, N\}$, we control the dynamics of agent i through the following chain of integrators:

$$\dot{p}_i = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}_{4\times 4} \otimes \mathbb{I}_{3\times 3} \cdot p_i + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \cdot u \tag{4}$$

with virtual input $u \in \mathbb{R}^3$ and state $p = [r^T, \dot{r}^T, \ddot{r}^T, \dddot{r}^T]^T \in \mathbb{R}^{12}$ where $r = [x, y, z]^T$. We control this linear system using feedback control with

$$u = -\langle \mathbf{K}, p - \eta(t) \rangle \tag{5}$$

given a desired $\mathbf{K} \in \mathbb{R}^{1\times 4}$, state $p = [r, \dot{r}, \ddot{r}, \dddot{r}]^T$, and desired trajectory $\eta(t) = [r_d, \dot{r}_d, \ddot{r}_d, \dddot{r}_d]^T$ and drive the linear system to the desired trajectory and derive the feedforward control inputs $\mu_{ff} = [f_{z,ff}, \omega_{bw,ff}^T]^T$ from differential flatness and generate feedback control inputs $\mu_{fb} = [f_{z,fb}, \omega_{bw,fb}^T]^T$ from a PID control loop [12] to control an individual quadcopter with control input $\mu = \mu_{ff} + \mu_{fb}$.

3. Specification Based Planning of a Quadcopter

Given a quadcopter with dynamics described in Section 2.3, in prior work, we developed a fully autonomous planner capable of navigating through suspended hoops [12]. Using this planner we define a hoop in $\mathbb{R}^3$, with the following definition:

Definition 3.1. We define a hoop set $\mathcal{H}=\{\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_n\}$ according to:

$$E_j(r) = \frac{(r_x - x_j)^2}{a^2} + \frac{(r_y - y_j)^2}{b^2} + \frac{(r_z - z_j)^2}{c^2} \tag{6}$$

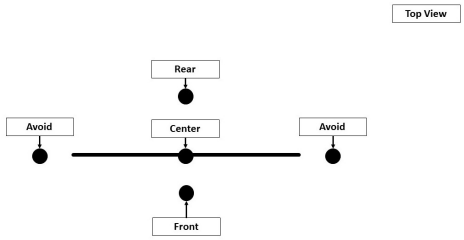
$$\mathcal{H}_i = \{r \in \mathbb{R}^3 \quad | \quad E_j(r) \leq 1\} \tag{7}$$

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 (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.

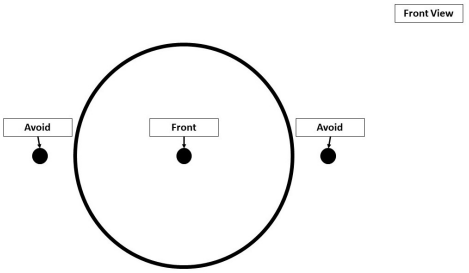
Additionally, each hoop contains five control points (two avoid points, front, rear, and center) in $\mathbb{R}^3$, as illustrated in Fig. 2. We leverage these control points as anchor points defining locations of interest near hoops and use them to develop *motion primitives* between hoops. These motion primitives are pre-defined trajectories and are chosen based on a quadcopters position and direction according to the desired sequence of hoops. A motion plan, formulated as an LTL specification, is provided to the hoop-sequencer planner.

3.1. Finding Satisfying Sequences Given an LTL Specification

We consider the scenario of users providing high-level system requirements in the form of LTL specifications through the following problem formulation.



(a) A top-down view of the orientation of control points for a hoop where the straight line depicted represents a hoop as seen from above. The five labeled dots represent the five control points (two avoid points, rear, center, and front) for each hoop.



(b) A front view of the hoop, depicted as a circle in the above image, with the five control points (two avoid points, rear, center and front). Note, the center and rear control points are occluded in this view.

Figure 2. The orientation of the control points around an example hoop. We define five control points as points in $\mathbb{R}^3$ at desired positions around hoops which are used as anchor points for trajectories generated between them.

Problem 3.1. Given a quadcopter and a set of hoops labeled by propositions, design a trajectory that satisfies a given LTL specification. For example, consider the following specification: "always ensure flying through $hoop_0$ implies $hoop_2$ is flown through before $hoop_1$ and eventually reach $hoop_1$". This specification can be represented by the LTL formula $\phi = \square(\lozenge hoop_0 \rightarrow \neg hoop_1 \cup hoop_2 \wedge \lozenge hoop_1)$.

In this section, we define transitions between LTL propositions and continuous space and develop a hoop-sequencer planner to accommodate LTL specifications.

Hoops ($hoop_i$) are defined as atomic propositions according to Definition 2.1 such that the proposition set is $\Pi = \{hoop_0, hoop_1, \ldots, hoop_n\}$. We check that propositions $hoop_i$ are satisfied by mapping the hoop set ($\mathcal{H}_i$) in Definition 7, to the propositions through the labeling function k, where $k(r) = \{hoop_i \in \Pi : r \in \mathcal{H}_i\}$. In other words, we map the position of a quadcopter ($r \in \mathbb{R}^3$) to a set of corresponding hoop propositions. For example, $k(r) = \{hoop_0\}$ iff r belongs to $\mathcal{H}_0$. For continuous quadcopter trajectories $\eta_c(t)$, we use a slight overload of notation for the following definition.

Definition 3.2. Let us define the labeling function k over a continuous trajectory where $k(\eta_c) = \{hoop_{i_1}, hoop_{i_2}, hoop_{i_3}, \ldots, hoop_{i_j}\}$ is the sequence of hoops visited by a quadcopter and $hoop_i \in \Pi$ and j indicates the j^{th} hoop in the sequence.

This labeling function generates a sequence of propositions from the continuous trajectory $\eta_c(t)$ for $t \geq 0$. For example, a sequence could have the form $k(\eta_c) = \{hoop_0, hoop_1, hoop_0\}$. If the continuous trajectory $\eta_c(t)$ is created such that all propositions ($hoop_i$) generated from the trajectory satisfy an LTL formula ζ, then we have successfully found a trajectory that satisfies a given LTL formula. Therefore, the trajectory η_c satisfies the LTL formula ζ iff $k(\eta_c) \models \zeta$.

After defining discrete propositions in continuous space, we can plan trajectories in the discrete space and map the trajectories back into the continuous domain using our hoop-sequencer planner. From the LTL specifications, we generate a NFA as a directed graph. A search through this graph for a run $q = q_0 q_1 \ldots q_N$ results in a satisfying run if $q_N \in F$, or q_N is in the set of accepting final states of the NFA. Associated with run q is a sequence of propositions ($\pi \in \Pi$) such that a word $\sigma = \pi_0, \pi_1, \cdots \in \Pi$ is accepted if there is an accepting run for σ. Once a satisfying run has been found, the satisfying word (i.e., a sequence of hoops) has been found and we apply our hoop-sequencer planner to find a continuous trajectory.

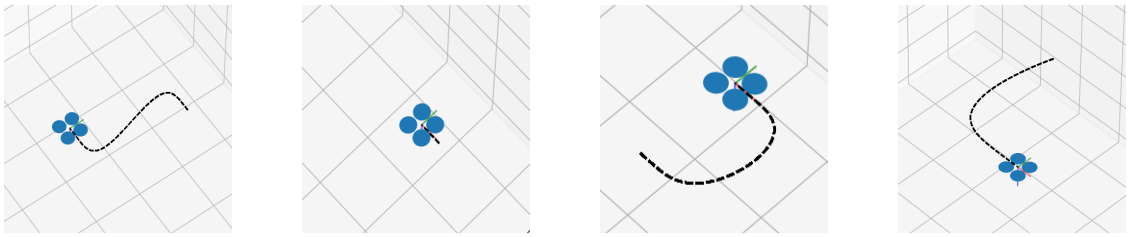
3.2. LTL Hoop-Sequencer Planner

Solution 3.1. In this section, we describe our hoop-sequencer trajectory planner in Algorithm 1. The algorithm receives as input the current pose of the robot (r), a set of n hoops ($\mathcal{H}$) (each defined by the control points previously mentioned), the input LTL specification (ϕ), and the previous control points visited by the quadcopter ($c_{1:m}$). These control points are used to generate four distinct curves based on the distance of the current pose and future control points along with the direction of the quadcopter and generates segments ($segment$) via spline interpolation that joins control points together.

We first define the current (pos_curr) and previous (pos_prev) pose which are found based on the order of the input sequence. In line 3, the past direction (dir_{past}) of the

last control point transition is recovered from the list of previous control points. We introduce a function `LTL_TO_SEQUENCE` in line 4 which receives an LTL specification, generates an equivalent NFA and searches for a satisfying sequence. In order to determine which direction (front or rear) the quadcopter must fly through the hoop, we use the Euclidean distance between each proposed hoop to find the closest control point between two consecutive hoops in the sequence. The corresponding directions are appended to the hoops in the sequence and the planner executes as before. From the LTL specification $\phi = \square(\lozenge hoop_0 \rightarrow \neg hoop_1 \cup hoop_2 \wedge \lozenge hoop_1)$, we get the sequence: $\Sigma_{prop} = \mathcal{G} = (hoop_1)(hoop_2)^N$ in prefix-suffix form. We note that Σ_{prop} is used to denote the accepting word for the NFA generated from our sample LTL specification and N indicates the hoop, or set of hoops, that can be visited N times during the execution of the algorithm. The resulting trajectory is shown in Fig. 4.

