Multi-Agent Coordination Subject to Counting Constraints: A Hierarchical Approach

Yunus Emre Sahin[1], Necmiye Ozay[1], and Stavros Tripakis[2]

[1] Electrical and Computer Engineering, University of Michigan, Ann Arbor, MI 48109, USA
ysahin@umich.edu, necmiye@umich.edu
[2] Department of Computer Science, Aalto University
02150 Espoo, Finland
stavros.tripakis@gmail.com

Abstract. This paper considers the problem of generating multi-agent trajectories to satisfy properties given in counting temporal logic. A hierarchical solution approach is proposed where a coarse plan that satisfies the logic constraints is computed first at the higher level, followed by a lower-level task of solving a sequence of multi-agent reachability problems. Collision avoidance and potential asynchronous executions are also dealt with at the lower-level. When lower-level planning problems are found to be infeasible, these infeasibility certificates are incorporated into the higher-level problem to re-generate plans. The results are demonstrated with several examples that shows how the proposed approach scales with respect to different parameters.

Keywords: counting constraints, multi-agent path planning, formal methods, hierarchical planning

1 Introduction

Autonomous multi-agent systems can potentially serve the society in many applications, such as safety-critical search and rescue missions, warehouse robotics, autonomous vehicles. These tasks require coordination of large number of agents to be effective. Therefore, we need scalable tools that can handle large number of agents working in complex environments.

Traditionally, multi-agent coordination problems are formulated as simple tasks such as reaching a goal state while avoiding unsafe regions and collisions [1–4] or reaching a consensus [5, 6]. Temporal logics, such as linear temporal logic (LTL), provide a more powerful framework where more complex tasks can be specified [7–14]. However, the curse of dimensionality prevents these methods to be applied to a large number of agents. To alleviate this concern, hierarchical methods for LTL constraints are proposed in [7, 12, 15]. In our early work [16], we introduced *counting linear temporal logic (cLTL)*, a formal language designed specifically to define multi-agent tasks, and showed that the solution method can scale to hundreds of agents. However, one of the limiting factors in multi-agent setting is the synchronization of agents. In real-life applications, perfect synchronization is difficult to achieve and synchronization errors might lead to collisions or violation of the specifications. In this respect, a method that is based on assigning priorities to the agents to handle synchronization errors is proposed in [13], which could lead to conservatism. We provided an extension of cLTL in [17], called *counting linear temporal logic plus (cLTL+)*, and showed how to generate solutions that are robust to synchronization errors without needing to prioritize the agents. However, generating robust solutions is either conservative and/or computationally expensive and limits the scalability of these methods.

This paper proposes a hierarchical method to generate multi-agent trajectories to satisfy properties given in *counting temporal logic plus without 'next' operator (cLTL+$_{\setminus\bigcirc}$)*. Similar to [17], agent dynamics are given by a transition system. At the higher level, an abstraction of this transition system is computed and used to generate *coarse plans* that satisfy the specifications. These plans are then refined at the lower level by solving a sequence of multi-agent reachability problems. With this hierarchy, we shift the computational burden of avoiding collisions to the lower level where it can be handled much more efficiently. The method proposed in this paper scales better with the size of the transition system as the size of the abstraction is guaranteed to be smaller than or equal to the original transition system. This paper also shows that, under mild conditions, solutions robust to synchronization errors can be generated. As opposed to [17], which requires a bound on the maximum synchronization error to achieve such robustness, this paper shows how to generate robust solutions that satisfy the specifications even if this bound is not known, by using a lower-level execution policy to 'correct' synchronization errors at run-time. Moreover, the complexity of our algorithm does not depend on the synchronization error bound.

The rest of the paper is structured as follows: Section 2 introduces the notation and provides definitions that are used in the rest of the paper. Section 3 provides a hierarchical method to solve the problem of interest and shows how to handle with synchronization errors. Section 4 demonstrates the efficacy of our method with several examples and Section 5 concludes the paper.

2 Preliminaries

The set of non-negative integers is denoted by $\mathbb{N}$ and the set of integers from 1 to N is denoted by $[N]$. An atomic proposition is a statement that is either *false* or *true*. The set $\mathbb{B} = \{\bot, \top\}$ denotes the logical values false and true, respectively. Cardinality of a set A is denoted by $|A|$.

2.1 Transition Systems

Definition 1 *A* transition system *is a tuple $T = (V, E, AP, L)$ where V is a finite set of states, $E \subseteq V \times V$ is a transition relation, AP is a finite set of atomic propositions and $L : V \to 2^{AP}$ is a labeling function.*

Transition system $T = (V, E, AP, L)$ represents the workspace agents operate in. Transitions systems that capture the behaviors of complex robotic systems can be obtained by using motion primitives [18, 19] or abstraction-based methods [20, 21]. Set of states V is a partition of the workspace and E corresponds to transitions between these states. Labeling function L marks the states with atomic propositions AP. These markings are used to define regions of interest. For instance some states can be marked as 'charging station', or 'obstacle', etc.

A *path* of T, also called a *T-path*, is a sequence of states $\pi : v_0 v_1 \ldots$ such that $(v_t, v_{t+1}) \in E$ for all t. A path can have a finite length, or it can be infinite. We denote by $\pi(t)$ the t^{th} element of π. A pair of T-paths π_1 and π_2 are said to be in *collision* if there exists a t such that $\pi_1(t) = \pi_2(t)$ or, $\pi_1(t+1) = \pi_2(t)$ and $\pi_2(t+1) = \pi_1(t)$. Each path has a *trace* associated with it, $\sigma_\pi : \mathbb{N} \to 2^{AP}$ where $\sigma_\pi(t) = L(\pi(t))$. A pair of traces σ_1 and σ_2 are said to be *stutter trace equivalent*, if removing repetitions makes them identical. For example, $\sigma_1 = \{a\}\{a\}\{a,b\}\{b\}\{b\}\{c\}$ and $\sigma_2 = \{a\}\{a,b\}\{b\}\{c\}\{c\}$ are stutter trace equivalent but σ_1 and $\sigma_3 = \{a\}\{a,b\}\{b\}\{c\}\{b\}\{c\}$ are not. With a slight abuse of notation, we also say a pair of T-paths π_1 and π_2 are stutter trace equivalent, if corresponding traces σ_{π_1} and σ_{π_2} are stutter trace equivalent.

