Reconfiguration in Motion Planning of Single- and Multi-agent Systems under Infeasible Local LTL Specifications

Meng Guo and Dimos V. Dimarogonas

Abstract—A reconfiguration method for the model-checking-based motion planning of single- and multi-agent systems under infeasible local LTL specifications is proposed. The method describes how to synthesize the motion plan that fulfills the infeasible task specification the most, and how the infeasible task specification is relaxed. The novelty is the introduction of a metric within the atomic proposition domain, and the relative weighting between the implementation cost of a motion plan and its distance to the original specification. For multi-agent systems, a dependency relation and relative priorities are incorporated when the tasks are assigned independently to each agent. Simulations are presented to illustrate the method.

I. INTRODUCTION

Temporal-logic-based motion planning provides a fully automated correct-by-design controller synthesis approach for autonomous robots. Temporal logics such as Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) provide formal high level languages that can describe planning objectives more complex than the well-studied point-to-point navigation [14], [25], [27]. In this paper, we follow an approach that has gained significant popularity in recent years. The task specification is given as an LTL formula with respect to a discretized abstraction of the robot motion [1], [6], [19], [30]. Then a high-level discrete plan is found by off-the-shelf model-checking algorithms given the finite transition system and the task specification [2], [3], [11], [18]. This plan is then implemented through the corresponding low-level hybrid controller [10], [20], [24].

As stressed in [16], [17], [31], the above motion planning framework reports a failure when the given task specification is not realizable in the current workspace and under the agent dynamics. It is desired that users could get feedbacks about why the planning has failed and how to resolve this failure. This problem is addressed by [16] and [17] for single-agent systems by a systematic way to find the relaxed specification that is closest to the original one and can be fulfilled by the system. Detailed comparisons between our work and [17] can be found at the beginning of Section III. In short, this paper emphasizes mainly how to synthesize the motion plan that fulfills the infeasible task specification the most, and how the task specification is relaxed. [31] introduces a way to analyze the environment and system components contained in the infeasible specification, and identify the possible cause.

On the other hand, this work complements the topic about revising the motion plan under fixed LTL specifications when the workspace model or agent dynamics are updated, like in the cases of real-time revising [13] and local “patching” [26].

More importantly, we investigate the reconfiguration problem within the same framework also for multi-agent systems. Many existing works [10], [15], [32] consider the problem of decomposing a global specification to bisimilar local ones in a top-down manner. We, from an opposite viewpoint, assume that the local task specifications are assigned independently to each agent and there is no specified global task. The joined execution of these tasks may not be mutually feasible even if the individual one is. A decentralized solution is proposed to synthesize the individual motion plans that violate the mutual specification the least. Moreover, the priorities among the agents play an important role in the reconfiguration for multi-agent systems. This issue was indicated in our earlier work [12] where a framework for decentralized verification from local LTL specifications is proposed. However the way to resolve the conflicting specifications is not considered there. Real-time replanning for multi-vehicle networks is considered in [4] under safety constraints.

The main contribution of this work is the proposal of a generic framework to reconfigure the infeasible task specifications for both single- and multi-agent systems. In particular, the motion plans that fulfill the infeasible specifications the most are obtained. We allow the user-defined choice of the relative weighting between the implementation cost of the motion plan and how much this plan fulfills the original task specification. Multi-agent systems are also exploited and a decentralized approach is proposed by considering the dependency and priority relations.

The rest of the paper is organized as follows: Section II briefly introduces the model-checking-based motion planning. In Section III, we discuss the reconfiguration problem for single-agent systems. Section IV extends the results to multi-agent systems under local infeasible LTL specifications. Numerical simulations are presented in Section VI.

II. MODEL-CHECKING-BASED MOTION PLANNING

A. Task Specification in LTL

We focus on the task specification \( \phi \) given as an Linear Temporal Logic (LTL) formula. The basic ingredients of an LTL formula are a set of atomic propositions (APs) and several boolean and temporal operators. LTL formulas are formed according to the following grammar [3]: \( \phi := \text{true} | a | \phi_1 \land \phi_2 | \neg \phi | \bigcirc \phi | \phi_1 \cup \phi_2 \), where \( a \in AP \) and \( \bigcirc (\text{next}), \cup (\text{until}) \). For brevity, we omit the derivations.
of other useful operators like \( \square \) (always), \( \diamond \) (eventually), \( \Rightarrow \) (implication) and refer the readers to Chapter 5 of [3].