In lines 6 - 8 we define the next control point (pos_next) and the distance vector ($dist_{curr,next}$) to the next control point. In line 6, the function `NEXT_CONTROL_POINT` uses the current sequence to determine which control point on a labeled hoop is next. The function `DIRECTION` in Line 8 gets the direction that the quadcopter is heading. This will inform our algorithm which type of trajectory, out of four pre-determined curves, will be chosen for a particular path segment. Starting at Line 9, hoops that are in the state space but not in the input sequence ($\mathcal{G}$) are avoided using the `AVOID` function. The `AVOID` function receives as input the current position, a hoop from the list of hoops and the desired next position. If a hoop is between these two points, the function will choose the closest avoid control point of a particular hoop without crossing through that hoop and that resulting *segment* from the avoid control point to the current position will be added to the total trajectory $\eta(t)$.



(a) `S_CURVE`: an arc produced when $dist_x > dist_{yz}$ and quadcopter is not changing direction

(b) `STRAIGHT`: a straight line curve generated in the x-direction when $dist_{yz}$ is zero

(c) `U_TURN`: curve produced when quadcopter is changing direction

(d) `TURN`: curve produced when $dist_x \geq dist_{yz}$ and quadcopter is not changing direction

Figure 3. Trajectory segments that are generated in the hoop-sequencer planner.

Characterizing a Trajectory Segment: For the four types of trajectories that can be generated, all are parameterized by time and are described by our algorithm in lines 15 - 25. We describe each function in detail here.

`STRAIGHT`: If $dist_{yz} = 0$, The function `STRAIGHT` in line 15 generates a trajectory segment length $||\eta_s(t)|| = |r_x^c - r_x^w|$ where r_x^c is the $x - component$ of the current pose and r_x^w indicates the $x - component$ of the previous pose.

`U_TURN`: A trajectory the length of two quarter arcs joined by a straight segment is returned by the `U_TURN` function in Line 19 where $||\eta_s(t)|| = (dist_{yz} - dist_x) + (dist_x \cdot \frac{\pi}{2})$.

`S_CURVE`: In line 21, the length of the trajectory segment is created by first defining a right triangle leg (rt), such that $rt = \delta\sqrt{dist_x^2 - dist_{yz}^2}$. Then, θ and the hypotenuse (h) of the triangle are defined as $\theta = atan2(dist_{yz}, dist_x)$ and $h = \frac{rt}{\sin\theta}$, respectively.

Algorithm 1: Hoop-Sequencer Planner

input : pose of quadcopter r, hoops $\mathcal{H}$, LTL formula ϕ, control_points_prev $c_{1:m}$
output: trajectory $\eta(t)$

$pos\_curr \leftarrow r$
$pos\_prev \leftarrow c_m$
$dir_{past} \leftarrow \texttt{DIRECTION}(pos\_curr, pos\_prev)$
$\mathcal{G} = \texttt{LTL\_TO\_SEQUENCE}(\phi)$
for $i \leftarrow 0$ **to** $len(\mathcal{G})$ **do**
    $pos\_next \leftarrow \texttt{NEXT\_CONTROL\_POINT}(\mathcal{G}_i)$
    $dist_{curr,next} \leftarrow \texttt{DISTANCE}(pos\_curr, pos\_next)$
    $dir_{curr,next} \leftarrow \texttt{DIRECTION}(pos\_curr, pos\_next)$
    for *hoop in* $\mathcal{H}$ **do**
        if *pos_next != hoop* **then**
            $segment \leftarrow \texttt{AVOID}(pos\_curr, hoop, pos\_next)$
        end
    end
    if $dist^{yz}_{curr,next} == 0$ **then**
        $segment \leftarrow \texttt{STRAIGHT}(pos\_curr, pos\_prev)$
    end
    if $dir_{curr,next} == dir_{past}$ **then**
        if $dist^{x}_{curr,next} \leq dist^{yz}_{curr,next}$ **then**
            $segment \leftarrow \texttt{U\_TURN}(dist_{curr,next})$
        else
            $segment \leftarrow \texttt{S\_CURVE}(dist_{curr,next})$
        end
    else
        $segment \leftarrow \texttt{TURN}(dist_{curr,next})$
    end
    $pos\_prev \leftarrow pos\_curr$
    $pos\_curr \leftarrow \texttt{RETURN\_CONTROL\_POINT}(segment)$
    $\eta(t) = \eta(t) + segment$
end
return $\eta(t)$

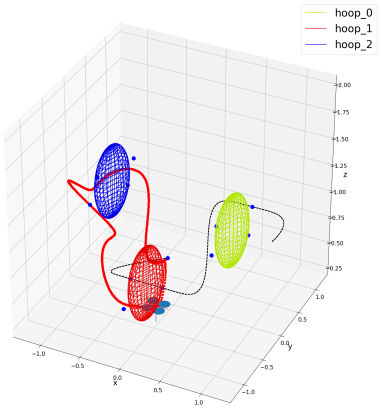


Figure 4. A satisfying run from the LTL specification $\phi = \square(\lozenge hoop_0 \rightarrow \neg hoop_1 \cup hoop_2 \wedge \lozenge f hoop_1)$. We show the prefix portion of the trajectory in black and the suffix portion in red. The suffix portion indicates the set of hoops that can be visited infinitely often.

The total length of the trajectory results in $||n_s(t)|| = 2(2h\theta)$ which generates two equal length tangent arcs or an "s-curve" in function `S_CURVE`.

`TURN`: If the direction is not the same as last, the trajectory length becomes a straight segment followed by a half-circle that changes the direction of the quadcopter such that $||\eta_s(t)|| = dist_x + (dist_y \cdot \frac{\pi}{2})$, the function `TURN` generates this segment.

In Fig. 3 we show each type of trajectory segment that our planner can generate. The path lengths are then transformed into time segments via $t_s = \frac{||\eta_s(t)||}{v}$, where v is the desired speed. These time segments (t_s) along with the waypoints $(w_{1:m})$ given by the input sequence are used to generate smooth trajectories $(\eta(t))$ via spline interpolation.

3.3. Optimizing Sequences Using Cross-Entropy

Although the hoop-sequencer planner is able to generate trajectories from user provided LTL specifications, the path is generally not optimal. This is due to the fact that the motion primitives used to generate trajectories are only constrained to satisfy continuity requirements. By constraining our trajectories with a cost function, we can generate trajectories that not only satisfy the primary objective of the user but secondary objectives of the system (e.g. minimize fuel cost, average velocity, etc.). We draw from the following scenario to motivate our problem:

Problem 3.2. Given a hoop-sequencer planner for quadcopters, utilize the cross-entropy method to return trajectories that satisfy the sequence as well as minimize cost. Use this method to guarantee an optimal trajectory with respect to a cost function $\mathcal{J}(r, u)$, parameterized by the robot pose, r, and its control input u.

Solution 3.2. We use the cross-entropy method as a stochastic optimization technique for choosing trajectories according to our cost function, which we define to minimize the total length of the trajectory, i.e., $\mathcal{J}(r, u) = ||\eta(\cdot)||$.

Optimizing the Hoop-Sequencer Planner: From the augmented hoop-sequencer planner, we apply cross-entropy optimization to reduce the cost of the sampled trajectories once they are generated. Our algorithm is adapted from [18] with modifications on sampling initial means. We sample from the hoop-sequencer planner to generate initial means to ensure that only the subset of the state space relevant to

our hoop sequence is sampled.

```
1  input : LTL formula φ, hoop_propositions i → n H_{i:n}, number of trajectories T,
           optimal cost Σ, elite set modifier ρ, sampling distribution p(μ_0, v), iteration
           number N
   output: best_path η(t)
2  n := initial iteration number
3  μ_0 = Hoop-Sequencer Planner(r, H_{i:n}, φ, w_{1:m})
4  best_cost := ∞
5  while best_cost > Σ and n < N do
6      for i in T do
7          path_samples → p(·, v)
8          η(t) → PATH(path_samples)
9          path_check → LTL_PARSER(φ, η(t), H_{i:n})
10         if path_check==True then
11             sorted_trajectories → η_1(t) < η_2(t) ··· < η_p(t)
12             best_cost = sorted_trajectories_0
13             elite_set = ρ * sorted_trajectories
14             p(·, v) → UPDATE(sorted_trajectories)
15 return η(t)
```

Path samples are sampled from a multivariate Gaussian distribution. The means (μ_0) are initialized to be n equidistant samples from the augmented hoop-sequencer planner. As a result, we get path samples from line 7. Line 8 defines a `PATH` function that receives as input samples from the distribution $p(\mu_0, v)$ and generates a trajectory via spline interpolation. This trajectory is then checked in line 9 where it is monitored for inclusion in the hoop set ($\mathcal{H}$). The trajectory is then parsed in the syntax of the accepting set of hoops and checked for whether it satisfies the string received initially from the augmented planner generated from the LTL formula (ϕ). If this check returns $True$ the trajectory is returned, otherwise a new trajectory is sampled. Trajectories are collected and sorted from best cost to worst cost and an elite set is chosen corresponding to a subset of trajectories (T). We then update the probability distribution using the elite subset of trajectories.