2.2 Counting Linear Temporal Logic

We now move on to temporal constraints. We assume that specifications are given in *counting temporal logic plus without 'next' operator (cLTL+$_{\backslash\bigcirc}$)*, a counting logic that enables us to specify multi-agent tasks in a concise manner [17]. This logic is an extension of commonly used linear temporal logic (LTL) [22]. In cLTL+$_{\backslash\bigcirc}$, the inner logic defines tasks that can be satisfied by a single agent. For example *"repeatedly visit the charging station"*, *"do not enter region A before visiting region B"*, etc., are tasks that can be expressed in the inner logic. Each inner logic formula is then paired with a non-negative integer in the outer logic to specify the minimum number of agents that are required to satisfy the inner logic formula. For example, *"Every agent should repeatedly visit the charging station and the number of agents in region A should be less than 5 until region B is populated by at least 2 agents"*.

For the sake of completeness, we provide the syntax and the semantics of cLTL+$_{\backslash\bigcirc}$. The inner logic over atomic proposition set AP is identical to LTL without 'next' operator over AP and the grammar is given by the following:

$$\phi :: \top \mid ap \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \,\mathcal{U}\, \phi_2 \tag{1}$$

where $ap \in AP$ and ϕ, ϕ_1, ϕ_2 are LTL formulas. Individual tasks such as "Do not enter region A before visiting region B", can be expressed as $\neg a \,\mathcal{U}\, b$ using inner logic. We denote by Φ the set of all inner logic formulas. If an infinite trace σ satisfies LTL formula ϕ at time t, it is denoted by $\sigma, t \models_{LTL} \phi$, where the subscript LTL emphasizes that LTL semantics is used [22].

The outer logic is slightly different from LTL, as it is based on a new type of proposition, which is called *temporal counting proposition (tcp)*. A *tcp* is obtained by pairing an LTL formula ϕ with a non-negative integer m. The syntax and semantics of the outer logic are given as follows:

$$\mu :: \top \mid tcp \mid \neg\mu \mid \mu_1 \wedge \mu_2 \mid \mu_1 \,\mathcal{U}\, \mu_2 \tag{2}$$

where $tcp = [\phi, m] \in \Phi \times \mathbb{N}$ and μ, μ_1, μ_2 are cLTL+$_{\setminus\bigcirc}$ formulas. A $tcp = [\phi, m] \in \Phi \times \mathbb{N}$ is said to be satisfied at time t if and only if ϕ is satisfied by at least m agents at time t. Note that, besides the new type of propositions, the syntax of the outer logic is similar to that of LTL without next. We now provide the semantics with the assumption that agents move synchronously. We later relax this assumption in §2.3. Given a collection $\Sigma = \{\sigma_{\pi_1}, \dots, \sigma_{\pi_N}\}$ of N infinite traces, the satisfaction of a cLTL+$_{\setminus\bigcirc}$ formula μ at time t is denoted by $\Sigma, t \models \mu$ and inductively defined as follows:

$$\begin{aligned}
&\Sigma, t \models [\phi, m] \text{ if and only if } |\{n \mid \sigma_{\pi_n}, t \models_{LTL} \phi\}| \geq m,\\
&\Sigma, t \models \neg\mu \text{ if and only if } \Sigma, t \not\models \mu,\\
&\Sigma, t \models \mu_1 \wedge \mu_2 \text{ if and only if } \Sigma, t \models \mu_1 \text{ and } \Sigma, t \models \mu_2,\\
&\Sigma, t \models \mu_1 \,\mathcal{U}\, \mu_2 \text{ iff there exists } l \geq 0 \text{ such that } \Sigma, t+l \models \mu_2\\
&\qquad\qquad\qquad \text{and } \Sigma, t+k \models \mu_1 \text{ for all } k < l.
\end{aligned} \tag{3}$$

The semantics of the outer logic, with the exception of satisfaction of a tcp, is similar to regular LTL. The intuition is that Σ represents the behaviors of N agents, where σ_{π_n} corresponds to the behavior of agent a_n. Agent a_n is said to satisfy the inner logic formula ϕ at time t if $\sigma_{\pi_n}, t \models_{LTL} \phi$. Then $[\phi, m]$ is satisfied at time t if the number of agents satisfying ϕ at time t are greater than or equal to m, i.e., $|\{n \mid \sigma_{\pi_n}, t \models_{LTL} \phi\}| \geq m$. When $\Sigma, 0 \models \mu$, we say that μ is satisfied by Σ and write $\Sigma \models \mu$. Other commonly used operators such as $\Diamond$(*eventually*) and $\Box$(*always*) are defined in the usual way: $\Diamond a \doteq true \,\mathcal{U}\, a$ and $\Box a \doteq \neg(\Diamond\neg a)$.

Example 1. Assume the following specification for N agents is given in plain English: "Every agent should repeatedly visit the charging station and the number of agents in region A should be less than 5 until region B is populated by at least 2 agents". Mark region A, region B and the charging station, with atomic propositions a, b and c, respectively. Then the specification is expressed in cLTL+$_{\setminus\bigcirc}$ as $\mu = [\Box\Diamond c, N] \wedge (\neg[a, 5] \,\mathcal{U}\, [b, 2])$.

The inner logic $\Box\Diamond c$ is satisfied by any agent if that agent repeatedly (infinitely many times) visits the charging station, marked by c. Then $[\Box\Diamond c, N]$ is satisfied if at least N agents (all agents) repeatedly visit the charging station. Similarly, $\neg[a, 5]$ (or $[b, 2]$) is satisfied at time t, if less than 5 (or at least 2) agents satisfy proposition a (or proposition b) at that time. Combining all, μ specifies the same task that is given in plain English.

2.3 Time Robustness

Our definitions so far assume perfect synchronization of agents. However, perfect synchronization of agents is a challenging task and synchronization errors, if not handled with care, might result in violation of the specifications. For instance, assume there are 2 agents and the specification requires a certain property p to be satisfied by at least one agent at all times, i.e., $\mu = \Box[p, 1]$. Let π_i denote a path for agent a_i and

$$\begin{array}{rcccccc} \sigma_{\pi_1} & = & \{p\} & \{\neg p\} & \{\neg p\} & \{\neg p\} & \ldots, \\ \sigma_{\pi_2} & = & \{\neg p\} & \{p\} & \{p\} & \{p\} & \ldots \end{array}$$

be corresponding traces. Property p is satisfied by agent a_1 only at time $t = 0$ and by a_2 at all times except $t = 0$. If agents are perfectly synchronized, the specification μ would be satisfied. However, if a_2 moves slower than intended and causes a synchronization error, μ would be violated.