Given an LTL formula \( \varphi \over AP \), there is a union of infinite words that satisfy \( \varphi \): \( \text{Words}(\varphi) = \{ \sigma \in (2^{AP})^\omega | \sigma \models \varphi \} \), where \( \models \) is an LTL formula over the same alphabets \( \chi \). There are \( N \) states \( \Pi =\{ \pi_0, \pi_1, \ldots, \pi_N \} \). These regions can be in different shapes, such as points of interests [21], triangles [6], polygons [1] and hexagons [29]. There are different cell decomposition schemes available, depending on the robot dynamics and associated control approaches, see [1], [2], [10] and [13]. Formally the control-driven finite transition system (FTS) is defined below:

**Definition 1 (Control-driven FTS):** The control-driven FTS is a tuple \( \mathcal{T} = (\Pi, \to c, \Pi_0, AP, L, W_c) \), where \( \Pi = \{ \text{the robot is in region } \pi_i, i = 1, 2, \ldots, N \} \); \( \to c \subseteq \Pi \times \Pi = \text{the transition relation}; \Pi_0 \subseteq \Pi = \text{the set of initial states}; AP = \text{the set of APs}; L : \Pi \to 2^{AP} \) is a labeling function, giving the subset of \( AP \) which are true at state \( \pi_i; W_c : \rightarrow c \rightarrow \mathbb{R}^+ \) reflects the implementation cost (time or energy) of each transition.

We assume that \( \mathcal{T} \) does not have a terminal state [3]. An infinite path of \( \mathcal{T} \) is an infinite sequence of states \( \tau = \pi_0 \pi_1 \pi_2 \ldots \) such that \( (\pi_i, \pi_{i+1}) \in \to c \) for all \( i > 0 \). Its trace is the sequence of APs that are true at the states along the path, i.e., \( \text{trace}(\tau) = L(\pi_0) L(\pi_1) L(\pi_2) \ldots \). Given \( \varphi \) is an LTL formula over the same AP the satisfaction relation \( \tau \models \varphi \) if and only if \( \text{trace}(\tau) \in \text{Words}(\varphi) \). The infinite path \( \tau \) that satisfies \( \varphi \) is called a motion plan for the task \( \varphi \).

**C. Motion Plan Synthesis**

A valid motion plan \( \tau \) can be found by checking the emptiness of the product Büchi automaton, see [9] and Algorithm 11 in [3]. The product Büchi automaton is defined as \( \mathcal{A}_G = \mathcal{T} \times \mathcal{A}_\varphi = (Q, 2^{AP}, \delta, Q_0, F) \), where \( Q = \Pi \times Q \); \( Q_0 = \Pi_0 \times Q_0 \) are the accepting states; \( \delta \subseteq Q \times Q \) is the transition relation.

\( (\langle \pi_i, q_m \rangle, \langle \pi_j, q_n \rangle) \in \delta \) if and only if \( (\pi_i, \pi_j) \in \to c \) and \( (q_m, L(\pi_i), q_n) \in \delta \). There exists a motion plan satisfying \( \varphi \) if and only if \( \mathcal{A}_G \) has at least one accepting run [3].

**Lemma 1 (Feasibility and Projection):** An LTL specification \( \varphi \) is feasible over the FTS \( \mathcal{T} \) if and only if \( \mathcal{A}_G = \mathcal{T} \times \mathcal{A}_\varphi \) has an accepting run. Furthermore, for any accepting run \( R = (\pi_0, q_0) (\pi_1, q_1) \ldots \) of \( \mathcal{A}_G \), its projection onto \( \mathcal{T} \) the sequence \( \tau = \pi_0 \pi_1 \ldots \) satisfies \( \varphi \) [33].

The lower-level hybrid controller [6] that implements the motion plan is synthesized by executing the controllers associated with the transitions along the motion plan.

**III. RECONFIGURATION OF SINGLE-AGENT SYSTEMS**

An intriguing question to ask about the framework introduced in Section II is what if the given task specification is not feasible. How should the specification be relaxed and more importantly how to synthesize the motion plan that satisfies the relaxed specification, while at the same time violating the original specification the least possible?