Example 1. Using the cross-entropy hoop-sequencer planner, we optimize over the LTL formula $\phi = \square(\lozenge hoop_0 \rightarrow \neg hoop_1 \cup hoop_2 \wedge \lozenge hoop_1)$ provided in Section 3.1. In Fig. 6, we show the optimized trajectory of the LTL formula with the constraint that *each* trajectory segment (prefix and suffix) length should be less than 5 meters or $\mathcal{J}(r, u) \leq 5$. In Fig. 5 we show the sampled paths over an iteration of the algorithm.

4. Multi-Agent Task Allocation via Cross Entropy

In the previous section, we presented a planner capable of utilizing LTL to delegate high-level user specifications to a quadcopter. Through the introduction of motion primitives we simplify the path planning problem and introduce cross-entropy to optimize trajectory costs over a desired cost function. However, for large LTL specifications using a single agent may become infeasible. By using multiple agents, we can scale up the number of tasks a group of robots can perform and reduce the total time required to complete the tasks. Therefore, in this section we propose to use LTL as global task

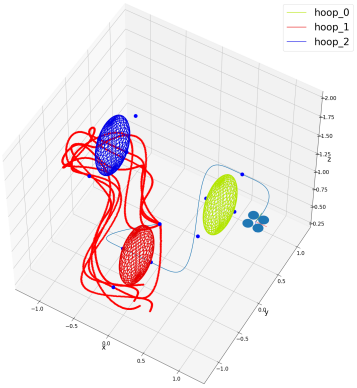


Figure 5. Here we show a series of trajectories generated from Algorithm 1. The prefix portion of the trajectory, represented in blue, satisfied the cost function $\mathcal{J} = ||\eta(t)||$ initially therefore, only one sample was needed. Several samples of the suffix portion, in red, of the trajectory were sampled before a satisfying trajectory was found.

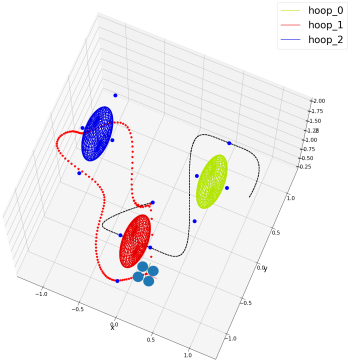


Figure 6. A simulation snapshot of the LTL cross-entropy hoop-sequencer planner after a satisfying run. The optimized trajectory of is shown here where each trajectory segment must satisfy the cost constraint $\mathcal{J}(r, u) \leq 5$.

specifications, allowing users to design global goals for multi-agent team execution. In addition to this, global goals enable scalability (i.e. goals are independent of the team size) and reduce cognitive load [31] 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 will give a brief overview of the transition systems used followed by an LTL decomposition framework that will allow for the decomposition of LTL specifications to a set of N agents.

4.1. Defining Transition Systems

We adopt the framework for defining task decomposition from [20], which 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 4.1. 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 a robot and transitions are based on the actions, $A_\mathcal{R}$, available to the robot at each state. For example, consider a simple robot transition system with two states $S = \{charging, not\_charging\}$. We can define a sequence of state action pairs $s = s_0, a_0, s_1$ to transition from the non-charging state (s_0) to the charging state (s_1) via action a_0, and vice versa. We next define the environment transition system $\mathcal{E}$ to capture the properties of the regions of interest for the agents.

Definition 4.2. 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
- $\Lambda_\mathcal{E} : V_\mathcal{E} \to 2^{\Pi_\varepsilon}$ is a labeling function that assigns atomic propositions to locations.

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

Definition 4.3. 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:

- $S_\mathcal{A} = V_\mathcal{E} \times S_\mathcal{R}$ is the set of 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
- $A_\mathcal{A} \subseteq S_\mathcal{A} \times S_\mathcal{A}$ is the set of 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 some 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.

Definition 4.4. The planning automaton $\mathcal{P}$ is a product automaton of an NFA $\mathcal{F}$, generated from LTL specification ϕ, and agent transition system $\mathcal{A}$ 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.

4.2. Decomposition Set

Given a multi-agent system with N agents, according to the set of automata $\mathcal{P}^{i,\dots,N}$, 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 $s^i = s_0 a_0, \dots, a_n s_n$. We give the following definition of finite LTL task decomposition.

Definition 4.5. [20] Let $\mathcal{T}_i$ with $i \in \{1, \dots, n\}$ 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:

$$\sigma_{j_1} \dots \sigma_{j_i} \dots \sigma_{j_n} \models \phi \tag{8}$$

for all permutations of $j_i \in \{1, \dots, n\}$ 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 [20].

We use this decomposition property 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 4.6. The team model automaton $\mathcal{T}$ is a union of the N local planning automata $\mathcal{P}^i$ 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:

- $S_\mathcal{T} = \{(r, q, s) : r \in \{1, \ldots, N\}, (q, s) \in S^i_\mathcal{P}\}$ 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
- $A_\mathcal{T} = \bigcup_i A^i_\mathcal{P} \cup \zeta$ 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 4.7. 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 [20]:

- $r_s \neq r_t$: 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
- $s_t = s^{r_t}_{0,\mathcal{A}}$: 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.

With discrete transition systems defined for a homogeneous team of agents, a decomposition framework, and a method to generate trajectories for a single agent we turn to the following problem formulation.

Problem 4.1. For a given set of homogeneous 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 fire-fighting quadcopter scenario in Section 5.3. 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 team 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.

5. MTAC-E Algorithm

Solution 5.1. 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 4.2; given we have designed cost functions for each agent in the problem, we need a way to find optimal trajectories by minimizing nominal trajectories via these cost functions. To find the associated minimized costs, we propose using cross-entropy optimization. In this framework, we use agent trajectory costs 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 unknown. Next, we present the multi-agent task

allocation cross-entropy (MTAC-E) algorithm followed by a complexity analysis of the algorithm and the case study mentioned in the problem statement.

5.1. Algorithm

The algorithm developed in this paper, provided in pseudocode format in Algorithm 2, 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 total 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 team 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 4.6 of the team product automaton in this framework such that via a standard breadth-first 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 switch transition occurs. By switching the agents at decomposition states, this algorithm optimizes the individual task function of each agent via cross-entropy and optimally allocates tasks to minimize the total cost of an individual agent. This algorithm will return a set of trajectories $\mathcal{N}$ with each agents individual trajectory $\eta_i(t)$.

5.2. 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 [6]. 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 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

Algorithm 2: 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 $\mathcal{N}$

$p_i :=$ initial_state($\mathcal{T}$)
$\alpha_i \to p_i :=$ agent i in initial state
$sequences_i :=$ sequence of states visited by agent i
$p \to p_i :=$ set p to initial state
while $p \notin$ *final_states*($\mathcal{T}$) **do**
  for q *in neighbors*(p) **do**
    $\eta_i(t)$, $cost_i \to$ `cost_to_go`(α_i, q, $sequences_i$, λ, $p(\cdot, v)$, K, ρ)
    if $q \in \mathcal{D}$ **then**
      $\eta_{j:n}(t)$, $cost_{j:n} \to$ `cost_to_go`($\alpha_{j:n}$, q, $sequences_{j:n}$,λ, $p(\cdot, v)$, K, ρ)
    end
    if $cost_{j:n} < cost_i$ **then**
      $\alpha_i \to \alpha_j$
      $p \to p_j$
      $sequences_j = sequences_j + sequences_{j,p\to q}$
    end
    else
      $sequences_i = sequences_i + sequences_{i,p\to q}$
    end
  end
end
$\mathcal{N} = \{\eta_1(t), \ldots, \eta_n(t)\}$
return $\mathcal{N}$

N agents in our framework, our algorithm can check trajectories with 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.

5.3. Case Study: Fire Fighting Quadcopters (Introduction)

We motivate the application of Algorithm 2 with a firefighting quadcopters scenario. This scenario naturally fits within a discrete planning framework for a multi-agent system due to multiple environment constraints that need to be satisfied within a defined area (e.g. verifying safe regions, checking for water sources, etc.). In addition to this, agents may have internal constraints that need to be satisfied that can be developed as the internal transition system of an agent. The MTAC-E algorithm given a global goal, a finite automata describing the operational environment and individual internal state for a team of agents, optimally plans trajectories for a set of agents given the following problem definition.

Example 2. For example, each agent may be a fire-fighting autonomous aircraft capable of collecting water, extinguishing fires and surveying goal locations. These agents 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 $\phi = \Diamond LOC_1 \wedge \Diamond LOC_2 \wedge \Box(SMOKE \implies CARRYING)$.

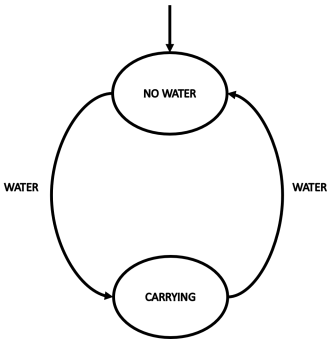


Figure 7. A transition system for a single agent which describes the internal state of a robot ($\mathcal{R}_i$). All robots start at the initial state 'NO WATER' and a location-based transition is used to determine when to transition to the 'CARRYING' state. If a robot is in the environment state that 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 4.1 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. 7 we represent the discrete internal transition system of robot i as $(\mathcal{R}_i)$.