When agents are allowed to move asynchronously, there are infinitely many ways a collection of infinite paths $\{\pi_1, \ldots, \pi_N\}$ could be executed. We identify a particular *execution* by a collection of mappings $\{k_1, \ldots, k_N\}$ where $k_n : \mathbb{N} \to \mathbb{N}$, called the *local time mapping of* a_n, maps the global time t to the position of the agent a_n on the map π_n at time t. In other words, the state of agent a_n at time t is given by $\pi_n(k_n(t))$. Since agents need to progress eventually and respect the order of states on a path, each k_n needs to satisfy the following assumptions:

$$k_n(0) = 0, \quad k_n(t-1) \leq k_n(t) \leq k_n(t-1) + 1, \quad \lim_{t \to \infty} k_n(t) = \infty \tag{4}$$

Given a collection of N infinite traces $\Sigma = \{\sigma_{\pi_1}, \ldots, \sigma_{\pi_N}\}$ and an execution $K = \{k_1, ..., k_N\}$, the pair (Σ, K) is called a *collective trace*. Semantics in (3) are stated with the assumption that agents move synchronously. To generalize to asynchronous executions, we replace every Σ of (3) with (Σ, K) and modify the first line as follows:

$$(\Sigma, K), t \models [\phi, m] \text{ if and only if } |\{n \mid \sigma_{\pi_n}, k_n(t) \models_{LTL} \phi\}| \geq m. \tag{5}$$

We say that (Σ, K) and (Σ', K') are *identical collective traces* if $\sigma_{\pi_n}(k_n(t)) = \sigma_{\pi'_n}(k'_n(t))$ for all t and for all n. In practice, it is reasonable to assume that there is an upper bound on the asynchrony between agents. The collection $K = \{k_1, \ldots, k_N\}$ is called a τ*-bounded asynchronous execution* if the difference between local times never exceeds τ, i.e., $\max_t(\max_{m,n}(|k_m(t) - k_n(t)|)) \leq \tau$.

Definition 2 *A cLTL+$_{\setminus \bigcirc}$ formula μ is said to be τ-robustly satisfied by a collection $\Sigma = \{\sigma_{\pi_1}, \ldots, \sigma_{\pi_N}\}$ at time t, denoted $\Sigma, t \models_\tau \mu$, if following holds for all τ-bounded executions K:*

$$(\Sigma, K), \alpha \models \mu, \textit{ for all } \alpha \textit{ such that } \min_n k_n(\alpha) = t \tag{6}$$

We write $\Sigma \models_\tau \mu$ for $\Sigma, 0 \models_\tau \mu$. Also, we extend the satisfaction relation from traces to paths of a transition system. If $\Pi = \{\pi_1, ..., \pi_N\}$ is a collection of paths, then $\Pi \models \mu$ means $\{\sigma_{\pi_1}, \ldots, \sigma_{\pi_N}\} \models \mu$.

Note that $\models_\tau$ implies $\models_{\tau'}$ for all $\tau' < \tau$. For more details about robust satisfaction of a cLTL+ formula, we refer the reader to Definition 7 of [17].

2.4 Problem Definition

We now formally state the problem we are interested in:

Problem 1 *Let a transition system $T = (V, E, AP, L)$, a set of agents $\mathcal{A} = \{a_1, \ldots, a_N\}$ with initial conditions $S_0 : \mathcal{A} \to V$, a cLTL+ formula μ over AP and non-negative integer $\tau \in \mathbb{N}$ be given. Synthesize a collection $\Pi = \{\pi_1, ..., \pi_N\}$ of collision-free T-paths such that $\Pi \models_\tau \mu$ and $\pi_n(0) = S_0(a_n)$ for all $a_n \in \mathcal{A}$.*

An instance to Problem 1 is characterized by a tuple $(T, \mathcal{A}, S_0, \mu, \tau)$ and a solution to an instance would be a collection $\Pi = \{\pi_1, ..., \pi_N\}$ of T-paths. We first solve for the special case $\tau = 0$. Solutions generated for this case are not robust to synchronization errors, so we discuss how the algorithm can be modified to deal with the case $\tau = 1$. Furthermore, under mild assumptions, we show that robust solutions can be generated such that μ is satisfied for any τ.

Note that the problem definition only imposes that generated paths are collision-free, which is defined for synchronous executions, i.e., when $\tau = 0$. This suffices to establish our main theoretical result, which is synchronous satisfaction of μ (Proposition 2). Under certain assumptions, such paths are indeed sufficient (as shown in §3.5) for the existence of execution policies that guarantee that agents do not collide even when they navigate these paths asynchronously.

3 Solution Method

The previous method presented in [17] to solve Problem 1 and, in general, other existing algorithms to solve similar multi-agent problems are computationally expensive. The problem becomes intractable if the size of the transition system T is large. We propose a *hierarchical* approach that scales better with the size of the transition system. Our method is illustrated in Algorithm 1. In the following, we give an overview of the algorithm and then explain its parts in detail.

Given $(T, \mathcal{A}, S_0, \mu, 0)$, we first compute a new transition system T^{abs}, called *abstraction* of T. How to compute T^{abs} is described in §3.1. The motivation behind computing an abstraction is that it has a smaller size compared to the original transition system; hence, it would decrease the computational resources required to solve Problem 1. After adjusting the initial conditions of T^{abs} as S_0^{abs} and we solve a slightly modified version of Problem 1 instance $(T^{abs}, \mathcal{A}, S_0^{abs}, \mu, 1)$, relaxing the collision avoidance constraint. A solution to this instance is called an *abstract plan*, which is a collection of T^{abs}-paths. These *abstract paths* satisfy the logic constraints and can be seen as guidelines. Rather than explicitly assigning each agent a path, they indicate what propositions it needs to satisfy and in which order. We then replace each abstract path with a stutter trace equivalent T-path to generate a solution to $(T, \mathcal{A}, S_0, \mu, 0)$.