An approximate algorithm is provided in [17] that partially answers the above question. It generates a relaxed specification automaton \( \mathcal{A}_G' \) which is close to \( \mathcal{A}_G \) (see Section III-C [17]). Then a motion plan can be synthesized by following the procedure as described in Section II-C. However there are often more than one accepting run within \( \mathcal{T} \times \mathcal{A}_\varphi \), and they may fulfill the original \( \varphi \) to different extents. We instead aim to find the motion plan that fulfills \( \varphi \) the most with respect to certain criterion, based on which then the relaxed specification is constructed.

**A. Relaxed Product Automaton**

Since \( \varphi \) is infeasible and \( \mathcal{A}_G \) does not have an accepting run by Definition 1, we need to relax the constraints imposed by \( \mathcal{A}_G \) to allow more transitions within \( \mathcal{A}_G \).

**Definition 2 (Relaxed Product Automaton):** The relaxed product Büchi automaton \( \mathcal{A}_G = \mathcal{T} \times \mathcal{A}_\varphi = (Q', 2^{AP}, \delta', Q'_0, F', W_r) \) is defined as follows:

- \( Q' = \Pi \times Q \) and \( q' = (\pi, q) \), \( \forall \pi \in \Pi \) and \( \forall q \in Q \).
- \( 2^{AP} \) is an alphabet: \( AP = \{ a_1, a_2, \ldots, a_K \} \).
- \( \delta' \subseteq Q' \times Q' \): \( (\pi_i, q_m), (\pi_j, q_n) \in \delta' \) if \( (\pi_i, \pi_j) \in \to c \) and \( l \in 2^{AP} \) such that \( (q_m, l, q_n) \in \delta \).
- \( Q'_0 = \Pi_0 \times Q_0 \) is the set of initial states.
- \( F' = \Pi \times F \) is the set of accepting states.
- \( W_r : \delta' \rightarrow \mathbb{R}^+ \) is the weight function to be defined.

Two differences between \( \mathcal{A}_G \) and \( \mathcal{A}_G \) defined in Section II-C are: (i) the constraint \( "q_m, L(\pi_i), q_n \in \delta'" \) when defining
\( \delta_p \) is relaxed to \( \exists l \in 2^{AP} \) such that \((q_m, l, q_n) \in \delta'\) when defining \( \delta' \) here; (ii) the weight function \( W_r \) is only introduced for \( A_r \). Firstly we introduce the evaluation function \( \text{Eval}: 2^{AP} \rightarrow \{0, 1\}^K \):

\[
\text{Eval}(l) = \nu \iff [\nu_i] = \begin{cases} 1 & \text{if } a_i \in l, \\ 0 & \text{if } a_i \notin l, \end{cases} \quad (2)
\]

where \( i = 1, 2, \ldots, K, l \in 2^{AP} \) and \( \nu \in \{0, 1\}^K \). Then a metric \((2^{AP}, \rho)\) is defined as

\[
\rho(l, l') = \|\nu - \nu'\|_1 = \sum_{i=1}^{K} |\nu_i - \nu'_i|, \quad (3)
\]

where \( \nu = \text{Eval}(l), \nu' = \text{Eval}(l') \) and \( l, l' \in 2^{AP}, \|\cdot\|_1 \) is the \( \ell_1 \) norm [7]. Then we could define the distance between an element \( l \in 2^{AP} \) to a set \( \chi \subseteq 2^{AP}(\chi \neq \emptyset) \) [7]:

\[
\text{Dist}(l, \chi) = \begin{cases} 0 & \text{if } l \in \chi, \\ \min_{\nu' \in \chi} \rho(l, l') & \text{otherwise}. \end{cases} \quad (4)
\]

Note that \( \text{Dist}(l, \chi) \) is not defined for \( \chi = \emptyset \). An example of computing the \( \text{Dist} \) function is given in Figure 1. Now we give the formal definition of \( W_r \) of \( A_r \):

\[
W_r((\langle \pi_i, q_m \rangle, \langle \pi_j, q_n \rangle)) = W_c(\pi_i, \pi_j) + \alpha \cdot \text{Dist}(L(\pi_i), \chi(q_m, q_n)), \quad (5)
\]