The environment transition system, Fig. 8, 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 $\mathbb{R}^3$, defined in Section 2.

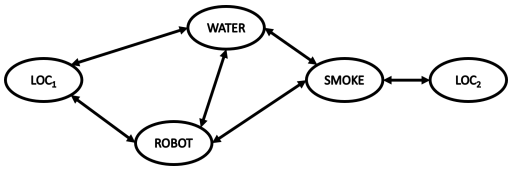


Figure 8. The environment transition system where each state indicates a desired region of interest. The initial state of the environment is the 'ROBOT' state. In the fire fighting example, LOC_2, cannot be reached unless the quadcopter passes through the SMOKE region.

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. 9. 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.

5.4. Simulation

We apply Algorithm 2 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

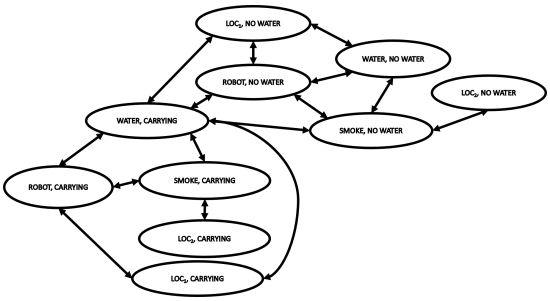


Figure 9. The full agent transition system for a quadcopter. Transitions to the 'CARRYING' state can only be fulfilled once the agent has retrieved water from the environment node.

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, $u(t) = \ddot{r}$, where $r = [x, y, z]^T \in \mathbb{R}^3$, the position of the center of mass of the robot. We set $\lambda = 0$, indicating a desired optimal cost for each agent of minimal trajectory length and cost. In general, a trade-off is made between picking a reasonable λ and algorithm run-time, which is why limiting the number of iterations is desirable.

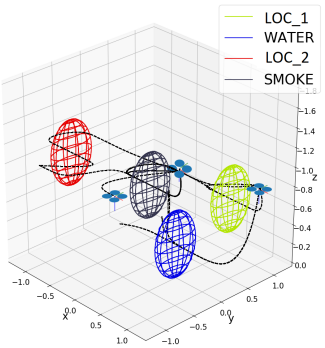


Figure 10. Three quadcopters during a simulated fire fighting mission. The entire team is given the specification $\phi = \Diamond LOC_1 \wedge \Diamond LOC_2 \wedge \Box(SMOKE \implies CARRYING)$. Each quadcopter is considered during the iteration through the product automaton of the system, switches to another quadcopter are considered when the cost is beneficial for the team.

Results are shown in Fig. 10 where three quadcopters are shown satisfying the LTL formula ϕ. The resulting 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. The returned sequences are one of many satisfying sequences returned by a search over the team automaton, $\mathcal{T}$, and additional constraints in the graph can modify which sequences are returned. In Fig. 11, the cost per quadcopter over 12 iterations is shown for satisfying a single proposition (LOC_2). In this example, the MTAC-E algorithm selects $quad_2$ as the quadcopter to transition to LOC_2.

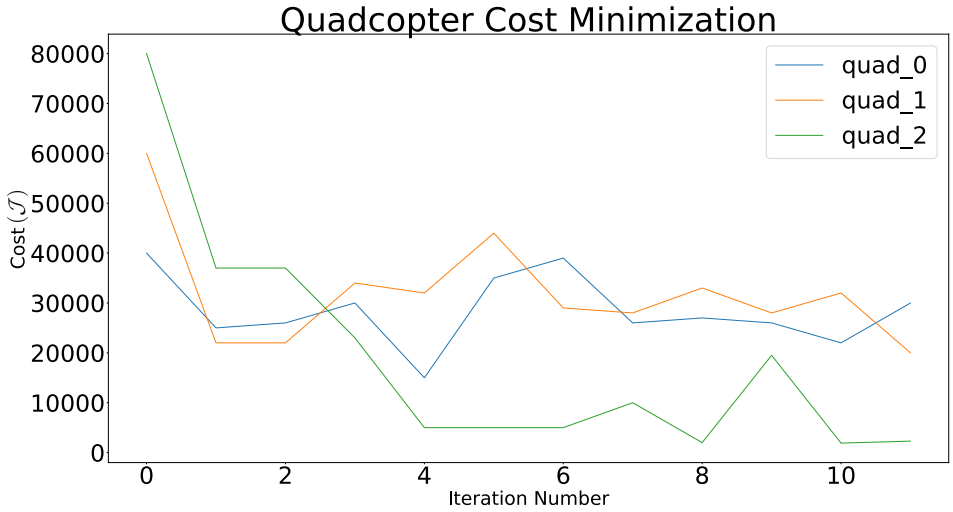


Figure 11. The MTAC-E Algorithm iterates 12 times over a subset of trajectories and produces the trajectory with the lowest cost after all iterations. Here, we show the algorithm evaluating which quadcopter should transition to LOC_2. This calculation is formulated in our cost function where we minimize 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.

6. Online MTAC-E

The MTAC-E algorithm developed in Section 4 allowed the use of a stochastic optimization technique for task allocation of a multi-agent system. In addition to this, we utilize cost functions to assign agents a task based on the individual constraints of an agent and its environment. However, this algorithm was developed in an offline manner, meaning it must run to completion before tasks can be delegated to a set of agents. This is time consuming and detrimental for time critical task assignment scenarios. The offline algorithm also considers optimal costs defined a-priori by an expert individual. This assumption makes it difficult for users to utilize this algorithm without prior information of the desired cost function, agent and environment that a multi-agent system will operate in. We solve both these issues by developing an online method of cross-entropy, modified to operate over multi-dimensional probability distributions. This method allows fast generation of trajectories in real-time and an expert independent method of optimal cost generation.

In order to apply an online version of the MTAC-E algorithm, we consider the trajectory $\mathcal{X}_t$ sampled at time t. The trajectory $\mathcal{X}_t = (\mathcal{X}_{1t}, \ldots, \mathcal{X}_{nt})$ is a vector of i.i.d. random variables $\mathcal{X}_{it}$ drawn from known multivariate Gaussian distributions $\mathcal{N}(\mu, \Sigma)$. Similar to [32], we consider a sample, $\mathcal{X}_{N_e}$, elite if at position $\rho \cdot |X|$, with $\rho > 0$ and $|X|$ the cardinality of the set of trajectories drawn at time t, it belongs to the subset of ordered trajectories $\{\mathcal{X}_0 < \mathcal{X}_1 < \cdots < \mathcal{X}_{\rho \cdot |X|}\}$. At each time step, this sample is chosen based on if its corresponding cost $f(\mathcal{X}_{N_e}) \geq \lambda_t$ where λ_t is the elite threshold. The threshold (λ_t) either increases, decreases or stays constant according to four cases depending on where the new sample and the dropout sample belong in the set of N trajectories:

(1) New sample $(\mathcal{X}_{[N_e+1]})$ and dropout sample $(\mathcal{X}_{[N_e]})$ are elite, this is a rare event with probability ρ and as such has a small probability of occurring. In this event,

the threshold, λ and threshold position stays the same and does not change.

(2) New sample is elite but dropout sample is not elite. In this case, the threshold value will increase by the difference in cost between the new sample and the sample at the end of the elite set such that $\lambda_{t+1} = \lambda + f(\mathcal{X}_{[N_e+1]}) - f(\mathcal{X}_{[N_e]})$. This increase in λ is attributed to "expanding" the search of trajectories with similar costs to gain membership to the elite set.

(3) New sample and dropout are not elite. The threshold will stay the same.

(4) New sample is not elite and dropout sample is elite. In this case, the threshold value will decrease since the thresholding may be too high for samples to be considered elite. Thus, the threshold is lowered to $\lambda_{t+1} = \lambda + f(\mathcal{X}_{[N_e-1]}) - f(\mathcal{X}_{[N_e]})$.

Let $\Delta_t = E(f(\mathcal{X}_{[N_e+1]}) - f(\mathcal{X}_{[N_e]})|\mathcal{G}_t)$ be the update step for the threshold where $\mathcal{G}_t$ is the σ-algebra of all known random outcomes up to time t and $f(\mathcal{X}_{[N_e]})$ as the trajectory cost measured at the elite set threshold. Given that the sample positions within the set of trajectories are distributed uniformly,

$$\begin{aligned}
&E(\lambda_{t+1}|\mathcal{G}_t, \text{new sample is elite}) \\
&= \lambda_t + P(\text{case 1}) \cdot E(f(\mathcal{X}_{[N_e+1]}) - f(\mathcal{X}_{[N_e]})|\mathcal{G}_t) + P(\text{case 2}) \cdot 0 \\
&= \lambda_t + (1-\rho) \cdot \Delta_t.
\end{aligned}$$

Likewise, for the non-elite sample case

$$\begin{aligned}
&E(\lambda_{t+1}|\mathcal{G}_t, \text{new sample is not elite}) \\
&= \lambda_t + P(\text{case 3}) \cdot 0 + P(\text{case 4}) \cdot E(f(\mathcal{X}_{[N_e-1]}) - f(\mathcal{X}_{[N_e]})|\mathcal{G}_t) \\
&= \lambda_t - \rho \cdot \Delta_t.
\end{aligned}$$

The update step is equivalent to the product of the average difference between samples and the difference between $\mathcal{X}_t$ evaluated at the ρ^{th} probability and 1 sample above it. In order to create an update step for trajectories sampled from a Gaussian distribution, we develop a Gaussian approximation for multivariate distributions. Prior works that utilize online cross entropy for task allocation used an update step based on uniform or univariate Gaussian distributions, relying on one dimensional samples. Our work extends this by formulating an update step for multivariate Gaussian distributions in the following section.