Construction of the abstraction ensures the existence of stutter trace equivalent T-paths. However, a collection of such paths is not guaranteed to be collision free. To prevent collisions, we solve a sequence of path planning problems at the lower level. If all the path planning problems are feasible, a solution to $(T, \mathcal{A}, S_0, \mu, 0)$ can be extracted. On the other hand, if the abstract plan is not feasible, we generate a counter example to be used in the higher level and obtain a different abstract plan. These steps are repeated until a solution is found or the algorithm terminates with no solution. In the following, we explain the steps of the main algorithm in greater detail.

3.1 Abstraction

Given a transition system $T = (V, E, AP, L)$, we define an equivalence relation $\sim$ on V as follows:

$$u \sim v \text{ if and only if } u = v, \text{ or } L(u) = L(v) \text{ and there exist } T\text{-paths } \pi_{uv} \text{ and } \pi_{vu} \text{ from } u \text{ to } v \text{ and from } v \text{ to } u \text{ such that } \sigma_{\pi_{uv}}(t) = \sigma_{\pi_{vu}}(t) = L(u) \text{ for all } t. \tag{7}$$

Relation $\sim$ partitions V into equivalence classes $V_1, ..., V_C$, for some $C \in \mathbb{N}$, such that all V_i are pairwise disjoint and all states in each V_i are equivalent, with $V = \cup_{i=1}^{C} V_i$. In words, each class induces a strongly-connected subgraph within which all nodes satisfy the same property. We create a state v_i^{abs} for each equivalence class V_i and denote the set of all such states by V^{abs}. To map the states of V to states of V^{abs}, we define $\mathcal{R} : V \to V^{abs}$:

$$\mathcal{R}(v) \doteq v_i^{abs} \text{ if } v \in V_i. \tag{8}$$

By definition, inverse of $\mathcal{R}$ maps the states of V^{abs} to equivalence classes, i.e., $\mathcal{R}^{-1}(v_i^{abs}) = V_i$. Next we define *abstraction of* T, denoted by $T_{abs} \doteq (V^{abs}, E^{abs}, AP, L^{abs})$, such that:

$$\begin{aligned} E^{abs} &\doteq \{(v_i^{abs}, v_j^{abs}) \mid \exists (u, v) \in E, u \in \mathcal{R}^{-1}(v_i^{abs}) \text{ and } v \in \mathcal{R}^{-1}(v_j^{abs})\}, \\ L^{abs}(v_i^{abs}) &\doteq L(v) \text{ for any } v \in \mathcal{R}^{-1}(v_i^{abs}). \end{aligned} \tag{9}$$

The mapping L^{abs} is well-defined because $L(v)$ is guaranteed to be the same no matter which $v \in \mathcal{R}^{-1}(v_i^{abs})$ is chosen, by definition of equivalence (7). Furthermore, the existence of T^{abs} is guaranteed because the equivalence classes form a partition of V and in the worst case, each state $v \in V$ would belong to a different equivalence class. In that case, T^{abs} would be identical to T. Computation of the abstraction follows directly the definition (cf. Algorithm 37 in [22]). Initial conditions can be adjusted simply defining $S_0^{abs}(a_n) \doteq \mathcal{R}(S_0(a_n))$ for each $a_n \in \mathcal{A}$.

Equivalence relation defined in (9) is the coarsest stutter bisimulation for T (see Lemma 7.96 in [22]) and abstraction T^{abs} is stutter bisimulation equivalent to T (see Theorem 7.102 in [22]) as stated by the following proposition:

Proposition 1 *Let T be a transition system and T^{abs} be its abstraction. For any T-path π, there exists a stutter trace equivalent T^{abs}-path π^{abs}. Conversely, for any T^{abs}-path π^{abs}, there exists a stutter trace equivalent T-path π.*

3.2 Higher Level Solution of Problem 1

After obtaining an abstraction, we solve a slightly different version of Problem 1 instance $(T^{abs}, \mathcal{A}, S_0^{abs}, \mu, 1)$ to generate a collection of paths $\{\pi_1^{abs}, \dots, \pi_N^{abs}\}$.

First, we do not enforce collision avoidance since each state in V^{abs} corresponds to a set of states in the original transition system and could hold more than a single agent. Second, we impose additional constraints coming from counter-examples. These constraints are explained in more detail in §3.4. We then form an integer linear program (ILP) as it is explained in [17]. Solving the subsequent feasibility problem generates a collection $\Pi^{abs} = \{\pi_n^{abs}, \dots, \pi_N^{abs}\}$ of T^{abs}-paths that 1-robustly satisfies μ, i.e., $\Pi^{abs} \models_1 \mu$. We call such a collection an *abstract plan*. We remind the reader that each *abstract path* π_n^{abs} has a prefix-suffix form as cLTL+$_{\backslash\bigcirc}$formulas are interpreted over infinite horizons, i.e., there exists a non-negative integer $l^{abs} < h^{abs}$ such that for all n

$$\pi_n^{abs} = \pi_n^{abs}(0) \dots \pi_n^{abs}(l^{abs})(\pi_n^{abs}(l^{abs}+1) \dots \pi_n^{abs}(h^{abs}))^{\omega}. \qquad (10)$$

In other words, each agent is assigned a lasso shaped path that can be traversed indefinitely. As shown in [22], if $(T^{abs}, \mathcal{A}, S_0^{abs}, \mu, 1)$ has a solution, there exists a large enough h^{abs} such that there exists a solution in the form of (10).

3.3 Lower Level Solution of Problem 1

Proposition 1 guarantees that, for each T^{abs}-path π_n^{abs}, there exists a stutter trace equivalent T-path π_n. Each state of the abstraction T^{abs} corresponds to a set of states in the original transition system T. If an agent moves one from one state to another in the abstraction, it needs to move from one region to another in the original transition system. By construction of the T^{abs}, the existence of a path between such two regions is guaranteed. However, a collection of these paths might be in collision. To generate a collection of collision-free and stutter trace equivalent T-paths, we solve a sequence of *generalized multi-agent path planning (GMAPP)* problems. Following is the formal definition of GMAPP that we use:

Problem 2 *Let a transition system* $T = (V, E, AP, L)$*, a set of agents* $\mathcal{A} = \{a_1, \dots, a_N\}$*, a time horizon* $h \in \mathbb{N}$ *and injective mappings* $x_I, x_G, X_I, X_G : \mathcal{A} \to 2^V$ *be given. Find a collection of collision-free* T*-paths* $\{\pi_1, \dots \pi_N\}$ *such that for all* $a_n \in \mathcal{A}$, $\pi_n(0) \in x_I(a_n)$, $\pi_n(h) \in x_G(a_n)$ *and for each agent there exists a positive integer* $0 < l_n < h$ *where* $\pi_n(t) \in X_I(a_n)$ *for all* $0 \le t \le l_n$ *and* $\pi_n(t) \in X_G(a_n)$ *for all* $l_n < t \le h$.