where \((\langle \pi_i, q_m \rangle, \langle \pi_j, q_n \rangle) \in \delta'; \alpha \geq 0 \) is a design parameter; \( \chi(q_m, q_n) = \{ l \in 2^{AP} | (q_m, l, q_n) \in \delta \} \) consists of all input alphabets that enable the transition from \( q_m \) to \( q_n \) in \( A_r \). Since by Definition 2 there exists \( l \in 2^{AP} \) that \((q_m, l, q_n) \in \delta, \chi(q_m, q_n) \neq \emptyset \) is ensured. \( W_c(\pi_i, \pi_j) \) is the implementation cost of the transition from \( \pi_i \) to \( \pi_j \) in \( T \). \( \text{Dist}(L(\pi_i), \chi(q_m, q_n)) \) measures how much the transition from \( \pi_i \) to \( \pi_j \) violates the constraints imposed by the transition from \( q_m \) to \( q_n \). Being 0 means that \( A_r \) is not violated, while the larger the distance is the more \( A_r \) is violated. The design parameter \( \alpha \) is used to reflect the relative penalty on violating the original specification, and also the user’s preference on a motion plan that has less implement cost or that fulfills the task specification more. The penalty on violating \( A_r \) is increased when \( \alpha \) is larger.

**B. Problem Statement**

Note that \( A_r \) is more connected than the conventional product automaton \( A_p \) in section II-C. Since \( A_p \) does not have an accepting run, we instead search for an accepting run within \( A_r \). However the existence of an accepting run alone is not enough because: (i) they have different implementation costs; (ii) we would like to measure how much they violate the original specification. Thus we consider the accepting runs with the following prefix-suffix structure:

\[
R = q_0' q_1' \cdots [q_k' q_{k+1}' \cdots q_n']^\omega
= (\langle \pi_0, q_0 \rangle \langle \pi_1, q_1 \rangle \cdots \langle \pi_k, q_k \rangle \cdots \langle \pi_n, q_n \rangle)^\omega, \quad (6)
\]

where \( q_0' = \langle \pi_0, q_0 \rangle \in Q_0' \) and \( q_k' = \langle \pi_k, q_k \rangle \in F'. \)

Note that there are no correspondences among the subscripts. Clearly \( R \) consists of two parts: the prefix part \((q_0' q_1' \cdots q_k')\) from an initial state \( q_0' \) to one accepting state \( q_k' \) that is executed only once and the suffix part \((q_k' q_{k+1}' \cdots q_n')\) from \( q_k' \) back to itself that is repeated infinitely. An accepting run with the prefix-suffix structure has a finite representation as (6), and more importantly it allows us to define the total cost of an accepting run (similar to Definition 4.5 in [33]):

\[
\text{Cost}(R) = \sum_{i=0}^{k-1} W_r(q_i', q_{i+1}) + \gamma \sum_{i=k}^{n-1} W_r(q_i', q_{i+1}) = \text{cost}_r + \alpha \cdot \text{dist}_\varphi, \quad (7)
\]

where \( \text{cost}_r = (\sum_{i=0}^{k-1} + \gamma \sum_{i=k}^{n-1}) W_c(\pi_i, \pi_{i+1}) \) is the accumulated implementation cost of the motion plan \( r \), i.e., the projection of \( R \) onto \( T \); \( \text{dist}_\varphi = (\sum_{i=0}^{k-1} + \gamma \sum_{i=k}^{n-1}) \text{Dist}(L(\pi_i), \chi(q_i, q_{i+1})) \) is the accumulated distance of \( r \) to \( A_r \). The first summation in (7) represents the accumulated weights of transitions along the prefix and the second is the summation along the suffix. Note that \( \gamma \geq 0 \) represents the relative weighting on the cost of transient response (the prefix) and steady response (the suffix) to the task specification [33].

The prefix-suffix structure is more of a way to formulate the total cost of an accepting run, rather than a conservative assumption. If an accepting run exists, by its definition at least one accepting state should appear infinitely often. Among all the finite number of cycles starting for this accepting state and back to itself there is one with the minimal cost. Thus an accepting run of the form (6) can be built using this minimal cycle as the periodic suffix. Now we would like to state the problem for single-agent systems:

**Problem 1:** Given an infeasible specification \( \varphi \) over the FTS \( T \), find the accepting run of \( A_r \) that minimizes the cost by (7) and the corresponding motion plan \( r \).