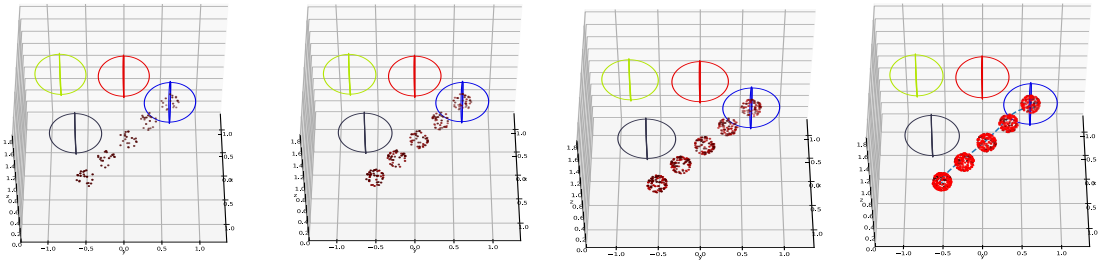
6.1. Developing the Update Step

The update step for samples from multivariate distributions will be a $n \times m$ matrix corresponding to n samples from $\mathcal{X}_t$, with each sample a m-dimensional vector. Consider a continuous update step that measures samples from normal distributions according to the desired ρ^{th} percentile. From the inversion principle [33]:

Fact 6.1. Let Φ be the cumulative distribution function on $\mathbb{R}^n$ with the inverse Φ^{-1} defined as

$$\Phi^{-1}(p) = \inf\{x \in \mathbb{R}^n : \Phi(x) \leq p, 0 < p < 1\}. \tag{9}$$

(1) If U is a uniform [0, 1] random variable then $\Phi^{-1}(U)$ has distribution Φ.



(a) t=2.5 (b) t=5 (c) t=7.5 (d) t=10

Figure 12. Samples drawn from the inverse distribution $\mathbb{F}^{-1}$ are shown in the above plot. Sample means μ are measured and shifted by the quantity given by the product of the covariance of $\mathcal{N}(\mu, \Sigma)$ and the standard normal inverse function Φ^{-1} evaluated at the ρ^{th} quantile. In these plots, sample means are measured via the straight line distance between a quadcopter and the desired hoop, the covariance matrix is $\Sigma = 0.05 \cdot \mathcal{I}$ and $\rho = 0.05$. At each iteration, new samples are drawn from the quantile $\mathbb{F}^{-1}$, here we show samples, corresponding to points drawn at the ρ^{th} quantile, drawn at different times during the runtime of the sampling algorithm. The final iteration at t=10 contains the trajectory the quadcopter follows.

(2) If X has distribution Φ, then $\Phi(X)$ is distributed uniformly on [0, 1].

Proof. The first statement can be verified through
$P(\Phi^{-1}(U) \leq x) = P(\inf\{y \in \mathbb{R}^n : \Phi(y) \leq U\} \leq x) = P(U \leq \Phi(x)) = \Phi(x)$.
The second statement is a result of the following relationship
$P(\Phi(X) \leq u) = P(X \leq \Phi^{-1}(u)) = \Phi(\Phi^{-1}(u)) = u$ □

The normal distribution we sample from is not a standard distribution, therefore our inverse function Φ^{-1} is

$$\mathbb{F}^{-1}(\rho) = \mu + \Sigma \cdot \Phi^{-1}(\rho) \tag{10}$$

where ρ is the desired percentile of $\mathcal{N}(\mu, \Sigma)$ and Σ is the covariance matrix. In [34], the average of two samples from a normal distribution is $2\sigma/\sqrt{\pi}$, extending to the multivariate distribution case, we form $\Delta_t = E|\mathcal{X}_1 - \mathcal{X}_2| \cdot \delta_{\mathbb{F}^{-1}(\rho)}$ as the difference between the quantile functions for a particular sample weighted by the expectation of two samples from a normal distribution with known parameters such that

$$\delta_{\mathbb{F}^{-1}(\rho)} = \frac{1}{2}\mathbb{F}^{-1}(1 - \rho + \frac{1}{N}) - \mathbb{F}^{-1}(1 - \rho) \tag{11}$$

$$\Delta_t = \frac{1}{\sqrt{\pi}} \cdot \Sigma[\mathbb{F}^{-1}(1 - \rho + \frac{1}{N}) - \mathbb{F}^{-1}(1 - \rho)]. \tag{12}$$

This formulation of Δ_t is a matrix and a scalar form is desirable for doing computations on λ, another scalar. We propose the following definition to achieve a scalar approximation of delta, $\Delta_{\tilde{t}}$.

Fact 6.2. Let Δ_t be the multivariate update step for threshold cost, λ. The scalar approximation of Δ_t is formulated as

$$\Delta_{\tilde{t}} = \min_i \sum_{j=1}^{m} |\Delta_{ij}| \in \mathbb{R} \tag{13}$$

minimizing the sum of quantile differences and returning the smallest difference be-

tween quantile samples.

Using $\Delta_{\tilde{t}}$ we update the desired threshold λ_t in real-time as samples are acquired. However, the quantile function $\Phi(x)$ is not easily acquired. In the next subsection, we define how to sample from a hyperellipsoid to generate uniformly sampled points from the ρ^{th} quantile of $\Phi(x)$.

6.2. Generating Quantile Function for Multivariate Gaussian

We begin by defining our problem as finding the contour line of a hyperellipsoid derived from the parameters of $\mathcal{N}(\mu, \Sigma)$.

Fact 6.3. Let $\mathbf{Y}$ be a sample from the surface of the hyperellipsoids with the following form

$$(\mathbf{Y} - \overrightarrow{\mu})^T \Sigma^{-1} (\mathbf{Y} - \overrightarrow{\mu}) \leq c^2$$

Where c is the desired distance of $\mathbf{Y}$ from mean $\overrightarrow{\mu}$. A semi-axis of the hyperellipsoid is $\sigma_i = \pm c\sqrt{\gamma_i} v_i$ and contains 100(1-ρ)% of the sampling distribution and $c^2 = \chi^2_{\rho,\alpha}$, the chi-squared distribution for α degrees of freedom measured at its ρ^{th} value.

Proof. The eigendecomposition of the covariance matrix is $\Sigma = VDV^T$, with V as the eigenvector matrix and D as the diagonal matrix of eigenvalues γ_i. We find the square root of $\Sigma = VS^{1/2}V^T$. $\Sigma = \Lambda\Lambda^T$ where $\Lambda = VD^{1/2}$. The matrix Λ scaled by a factor c results in $\Lambda^* = c\Lambda$ and likewise $\Sigma^* = c^2\Lambda\Lambda^T = c^2 VD^{1/2}D^{1/2}V^T = c^2\Sigma$ which determines the contour of the distribution considered by the sampled vector $\mathbf{Y}$. □

With our desired distance value, c, we project points sampled from a uniform hyperspheroid, R, onto the covariance Σ, shifted by the mean vector, μ. The points from hyperspheroid, R are sampled using the following algorithm [35] and we develop the following algorithm for sampling from quantile functions of multivariate distributions:

We provide a brief description of Algorithm 3 here. We begin with sampling the chi-squared distribution for a desired number of samples m at percentile ρ with α degrees of freedom at line 1. Followed by this, we find the square root of the received value and store it as c. We find the eigendecomposition of the covariance and its corresponding square root followed by the definition of Λ at line 4. Afterwards we follow the developments from [35] to sample from a hyperellipsoid. In line 6 we take the square root of the sum of squares of samples from a normal distribution sampled m times. The matrix $X_{nz\times m}$ results in points uniformly distributed on a hypersphere. Because we wish to access the values of this hypersphere stretched by $c\Sigma$ and centered on mean z, we apply the linear equation in line 10 to find points sampled from the ρ^{th} quantile, as shown in Fig. 12 over 5 selected means. This proposed method allows us to sample from a desired percentile of our sampling distribution for multivariate Gaussians. Desired trajectories corresponding to elite samples can be drawn from this sampling function and we use this sampling function to generate Δ_t at each iteration.