We characterize an instance of Problem 2 by a tuple $(T, \mathcal{A}, h, x_I, x_G, X_I, X_G)$. Each agent $a_n \in \mathcal{A}$ needs to start from state within $x_I(a_n) \subset X_I(a_n) \subset V$ and reach a state in $x_G(a_n) \subset X_G(a_n) \subset V$. While doing so, agent a_n should stay in set of states $X_I(a_n) \cup X_G(a_n)$ for all times and it should not return back to $X_I(a_n)$ once in $X_G(a_n)$. Furthermore, collisions with other agents must be avoided. The intuition here is that set of states $X_I(a_n)$ and $X_G(a_n)$ correspond to two consecutive states on an abstract path. We use $x_I(a_n)$ (and $x_G(a_n)$) in case initial (and final) state needs to be explicitly specified. As it moves from one abstract state to the other, to prevent jittering, an agent leaving X_I should not return back.

GMAPP is a generalization of the classical multi-agent path planning (MAPP) problem where one assigns a single initial state and a single goal state to each agent and assumes that the set of 'safe' states are same for all agents. Despite that, many efficient MAPP algorithms, such as [1, 3, 4], can easily be modified to accommodate for these differences. We use an ILP based method similar to [3] to solve GMAPP problems. For all $a_n \in \mathcal{A}$ and for all $0 \leq t \leq h^{abs}$, we set

$$\begin{gathered} x_I^0(a_n) = S_0(a_n), \quad X_I^t(a_n) = \mathcal{R}^{-1}(\pi_n^{abs}(t)), \\ x_G^t(a_n) = X_G^t(a_n) = \mathcal{R}^{-1}(\pi_n^{abs}(t+1)) \end{gathered} \tag{11}$$

Starting from $t = 0$, let $\{\beta_1^t, \ldots, \beta_N^t\}$ be a solution to $(T, \mathcal{A}, h, x_I^t, x_G^t, X_I^t, X_G^t)$. For all $t > 0$ and $a_n \in \mathcal{A}$, we set

$$x_I^t(a_n) = \beta_n^{t-1}(h) \tag{12}$$

and solve the next GMAPP instance. For the special case $t = h^{abs}$, we set $x_G^t(a_n) = \beta_n^{l^{abs}}(0)$ to 'close the loop'. If all GMAPP instances can be solved, we define for all integers $0 \leq t < h^{abs}$ and $0 \leq \alpha < h$

$$\pi_n(th + \alpha) \doteq \beta_n^t(\alpha), \quad \pi_n(h^{abs}h) \doteq \pi_n(lh). \tag{13}$$

Intuitively, π_n is concatenation of β_n^t. Each π_n, similar to π_n^{abs}, is in prefix-suffix form, i.e., $\pi_n = \pi_n(0), \ldots, \pi_n(lh) \left(\pi_n(lh+1), \ldots, \pi_n(h^{abs}h)\right)^\omega$. Note that $\pi_n(t)$ is well-defined for all t and is a valid T-path.

If $(\mathcal{A}, T, h, x_I^t, x_G^t, X_I^t, X_G^t)$ has no solutions for some t, we roll back and update

$$x_G^{t-1}(a_n) \leftarrow x_G^{t-1}(a_n) \setminus \{\beta_n^{t-1}(h)\}. \tag{14}$$

We call an abstract plan *infeasible* if instance $(T, \mathcal{A}, h, x_I^0, x_G^0, X_I^0, X_G^0)$ has no solutions. In that case, we generate a counter example that prevents the same abstract plan to be generated. Details of this process are explained in §3.5.

Following proposition shows that the method proposed in this paper is sound.

Proposition 2 *Given a Problem 1 instance $(T, \mathcal{A}, S_0, \mu, 0)$, assume the collection $\Pi = \{\pi_1, \ldots, \pi_N\}$ is generated by Algorithm 1. Then Π is a solution to $(T, \mathcal{A}, S_0, \mu, 0)$, that is, Π are collision-free, $\pi_n(0) = S_0(a_n)$ for all $a_n \in \mathcal{A}$, and $\Pi \models_0 \mu$.*

Proof. Showing $\{\pi_1, ..., \pi_N\}$ are collision-free is straight-forward. Each GMAPP instance generates collision-free T-paths. Concatenation of them would also be collision-free. Furthermore, $\pi_n(0) = S_0(a_n)$ for all $a_n \in \mathcal{A}$ due to (11).

Next we show $\{\pi_1, ..., \pi_N\} \models_0 \mu$. Let $\Sigma = \{\sigma_{\pi_1}, \ldots, \sigma_{\pi_N}\}$ and $\Sigma^{abs} = \{\sigma_{\pi_1^{abs}}, \ldots, \sigma_{\pi_N^{abs}}\}$. Define synchronous execution $K = \{k_1, \ldots, k_N\}$ such that $k_n(t) = t$ for all $t \geq 0$ and for all $n \in [N]$. We first show that there exists a 1-bounded asynchronous execution $K^{abs} = \{k_1^{abs}, ..., k_N^{abs}\}$ such that collective traces (Σ, K) and (Σ^{abs}, K^{abs}) are identical.

Note that $\beta_n^t(h) \in \mathcal{R}^{-1}(\pi_n^{abs}(t+1)))$ due to (11). Furthermore, $\pi_n(0) = S_0(a_n) \in \mathcal{R}^{-1}(\pi_n^{abs}(0))$. This implies $L(\beta_n^t(h)) = L(\pi_n(ht)) = L(\pi_n^{abs}(t))$ for all

$t \geq 0$ due to (9). Also note that, for all $n \in [N]$, there exists a non-negative integer $l_n^t \leq h$ such that $L(\beta_n^t(t)) = L(\beta_n^t(0))$ for all $t \leq l_n^t$ and $L(\beta_n^t(t)) = L(\beta_n^t(h))$ for all $l_n^t < t \leq h$ due to definition of Problem 2, and equations (9) and (11). Therefore $L(\pi_n(th + \alpha)) = L(\beta_n^t(\alpha)) = L(\pi_n^{abs}(t))$.