Given \( A_r \) and a value of \( \alpha \), we call the solution to Problem 1 as the optimal accepting run \( R_{opt} \) under that \( \alpha \). Algorithm 1 takes as input arguments the weighted state graph [3] \( G(A_r) = (Q', \delta', W_r) \), the set of initial vertices \( I = Q_0' \) and the set of accepting vertices \( F = F' \). It utilizes Dijkstra’s algorithm [25] for computing the shortest path between pairs of vertices within a graph. In particular, denote the number of elements in \( I \) and \( F \) by \( |I| = L \) and \( |F| = M \). Function \( \text{MinPath} \) takes \((G, I, F)\) as inputs and outputs a \( L \times M \) matrix \( D_{IF} \), with the \((i_{th}, j_{th})\) element containing the value of the minimal cost from \( I_i \) to \( F_j \); and a \( L \times M \) cell \( P_{IF} \), with the \((i_{th}, j_{th})\) cell containing the sequence of vertices appearing in the path with minimal cost from \( I_i \) to \( F_j \). Function \( \text{MinCycl} \) is a variant of function \( \text{MinPath} \),
Algorithm 1: Function optRun(G, I, F)

Input: a weighted graph G, I, F.

Output: the optimal accepting run R_{opt}.

1. Compute the path with minimal cost from every initial vertex in I to every accepting vertex in F.
   \((D_{IF}, P_{IF}) = \text{MinPath}(G, I, F)\).

2. Compute the path with minimal cost from every accepting vertex in F and back to itself:
   \((D_{FF}, P_{FF}) = \text{MinCycl}(G, F)\).

3. For each column of \(D_{IF}\), find the element with the minimal value and the corresponding cell in \(P_{IF}\) (with the same index). Save them sequentially in \(1 \times M\) matrix \(D_{IF}\) and \(1 \times M\) cell \(P_{IF}\).

4. Find the element with the minimal value in \(D_{IF} + \gamma D_{FF}\) and its index \(j_{\text{min}}\).

5. Optimal accepting run \(R_{opt}\), prefix: the \(j_{\text{min}}\)-th element of \(P_{IF}\); suffix: the \(j_{\text{min}}\)-th element of \(P_{FF}\).

which outputs a \(1 \times M\) matrix \(D_{FF}\), with the \(j_{th}\) element containing the value of the minimal cost from \(F_j\) back to \(F_j\); and a \(1 \times M\) cell \(P_{FF}\) with the \(j_{th}\) cell containing the sequence of vertices appearing in the path with minimal cost from \(F_j\) back to \(F_j\) (as in Figure 2). Note that if a vertex is not reachable from another vertex, then the cost is \(+\infty\).

C. Motion Plan and Feedback

Algorithm 1 provides an optimal accepting run \(R_{opt}\) once \(\alpha\) is chosen in \(A_\varphi\). Then Algorithm 2 takes as inputs \(R_{opt}\), the FTS \(\mathcal{T}\) and the original specification automaton \(A_\varphi\). While iterating through the transitions along \(R_{opt}\) in sequence, it projects \(R_{opt}\) into \(\mathcal{T}\) to obtain the corresponding motion plan \(\tau\); it constructs the revised specification automaton \(A_\varphi'\) by adding new transitions to \(A_\varphi\) (as shown in Figure 1); it computes the implementation cost \(\text{cost}_\tau\) and the accumulated distance to \(A_\varphi\) \(\text{dist}_\varphi\) defined in (7). It can be verified that the obtained \(A_\varphi'\) is a valid relaxation of \(A_\varphi\) [16]. Note each accepting run \(R_{opt}\) corresponds to a unique motion plan \(\tau\) and a revised specification automaton \(A_\varphi'\).

Remark 1: Although \(A_\varphi\) may allow more transitions compared with \(A_p\), any run of \(A_\varphi\) can be projected onto \(\mathcal{T}\), resulting in a valid path of \(\mathcal{T}\). Namely, the transition relation of \(\mathcal{T}\) is never relaxed when constructing \(A_\varphi\). Thus the motion plan derived from Algorithm 2 is always implementable.

Lemma 2: Assume \(\tau\) and \(\text{dist}_\varphi\) are the derived from Algorithm 2. Then \(\text{dist}_\varphi = 0\) implies that \(\tau\) satisfies \(\varphi\).