Algorithm 3: Quantile Sampling

Input : desired value of quantile function ρ, covariance matrix Σ, mean vector $\vec{\mu}$, degrees of freedom α, number of samples m

Output: point sampled from ρ^{th} quantile,Y

$c^2 \rightarrow \mathcal{X}^2_{\rho,\alpha}$

$c \rightarrow \sqrt{\mathcal{X}^2_{\rho,\alpha}}$

$\Sigma^{1/2} = VD^{1/2}V^T$

$\Lambda = c\Sigma^{1/2}$

$X_{nz\times m} \sim \mathcal{N}(0,1)$:

$r_{ss} = \sqrt{\sum_{i=0}^{i=m}(X^2_{nz\times i})}$

$K_x = \mathbf{1}_{nz\times 1} \otimes r_{ss}$

$X_{nz\times m} = X_{nz\times m}/K_x$

$z = \vec{\mu} \cdot \mathbf{1}_{1\times\alpha}$

$Y = X^T_{nz\times m} \cdot \Lambda + z^T$

6.3. Online Cross Entropy

In Algorithm 4 from lines 1 - 5 we check if the run is the first run of the algorithm, if true, initial means μ_t are generated based on the current position of the quadcopter (r) and position of the hoop $(\mathcal{H})$. In addition to this we generate covariances (Σ), a randomly generated value for the desired cost (λ) and Δ_t is generated from the `Quantile_Sampling` function where we assume that the samples are generated from a distribution with N trajectories. If the run is not the initial run, we use the previous μ, λ, and Δ_t. Following this, we perform the matrix operation from Equation 13 on Δ_t. We sample a trajectory $\mathcal{X}_t$ from a normal distribution by interpolating between path samples drawn from the distribution using the means and covariances previously acquired. If the cost function $f(\mathcal{X}_t)$ and the trajectory satisfy the constraint in line 14 then λ_t is increased. Otherwise, λ is decreased conditional on the case that the trajectory goes through the hoop but does not satisfy the cost constraint. Finally, Δ_t is updated and we return the pose in $\mathcal{X}_t$ that corresponds to time t.

6.4. Update Δ_t Threshold

Our value for incrementing the desired threshold, γ, can also be modified with an exponential factor $\beta = ae^{-bt}$ with $a, b > 0$. The delta threshold is updated at each time step t such that $\Delta_{\tilde{t}+1} = (1-\beta)\Delta_{\tilde{t}} + \beta|f(x_t) - f(x_{t+1})|$. Using this update formulation, initially, $\Delta_{\tilde{t}}$ weights information from the costs between samples as more important than prior $\Delta_{\tilde{t}}$ values. This is beneficial for us since our samples are drawn uniformly from a quantile function whose initial samples may not be reliable but allows a sufficient search of the cost function space of nearby samples. In Fig. 13 we show how varying values of a and b affect the convergence of $\Delta_{\tilde{t}}$. In this figure, we plot the change in $\Delta_{\tilde{t}}$ as the quadcopter runs the online cross entropy algorithm to converge to a desired hoop in 10 seconds. Initial $\Delta_{\tilde{t}}$ update values can vary due to dependence on $f(\mathcal{X}_t)$, so instead we focus on the convergence rate of each plot. We can see that for $a = 0.1$ and $b = 0.01$ as time progresses, $\Delta_{\tilde{t}}$ quickly converges to a value close to zero as $\Delta_{\tilde{t}}$ moves from reliance on relative difference in measured cost to previous $\Delta_{\tilde{t}}$ values. On the other extreme, for $a = 0.001$ and $b = 0.0001$, $\Delta_{\tilde{t}}$ does not move far from its initial value and stays near the value it eventually converges to during the run

Algorithm 4: Online Cross Entropy

Input : robot position r, desired hoop $\mathcal{H}$, desired value of quantile function ρ, previous cost $f(x)_{t-1}$, previous time t_{t-1}, previous lambda λ_{t-1} previous delta Δ_{t-1}, previous means μ_{t-1}, previous covariancesΣ_{t-1}, previous samples X_{t-1}, number of total trajectories N

Output: updated desired state p_t, means μ_t, covariances Σ_t, path samples X_t, lambda λ_t, cost $f(x)_t$, delta Δ_t

if *initial run is True* **then**
  $\mu_t \rightarrow$ `generate_means`$(r, \mathcal{H})$
  $\Sigma_t \rightarrow$ `generate_initial_covariances`()
  $\lambda_t \rightarrow$ `random_generator`()
  $\Delta_t \rightarrow \frac{\Sigma}{\sqrt{\pi}} \cdot$ `Quantile_Sampling`$(1 - \rho + \frac{1}{N}, \mu_t, \Sigma_t)$ - `Quantile_Sampling`$(1 - \rho, \mu_t, \Sigma_t)$
end
else
  $\mu_t \rightarrow \mu_{t-1}$
  $\lambda_t \rightarrow \lambda_{t-1}$
  $\Delta_t \rightarrow \frac{\Sigma}{\sqrt{\pi}} \cdot$ `Quantile_Sampling`$(1 - \rho + \frac{1}{N}, \mu_t, \Sigma_t)$ - `Quantile_Sampling`$(1 - \rho, \mu_t, \Sigma_t)$
end
$\Delta_{\tilde{t}} \rightarrow \Delta_t$
$\mathcal{X}_t \sim \mathcal{N}(\mu, \Sigma)$
if $f(\mathcal{X})_t \geq \lambda_{t-1}$ *and* *goes_through_hoop*$(\mathcal{X}_t)$ *is True* **then**
  $\lambda_t \rightarrow \lambda_t + (1 - \rho) \cdot \Delta_{\tilde{t}}$
  $\mu_t, \Sigma_t \rightarrow$ `MLE`$(\mathcal{X}_t, \ldots, \mathcal{X}_{t-1})$
end
else if *goes_through_hoop*$(\mathcal{X}_t)$ *is True* **then**
  $\lambda_t \rightarrow \lambda_t - \rho \cdot \Delta_{\tilde{t}}$
end
$\Delta_t \rightarrow (1 - \beta)\Delta_{\tilde{t}-1} + \beta\Delta_{\tilde{t}}|(f(\mathcal{X}_t) - f(\mathcal{X}_{t-1})|$
$p_t \rightarrow \mathcal{X}_t(t)$
return p_t, μ_t, Σ_t, λ_t, $f(x)_t$, Δ_t

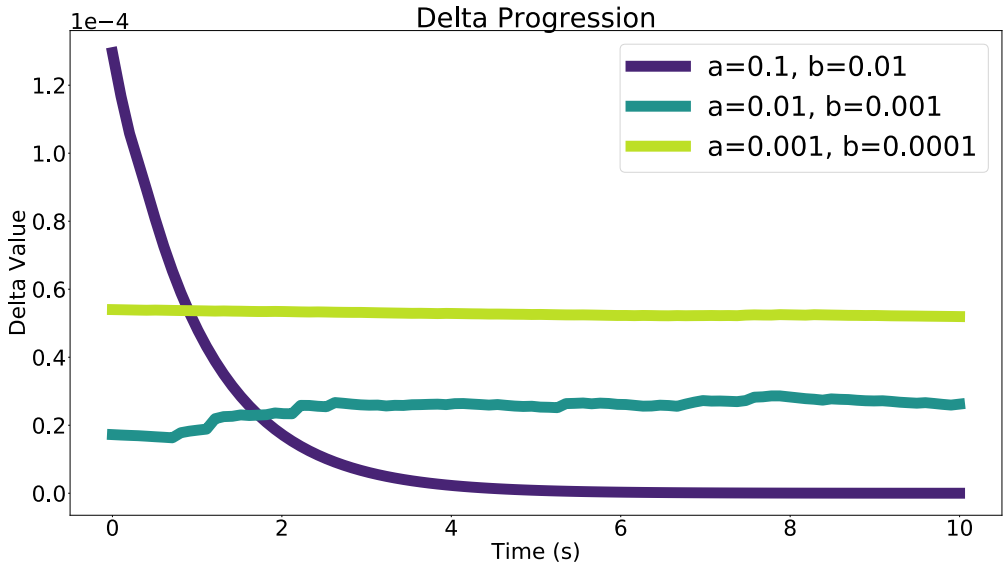


Figure 13. The time evolution of $\Delta_{\tilde{t}}$ for varying values of a and b. From the figure, a greater value of a and b indicate a greater dependence on the absolute error in sample trajectory costs $f(\mathcal{X})$ initially. This causes a greater change in magnitude of the step size of $\Delta_{\tilde{t}}$ than in smaller values of a and b and may reduce the sensitivity of the algorithm to desirable values of $\Delta_{\tilde{t}}$. We observe through empirical tests that values near $a = 0.01$ and $b = 0.001$ provide a reasonable trade-off between utilizing the trajectory costs and prior step size Δ_t for updating $\Delta_{\tilde{t}}$.

time. For all experiments, we chose $a = 0.01$ and $b = 0.001$ due to the values allowing for initial cost consideration, followed by slow convergence to values closer to expected quantile differences.

6.5. Online MTAC-E Algorithm

The Online MTAC-E algorithm presented in Algorithm 5 proceeds in the following section. The current robot poses for n robots is given, the time t, and the desired LTL specification (ϕ) to be satisfied. If the algorithm is in its first iteration we generate the sequence of propositions that satisfy the specification, otherwise we store this desired sequence for future iterations. Then for each node, in lines 5 - 14 a series of checks are done to verify if the node has been satisfied or if all the nodes have been satisfied. In addition to this, we track if each agent has had a chance to be assigned a proposition, when all agents have an assignment, the desired states are sent to each robot. If the robots are in the decomposition set $\mathcal{D}$ we consider the switch condition where each agents current state, desired trajectory to node q, and the cost function $f_i(x, u)$ associated with a particular agent to assign the node to the agent with the lowest cost. If the node is not in the decomposition set, the online cross entropy algorithm executes for a single agent. Due to the node not being in the decomposition set, once an agent is assigned to the set $\bar{D} = \{q \in \bar{\mathcal{D}} | q_i \neq q_0 \wedge (q_i, q_{i+1}) \in \mathcal{E}_{\bar{\mathcal{D}}}\}$ it may not transition to the decomposition set for the remainder of the run. The set of states p are returned for each robot agent where each state is the chain of integrators state referred to in Section 2. We have now formally generated an online method for task allocation using the stochastic optimization technique, cross entropy, for quadcopters. However, now we verify the algorithm against the offline version followed by experimental results.