Now initialize $k_n^{abs}(0) \doteq 0$ for all $n \in [N]$. Then iteratively define local times for all integers $0 < \alpha \leq h$ and $t \geq 0$ as

$$k_n^{abs}(th+\alpha) \doteq \begin{cases} k_n^{abs}(th+\alpha-1) & \text{if } \alpha \leq l_n^t \\ k_n^{abs}(th+\alpha-1)+1 & \text{if } \alpha > l_n^t \end{cases} \tag{15}$$

Note that $k_n^{abs}(t)$ is well-defined for all $t \geq 0$ and $L(\pi_n(t)) = L(\pi_n^{abs}(k_n^{abs}(t)))$ for all t. This implies that (Σ, K) and (Σ^{abs}, K^{abs}) are identical collective traces.

Moreover it is guaranteed that $k_n^{abs}(t+h) = k_n^{abs}(t) + 1$ and $k_n^{abs}(th) = k_m^{abs}(th)$ for all pairs of $n, m \in [N]$ and for all $t \geq 0$. This implies that, the collection $K^{abs} = \{k_1^{abs}, \ldots, k_N^{abs}\}$ is a 1-bounded asynchronous execution. Since $\Sigma^{abs} \models_1 \mu$, we have $\Sigma^{abs}, K^{abs} \models \mu$. Thus, $\Sigma \models_0 \mu$. $\square$

3.4 Counter Examples

Given a collection of T^{abs}-paths $\{\tilde{\pi}_1^{abs}, \ldots, \tilde{\pi}_N^{abs}\}$, assume the lower-level algorithm failed. In that case, the following constraints are generated and added to the higher level:

$$\neg\left(\bigwedge_n \left(\bigwedge_t \pi_n^{abs}(t) = \tilde{\pi}_n^{abs}(t)\right)\right) \tag{16}$$

Constraint (16) imposes that the same abstract plan will not be encountered again. However, this method removes only one abstract plan at a time, which might be inefficient. Algorithm would converge faster if a number of infeasible abstract plans can be eliminated all at once, similar to Irreducibly Inconsistent Set idea in [12]. When lower level fails for an abstract plan $\{\tilde{\pi}_1^{abs}, \ldots, \tilde{\pi}_N^{abs}\}$, we generate instances $(\mathcal{A}, T, h, x_I^t, x_G^t, X_I^t, X_G^t)$ for all t such that $x_I^t(a_n) = X_I^t(a_n) = \tilde{\pi}_n^{abs}(t)$ and $x_G^t(a_n) = X_G^t(a_n) = \tilde{\pi}_n^{abs}(t)$. If $(\mathcal{A}, T, h, x_I^t, x_G^t, x_S^t)$ is infeasible, generate the following constraint to prevent such an abstract step from being generated in the future:

$$\bigwedge_i \left(\left(\bigwedge_n \pi_n^{abs}(i) = \tilde{\pi}_n^{abs}(t)\right) \implies \neg\left(\bigwedge_n \pi_n^{abs}(i+1) = \tilde{\pi}_n^{abs}(t+1)\right)\right). \tag{17}$$

Additionally, if more than $|V_i|$ agents are assigned to abstract state v_i^{abs} at any time, it is obvious that collisions cannot be avoided. We impose appropriate constraints to prevent such trivial counter examples.

Algorithm 1 Main algorithm

```
1: $input$: $(T, \mathcal{A}, S_0, \mu, 0), h^{abs}, h$
2: $Counter\_Examples \leftarrow \{\}$
3: $T^{abs} \leftarrow abstract(T)$
4: $top$:
5: if $is\_high\_level\_feasible(T^{abs}, \mathcal{A}, S_0^{abs}, \mu, 1, Counter\_Examples)$ then
6:     $\{\pi_1^{abs}, \dots, \pi_N^{abs}\} = high\_level(T^{abs}, \mathcal{A}, S_0^{abs}, \mu, 1, Counter\_Examples)$
7: else
8:     return Infeasible
9: $t \leftarrow 0$
10: $x_I^t(a_n) \leftarrow S_0(a_n)$
11: while $t < h^{abs}$ do
12:     $X_I^t(a_n) \leftarrow \pi_n^{abs}(t)$
13:     $x_G^t(a_n) \leftarrow X_G^t(a_n) \leftarrow \pi_n^{abs}(t+1)$
14:     $rollback$:
15:     if $is\_GMAPP\_feasible(\mathcal{A}, G, h, x_I^t, x_G^t, X_I^t, X_G^t)$ then
16:         $\{\beta_n^t\} = GMAPP(\mathcal{A}, G, h, x_I^t, x_G^t, X_I^t, X_G^t)$
17:         $x_I^{t+1}(a_n) \leftarrow \beta_n^t(h)$
18:         $t \leftarrow t+1$
19:     else
20:         if $t > 0$ then
21:             $x_G^t(a_n) \leftarrow x_G^{t-1}(a_n) \setminus \{x_I^t(a_n)\}$
22:             $t \leftarrow t-1$
23:             goto $rollback$
24:         else
25:             $Counter\_Examples \leftarrow Counter\_Examples \cup \{\pi_1^{abs}, \dots, \pi_N^{abs}\}$
26:             goto $top$
27: $\pi_n = concatenate(\beta_n^0, \dots, \beta_n^{h^{abs}})$
28: return $\{\pi_1, \dots, \pi_N\}$
```

3.5 Handling Asynchrony

This section shows how to deal with asynchrony. First, a small modification to Algorithm 1 necessary to solve Problem 1 for $\tau = 1$ is presented. Then, we show how correct behavior for a given specification can be obtained for an arbitrary τ by using an execution policy that restricts the possible executions.

Given $(T, \mathcal{A}, S_0, \mu, 0)$, assume an abstract plan is generated at the higher level. For all t and all pairs of $n, m \in [N]$, we enforce in each GMAPP

$$\beta_n(t+1) \neq \beta_m(t). \tag{18}$$

With this modification, when the asynchrony between agents is 1-bounded, these paths can be executed without collisions in an open-loop fashion, no communication or sensing needed at run-time. Furthermore, any 1-bounded asynchronous execution of collection $\{\pi_1, \dots, \pi_N\}$, generated according to (13), would satisfy the specification μ.