Algorithm 5: Online MTAC-E Algorithm

Input : current robot poses $r_{i,\ldots,n}$ time t, LTL spec ϕ
Output: new poses $r_{i,\ldots,n}$, flag for specification satisfaction $satisfied\_ltl$

$satisfied\_ltl$ is `False`
if *first iteration* **then**
  `satisfying_run` $\rightarrow$ `generate_satisfying_run(`ϕ`)`
end
for q *in* $satisfying\_run$ **do**
  if q *is satisfied* **then**
    continue
  end
  if *all q satisfied* ***OR*** *all agents assigned* **then**
    if *all nodes satisfied* **then**
      $satisfied\_ltl$ is `True`
    end
    break
  end
  if $q \in \mathcal{D}$ **then**
    $r_{i,\ldots,n-1}$ $\rightarrow$ `switch_condition(`$r_{i,\ldots,n-1}$`,` $f_{i,\ldots,n-1}(x,u)$`, t)`
  end
  else
    r_j $\rightarrow$ `online_cross_entropy(...)`
  end
end
return $r_{i,\ldots,n}$, $satisfied\_ltl$

6.6. Comparison to Offline MTAC-E

In this section we compare the efficiency defined as the individual agent costs and run time performance of the online MTAC-E with the offline version. In Fig. 14, we compare the sum of agent costs for a predefined number of agents ranging from 1- 5. For each run, we assign the agents to satisfy the LTL specification . From the figure, we see that the online method is more efficient at overall task assignment as total agent costs are greatly reduced compared to the offline version. This is due to the fact that we must initially set an estimate for optimal costs for the offline MTAC-E while the online version can use a random estimate for optimal costs and iterate towards a local minima that is more efficient. In addition to this, a higher estimate of optimal costs needs to be given for the offline version in order for faster execution time otherwise the completion time of the program could be significantly long. However, we note there is an increase in the sum of agent costs since in the online MTAC-E algorithm, each time an agent is considered for a new task, the cost is added to the total trajectory cost for that agent.

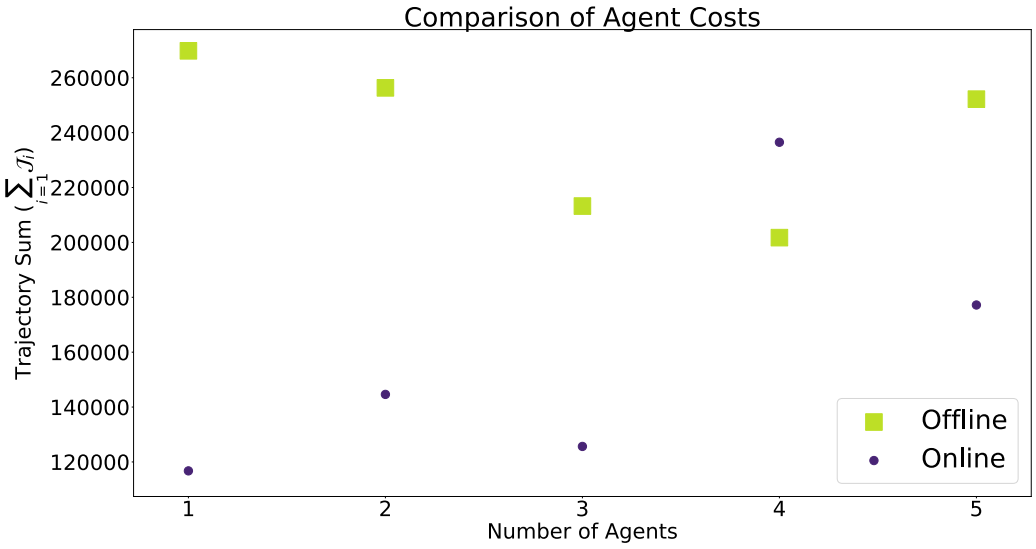


Figure 14. We compare the total trajectory costs for separate runs of the online and offline MTAC-E algorithm. We can see that the total costs associated with different numbers of agents is reduced for agents in the online case. This is due to a measurement of a known quantile function of a distribution and an optimal trajectory cost that is randomly initiated and updated at each sample step. This reduces the need of an expert to determine optimal costs for a particular domain.

Run time comparisons are done on a laptop with a 2.6 GHz Intel i7-4720HQ processor using the time module in Python. Each run is done in simulation and execution times are measured against complete satisfaction of the given LTL specification. We see from Fig. 15 a 3x - 5x factor of reduction of runtime for agents 1-5 indicating a faster algorithm for multi-agent task allocation.

From simulation results, we see the online MTAC-E algorithm is not only more efficient at minimizing the total costs for a multi-agent system but also in reducing individual agent costs and the overall run time associated with allocating tasks to a multi-agent system. We also verify our online algorithm experimentally in Section 7.

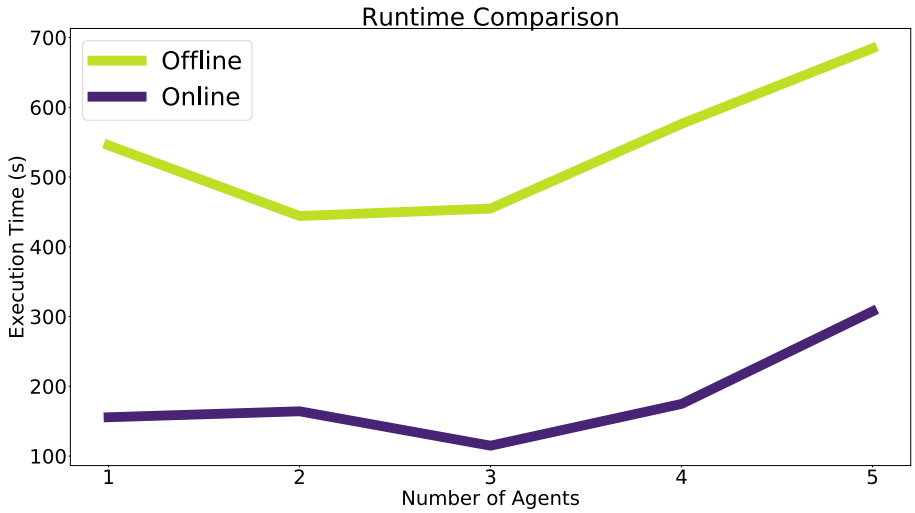


Figure 15. We compare the time to completion for both the online and offline MTAC-E algorithm. For all runs of the online MTAC-E with various numbers of agents considered, a lower total run-time is achieved.

7. Case Study: Fire Fighting Drones (Experiment)

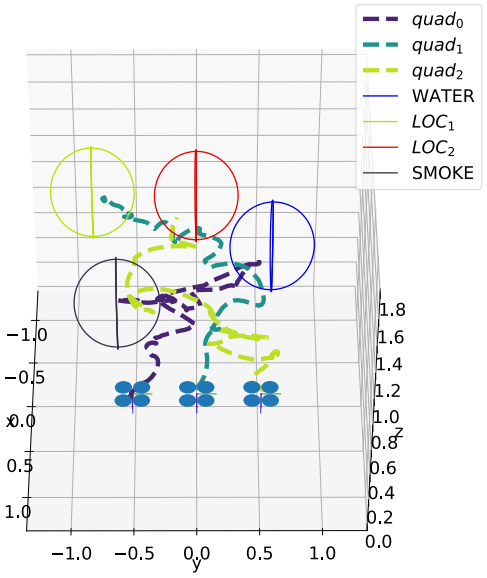
Experiments are validated on the Robotarium at Georgia Tech [36] using the Crazyflie 2.1 quadcopters. Control inputs are calculated from continuously differentiable splines generated online from desired waypoints using the differential flatness property of quadcopters described in Section 2.3. Quadcopter positions are tracked with a Vicon camera system with a tracking frequency of 100 Hz and the controller generates control inputs at a frequency of 50 Hz.