Under mild assumptions, these trajectories can be implemented such that the specifications are still satisfied and collisions are avoided for arbitrary τ.

Assume all agents can communicate with each other and can indefinitely stay in any state, i.e., $(v, v) \in E$ for all $v \in V$. Also assume that paths generated at the low level satisfy (18). Specification μ would be satisfied for all τ by $\{\pi_1, \dots, \pi_N\}$ when all agents use the following execution policy. If $\pi_n(t) = \pi_m(t')$ for some $t > t'$, agent a_n does not enter state $\pi_n(t)$ until agent a_m reaches $\pi_m(t'+1)$. Otherwise, a_n moves to the subsequent state on its path. Note that, generated paths might not be collision-free under τ-bounded asynchrony. Nonetheless, the policy above would prevent collisions and would not result in deadlock as shown in [4]. We further require that agents 'synchronize at abstract steps', meaning that agent a_n move to $\pi_n(ht+1)$ only after all agents a_m reach $\pi_m(ht)$.

Remark 1 (Termination) *Algorithm 1 is guaranteed to terminate as the number of abstract plans for a given h^{abs} is finite. If a solution does not exist for a certain h^{abs}, the higher-level problem will eventually become infeasible as more counter examples are generated and the algorithm will terminate.*

Remark 2 (Complexity) *Note that the complexity of the method proposed in this paper does not depend on τ. The higher-level problem is solved for $\tau = 1$ and this is enough to satisfy the specification for any τ as long as agents avoid collisions and synchronize at abstract steps. As a comparison, previous work in [17] introduces $O(N(h + |S|)\tau)$ additional decision variables and $O(N^2 h|S|\tau)$ additional constraints where N is the number of agents, h is the solution horizon, and $|S|$ is the number of states. Moreover, thanks to the use of abstraction, $|S^{abs}|$ is smaller than $|S|$ and horizon h^{abs} required to find a solution is smaller than h required for the non-hierarchical method.*

4 Results

All experiments are run on a laptop with 2.5 GHz Intel Core i7 and 16 GB RAM. Gurobi [23] is used as the underlying ILP solver. Our code is accessible at https://github.com/sahiny/cLTL-hierarchical.

Table 1. Run-time comparison (seconds)

	SMC-based [12]	**Hierarchical**			
N	(Fig 1, μ)	(Fig 1, μ)	(Fig 1, μ')	(Fig 2, μ)	(Fig 2, μ')
4	444.35	92.52	121.69	95.16	86.85
6	timeout	236.74	439.41	199.24	242.35
8	timeout	507.97	619.02	664.94	729.58
10	timeout	801.64	1665.95	1139.82	1275.62
12	timeout	1727.47	2109.11	1499.17	2114.63

Example 1: We borrow the multi-robot scenario from [12] where the workspace is illustrated in Figure 1. For each trial N agents are randomly initialized from the region marked with x_I, and specifications are given as:

$$\mu = \Box\Diamond[r_1, N] \wedge \Box\Diamond[r_2, N/2] \wedge \Box\Diamond[r_3, N/2]. \tag{19}$$

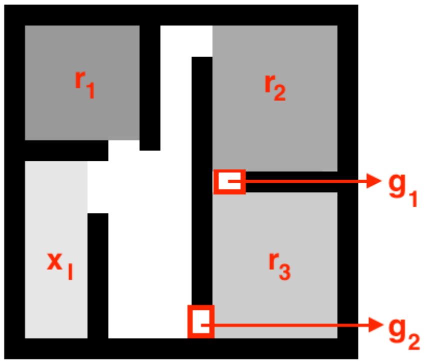

Fig. 1. Workspace taken from [12]

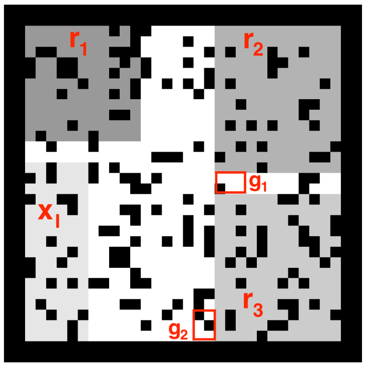

Fig. 2. A sample workspace with randomly generated obstacles

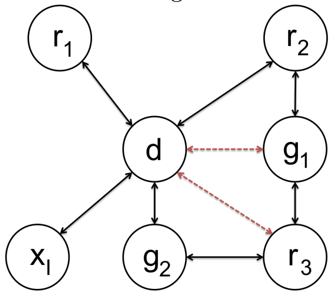

Fig. 3. Abstraction obtained for Figure 1 (solid arrows), and Figure 2 (solid and dashed arrows)

In words, we require all agents to repeatedly (infinitely many times) meet at r_1. Similarly, at least half of the robots should repeatedly meet both at r_2 and r_3. In [12], robot dynamics are modeled as chains of integrators and a satisfiability modulo convex (SMC) programming based method is proposed. The method proposed in this paper does not directly handle continuous dynamics. Hence, we grid the workspace into 30 by 30 squares of same size. Agents are allowed to move horizontally or vertically to neighboring states or stay in their current position. Note that this behavior is consistent with the continuous dynamics. The abstract transition system for this example is illustrated in Figure 3. We solve the problem for increasing N. Computation times averaged over 10 trials are shown in Table 1. SMC-based method in [12] can solve the problem only up to $N = 5$ agents under 30 minutes as it can be seen from the second column. Hierarchical method proposed in this paper, on the other hand, can solve the same problem up to $N = 12$ agents. We also remind the reader that, if agents are allowed to communicate, solutions generated with our method would be robust to arbitrary synchronization errors while [12] might lead to collisions and/or violation of the specification μ. We also note that the non-hierarchical approach in [17] timed out in all these instances due to the large number of states and resulting large horizon required for feasibility.

Example 2: We then modify the specifications and add the additional constraint that region marked with r_3 should be empty until both g_1 and g_2 are populated with at least one robot at the same time:

$$\mu' = \mu \wedge (\neg[r_3, 1]\, \mathcal{U}\ ([g_1, 1] \wedge [g_2, 1])) \tag{20}$$

Note that [12] cannot handle arbitrary cLTL+$_{\backslash\bigcirc}$formulas and expressing the same specification using regular LTL is not trivial. While any cLTL+$_{\backslash\bigcirc}$formula can be transformed into regular LTL as in [17], the length of the LTL formula specifying the same task could be exponentially longer, significantly increasing the computation times. Computation times for varying N are shown in Table 1.