We recreate the simulated scenario of fire-fighting quadcopters by indicating desired regions of interest with hoops, characterized by ellipsoids, pictured in Fig. 16b [1]. These hoops are covered with Vicon tracking markers, allowing us to record the corresponding center of the hoops and from the centers form the proposition sets. The regions of interest are satisfied if a quadcopter flies within 0.2 meters of the hoop. We give initial starting positions of

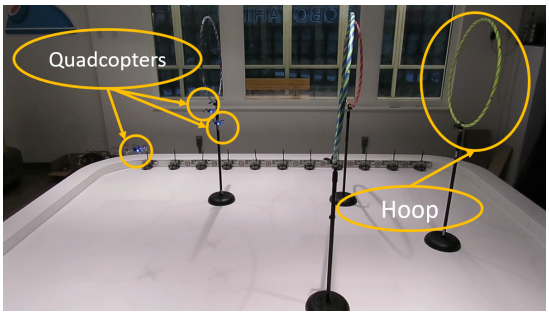
$$p_1 = \begin{bmatrix} 1 \\ -0.5 \\ 0.8 \end{bmatrix}^T, p_2 = \begin{bmatrix} 1 \\ 0 \\ 0.8 \end{bmatrix}^T, p_3 = \begin{bmatrix} 1 \\ 0.5 \\ 0.8 \end{bmatrix}^T$$

for three quadcopters and the desired global LTL specification $\phi = \Diamond LOC_1 \wedge \Diamond LOC_2 \wedge \Box(SMOKE \implies CARRYING)$. We plot the real trajectories of the quadcopters in Fig. 16a and given the initial positions, control inputs and global LTL specification we utilize the online MTAC-E algorithm to generate trajectories in real-time for each quadcopter. The assignments shown in the figure are dynamically allocated tasks given to each robot based on the individual cost function associated with each agent and environment constraints. These assignments are not strictly assigned and could change given new information (e.g. more samples from the online cross entropy algorithm) or a different cost function. We run the algorithm on a desktop with an Intel Core i7-

[1] Experiment Video: Online Multi-Agent Task Allocation via Cross Entropy



(a) The real trajectories of each quadcopter superimposed over the simulation environment. The locations of the hoops are the same as those in the experimental run.



(b) The quadcopters are flown in the Robotarium where hoops are used to validate the experiment. Each hoop has tracking markers to locate the center of mass of each hoop, used to calculate control points and the desired ellipsoids used to characterize the hoops. Quadcopters are also tracked with the Vicon system.

Figure 16. The trajectories of the quadcopters and a visualization of the experimental run are shown in the figure above.

7700K processor with 16 GB of RAM. The total run time of the experiment is 127 seconds and terminates when the entire desired sequence of hoops is satisfied for a specified LTL specification. Through this experiment, we show the implementation of the online MTAC-E algorithm on quadcopters which validates our task orchestration framework by decomposing a specification, delegating tasks and generating trajectories in **real-time** for a set of quadcopters.

8. Conclusion

In conclusion, we have generated a framework for generating sequences of hoops from user defined LTL specifications. Followed by a method for multi-agent task allocation utilizing this framework in addition to a stochastic optimization technique for assigning decomposed assignments to agents with unknown action costs and a known cost function. We extend the multi-agent task allocation methodology to generate desired trajectories online via hyperellipsoid sampling and estimation of minimal quantile distribution differences. This allowed us to create a faster and more efficient method of trajectory sampling for multi-agent task allocation given an LTL specification that contained system and environmental constraints. To the authors knowledge, this is one of the first papers to develop online task allocation for quadcopters using cross entropy optimization and validate the algorithm experimentally on hardware. In addition to this we propose a formulation for the update step using the quantile functions of multivariate Gaussian distributions. We verify this in experiment and simulation as a task orchestration framework for decomposing and delegating tasks and generating trajectories for a multi-agent system to satisfy high-level user specifications given these constraints.

Disclosure statement

No potential conflicts of interest were reported by the authors.

Funding

This material is partially based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE- 1650044, in part through the grant ARL DCIST CRA W911NF-17-2- 0181 and NSF project 1836932.

Data Availability Statement

Raw data were generated in the Robotarium at Georgia Tech. Derived data supporting the findings of this study are available from the corresponding author (CB) on request.

References

[1] Khamis AM, Elmogy AM, Karray FO. Complex task allocation in mobile surveillance systems. Journal of Intelligent & Robotic Systems. 2011;64(1):33–55.

[2] Khamis A, Hussein A, Elmogy A. Multi-robot task allocation: A review of the state-of-the-art. In: Cooperative robots and sensor networks 2015. Springer; 2015. p. 31–51.
[3] Duarte M, Gomes J, Costa V, et al. Application of swarm robotics systems to marine environmental monitoring. In: OCEANS 2016-Shanghai; IEEE; 2016. p. 1–8.
[4] Chen J, Moarref S, Kress-Gazit H. 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. p. 568–576.
[5] Aksaray D, Vasile CI, Belta C. Dynamic routing of energy-aware vehicles with temporal logic constraints. In: Proc. IEEE ICRA 2016; May 16–21,; Stockholm, Sweden; 2016. p. 3141–3146.
[6] Baier C, Kateon JP. Principles of model checking. Cambridge, Massachusetts: The MIT Press; 2008.
[7] Loizou SG, Kyriakopoulos KJ. 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. p. 153–158.
[8] Fainekos GE, Girard A, Kress-Gazit H, et al. Temporal logic motion planning for dynamic robots. Automatica. 2009;45(2):343–352.
[9] Wu M, Yan G, Lin Z, et al. Synthesis of output feedback control for motion planning based on ltl specifications. In: 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems; IEEE; 2009. p. 5071–5075.
[10] Vasile CI, Belta C. Sampling-based temporal logic path planning. CoRR. 2013; abs/1307.7263. Available from: http://arxiv.org/abs/1307.7263.
[11] Baier C, Katoen JP. Principles of model checking. MIT press; 2008.
[12] Banks C, Slovak K, Coogan S, et al. Specification-based maneuvering of quadcopters through hoops. In: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS); IEEE; 2019. p. 5191–5197.
[13] Kobilarov M. Cross-entropy motion planning. The International Journal of Robotics Research. 2012;31(7):855–871.
[14] de Boer PT, et al. A tutorial on the cross-entropy method. Annals of Operations Research. 2005;134(1):19–67.
[15] Karaman S, Frazzoli E. Sampling-based algorithms for optimal motion planning. The international journal of robotics research. 2011;30(7):846–894.
[16] Laguna M, Duarte A, Martí R. Hybridizing the cross-entropy method: An application to the max-cut problem. Computers & Operations Research. 2009;36(2):487–498.
[17] Chepuri K, Homem-De-Mello T. Solving the vehicle routing problem with stochastic demands using the cross-entropy method. Annals of Operations Research. 2005;134(1):153–181.
[18] Livingston SC, Wolff EM, Murray RM. Cross-entropy temporal logic motion planning. In: Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control; ACM; 2015. p. 269–278.
[19] Banks C, Wilson S, Coogan C, et al. Multi-agent task allocation using cross-entropy temporal logic optimization. In: International Conference on Robotics and Automation; 2020.
[20] Schillinger P, Burger M, Dimarogonas DV. Decomposition of finite LTL specifications for efficient multi-agent planning. In: Distributed autonomous robotic systems. Springer; 2018. p. 253–267.
[21] Zhang X, Wang K, Dai W. Multi-uavs task assignment based on fully adaptive cross-entropy algorithm. In: 2021 11th International Conference on Information Science and Technology (ICIST); 2021. p. 286–291.
[22] Gerevini AE, Haslum P, Long D, et al. Deterministic planning in the fifth international planning competition: Pddl3 and experimental evaluation of the planners. Artificial Intelligence. 2009;173(5-6):619–668.
[23] De Giacomo G, Vardi MY. Linear temporal logic and linear dynamic logic on finite traces.

In: Twenty-Third International Joint Conference on Artificial Intelligence; 2013.
[24] Holzmann GJ. The spin model checker: Primer and reference manual. Vol. 1003. Addison-Wesley Reading; 2004.
[25] Bauer A, Leucker M, Schallhart C. Runtime verification for LTL and TLTL. ACM Transactions on Software Engineering and Methodology (TOSEM). 2011;20(4):14.
[26] De Giacomo G, De Masellis R, Montali M. Reasoning on LTL on finite traces: Insensitivity to infiniteness. In: Twenty-Eighth AAAI Conference on Artificial Intelligence; 2014.
[27] Lacerda B, Parker D, Hawes N. Optimal and dynamic planning for markov decision processes with co-safe ltl specifications. In: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems; 2014. p. 1511–1516.
[28] Bishop CM. Pattern recognition and machine learning (information science and statistics). Berlin, Heidelberg: Springer-Verlag; 2006.
[29] Wang L, Theodorou EA, Egerstedt M. Safe learning of quadrotor dynamics using barrier certificates. In: 2018 IEEE International Conference on Robotics and Automation (ICRA); IEEE; 2018. p. 2460–2465.
[30] Mellinger D, Kumar V. Minimum snap trajectory generation and control for quadrotors. In: Robotics and Automation (ICRA), 2011 IEEE International Conference on; IEEE; 2011. p. 2520–2525.
[31] Kolling A, Walker P, Chakraborty N, et al. Human interaction with robot swarms: A survey. IEEE Transactions on Human-Machine Systems. 2016;46(1):9–26.
[32] Szita I, Lörincz A. Online variants of the cross-entropy method. CoRR. 2008; abs/0801.1988. Available from: http://arxiv.org/abs/0801.1988.
[33] Devroye L. Nonuniform random variate generation. Handbooks in operations research and management science. 2006;13:83–121.
[34] David H, Nagaraja H. Order statistics wiley. New York. 1981;.
[35] Dezert J, Musso C. An efficient method for generating points uniformly distributed in hyperellipsoids. In: Proceedings of the Workshop on Estimation, Tracking and Fusion: A Tribute to Yaakov Bar-Shalom; 2001.
[36] Pickem D, Glotfelter P, Wang L, et al. The Robotarium: A remotely accessible swarm robotics research testbed. In: Robotics and Automation (ICRA), 2017 IEEE International Conference on; IEEE; 2017. p. 1699–1706.