Example 3: Next, we keep x_I, r_1, r_2, r_3, g_1 and g_2 as they are and randomly select 20% of the states as obstacles. The abstract transition system for this example is illustrated again in Figure 3. Computation times for varying number of agents, and both specifications μ and μ' are again shown in Table 1. A video simulating the synthesized plans with synchronized agents can be seen at https://www.youtube.com/watch?v=SrPDQMRmcNU. We then assume same plans are executed asynchronously. At each step agents are delayed with $p = 0.3$ probability. Using the policy proposed in §3.5, agents are able to satisfy the specifications while avoiding collisions, as it can be seen from https://www.youtube.com/watch?v=xO8xK9pXUKI.

5 Conclusions

In this paper, we proposed a method to generate multi-agent trajectories to satisfy properties given in counting temporal logic. This method is hierarchical and introduces a trade-off between solving logic constraints and path planning. Using a smaller abstraction, high-level plans can be generated faster and more complex specifications can be handled. On the other hand, a more refined abstraction can be used if lower-level path generation is the bottleneck, as it results in easier multi-agent reachability problems. In future work, we would like to examine this trade-off between using a coarser or a more refined abstraction in a rigorous way. Additionally, we use an ILP based method to generate lower-level paths which is not optimized. We plan to implement a more efficient lower-level path generation algorithm to decrease solution times.

Acknowledgments: This work is supported in part by NSF grants CNS-1446298 and ECCS-1553873, and DARPA grant N66001-14-1-4045.

References

1. Surynek, P.: A novel approach to path planning for multiple robots in bi-connected graphs. In: Proc. of ICRA. pp. 3613–3619. IEEE (2009)

2. de Wilde, B., ter Mors, A.W., Witteveen, C.: Push and rotate: cooperative multi-agent path planning. In: Proc. of AAMAS. pp. 87–94 (2013)
3. Yu, J., LaValle, S.M.: Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics. IEEE Trans. on Robotics 32(5), 1163–1177 (2016)
4. Ma, H., Kumar, T.S., Koenig, S.: Multi-agent path finding with delay probabilities. In: AAAI. pp. 3605–3612 (2017)
5. Ren, W., Beard, R.W., Atkins, E.M.: A survey of consensus problems in multi-agent coordination. In: American Control Conference, 2005. Proceedings of the 2005. pp. 1859–1864. IEEE (2005)
6. Li, Z., Wen, G., Duan, Z., Ren, W.: Designing fully distributed consensus protocols for linear multi-agent systems with directed graphs. IEEE Transactions on Automatic Control 60(4), 1152–1157 (2015)
7. Kloetzer, M., Belta, C.: Temporal logic planning and control of robotic swarms by hierarchical abstractions. IEEE Trans. Robotics 23(2), 320–330 (2007)
8. Kloetzer, M., Belta, C.: Automatic deployment of distributed teams of robots from temporal logic motion specifications. IEEE Trans. Robotics 26(1), 48–61 (2010)
9. Karaman, S., Frazzoli, E.: Linear temporal logic vehicle routing with applications to multi-uav mission planning. International Journal of Robust and Nonlinear Control 21(12), 1372–1395 (2011)
10. Guo, M., Dimarogonas, D.V.: Multi-agent plan reconfiguration under local ltl specifications. The International Journal of Robotics Research 34(2), 218–235 (2015)
11. Haghighi, I., Sadraddini, S., Belta, C.: Robotic swarm control from spatio-temporal specifications. In: Proc. of CDC. pp. 5708–5713 (Dec 2016)
12. Shoukry, Y., Nuzzo, P., Balkan, A., Saha, I., Sangiovanni-Vincentelli, A.L., Seshia, S.A., Pappas, G.J., Tabuada, P.: Linear temporal logic motion planning for teams of underactuated robots using satisfiability modulo convex programming. In: Proc. of CDC. pp. 1132–1137. IEEE (2017)
13. Desai, A., Saha, I., Yang, J., Qadeer, S., Seshia, S.A.: Drona: A framework for safe distributed mobile robotics. In: Proc. of ICCPS. pp. 239–248. ICCPS '17, ACM, New York, NY, USA (2017), http://doi.acm.org/10.1145/3055004.3055022
14. Kantaros, Y., Zavlanos, M.M.: Distributed optimal control synthesis for multi-robot systems under global temporal tasks. In: Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems. pp. 162–173. IEEE Press (2018)
15. Bhatia, A., Kavraki, L.E., Vardi, M.Y.: Sampling-based motion planning with temporal goals. In: Proc. of ICRA. pp. 2689–2696 (2010)
16. Sahin, Y.E., Nilsson, P., Ozay, N.: Provably-correct coordination of large collections of agents with counting temporal logic constraints. In: Proc. of ICCPS. pp. 249–258. ACM (2017)
17. Sahin, Y.E., Nilsson, P., Ozay, N.: Synchronous and asynchronous multi-agent coordination with cltl+ constraints. In: Proc. of CDC. pp. 335–342. IEEE (2017)
18. Gray, A., Gao, Y., Lin, T., Hedrick, J.K., Tseng, H.E., Borrelli, F.: Predictive control for agile semi-autonomous ground vehicles using motion primitives. In: Proc. of ACC. pp. 4239–4244. IEEE (2012)
19. Paranjape, A.A., Meier, K.C., Shi, X., Chung, S.J., Hutchinson, S.: Motion primitives and 3d path planning for fast flight through a forest. The International Journal of Robotics Research 34(3), 357–377 (2015)
20. Pola, G., Girard, A., Tabuada, P.: Approximately bisimilar symbolic models for nonlinear control systems. Automatica 44(10), 2508–2516 (2008)

21. Wongpiromsarn, T., Topcu, U., Ozay, N., Xu, H., Murray, R.M.: Tulip: a software toolbox for receding horizon temporal logic planning. In: Proceedings of the 14th international conference on Hybrid systems: computation and control. pp. 313–314. ACM (2011)
22. Baier, C., Katoen, J.: Principles of Model Checking. MIT Press (1999)
23. Gurobi Optimization, I.: Gurobi optimizer reference manual (2016), http://www.gurobi.com