Proof: Since \(\text{Dist}(\cdot) \geq 0\), the accumulated distance \(\text{dist}_\varphi = 0\) implies \((q_m, L(\pi), q_n) \in \delta\) for all transitions \((\langle \pi, q_m \rangle, \langle \pi', q_n \rangle)\) along the optimal accepting run \(R_{opt}\). Thus \(R_{opt}\) is an accepting run for the un-relaxed product automaton \(A_p\). Its projection \(\tau\) satisfies \(\varphi\) by Lemma 1.

Algorithms 1 and 2 solve Problem 1 under a given \(\alpha\). However it may not be trivial to determine the appropriate value of \(\alpha\). As an extension, Algorithm 1 could be called under different \(\alpha\) to generate various optimal accepting runs, among which the unique ones are saved as the optimal accepting run candidates. Then for each optimal run, Algorithm 2 is called to compute the corresponding motion plan \(\tau\) and the associated \(\text{cost}_\tau, \text{dist}_\varphi\) as the feedback.

Remark 2: The proposed method can be applied directly when \(\varphi\) is feasible over \(\mathcal{T}\) without any modification. This is due to that when \(\alpha\) is large enough, i.e., the penalty on violating \(A_\varphi\) is severe, Algorithm 1 will automatically select the accepting run that satisfies \(\varphi\).

An example system is shown in Figure 3. The agent has to go from region \(\pi_0\) to \(\pi_3\) and stay there, at the same time avoid all regions satisfying properties \(a_2\) and \(a_3\). Three alternative motion plans are obtained by varying \(\alpha\), as shown in Figure 4: (i) when the penalty on violating \(\varphi\) is low, \(A_\varphi\) is revised by adding \((q_0, \emptyset, q_1)\) and \((q_1, \emptyset, q_1)\) to \(\delta\) and the corresponding motion plan is \(\pi_0\pi_1\pi_2\pi_3\) (black hexagram, \(\text{cost}_\tau = 65, \text{dist}_\varphi = 2\)); (ii) when the penalty is increased, \(A_\varphi\) is revised by adding \((q_0, \{a_2\}, q_1)\) to \(\delta\), where the motion plan is \(\pi_0\pi_1\pi_3\pi_2\) (cyan triangle, \(\text{cost}_\tau = 85, \text{dist}_\varphi = 1\)). Note that in plan (iii) the agent passes through \(\pi_2\) which satisfies only \(a_2\), instead of \(\pi_1\) which satisfies both \(a_2\) and \(a_3\).

IV. RECONFIGURATION FOR MULTI-AGENT SYSTEMS

As mentioned in the introduction, the reconfiguration of multi-agent systems under local infeasible LTL specifications is more difficult than the single-agent case, due to the following reasons: (i) the joined execution of multiple agents’ tasks may not be mutually feasible even though the individual one is; (ii) the priority of each agent plays an important role when deciding whose tasks should be changed. The first
aspect is because these tasks are assigned independently and some cooperative tasks have not been fully agreed before the deployment. The second aspect is because some agents’ tasks are safety or security critical and have to be fulfilled all the time, meaning that other agents have to comply when there are conflicts.

Assume the system we consider consists of \( N \) agents, denoted by agent \( i = 1, 2, \ldots, N \). Moreover, we denote the finite transition system of agent \( i \) by

\[
T_i = (\Pi_i, \rightarrow_i, \Pi_i \cup AP_i, L_i, W_i),
\]

its LTL specification by \( \varphi_i \); the specification automaton by

\[
A_{\varphi_i} = (Q_i, 2^{AP_{\varphi_i}}, \delta_i, Q_i \cup AP_{\varphi_i}, \mathcal{F}_i),
\]

where \( \chi_i(q_j, q_j') = \{ l \in 2^{AP_{\varphi_i}} | (q_j, l, q_j') \in \delta_i \} \).

For brevity, we omit the formal definition of all notations above but they follow the same structure as \( T \) and \( A_{\varphi_i} \) introduced in Section II. \( T_i \) abstracts agent \( i \)'s behavior within its workspace \( \Pi_i \). \( AP_i \) reflects the properties concerning agent \( i \) in \( T_i \). Note that \( AP_{\varphi_i} \) is the set of APs appearing in \( \varphi_i \).

A. Dependency and Mutual Feasibility

Suppose that one agent receives a cooperative task that involves other agents’ participation. In other words, one agent’s task specification contains APs of another agent.

Definition 3 (Dependency): Agents \( i \) and \( j \) are called dependent when one of the following conditions holds:

1. agent \( i \) depends on agent \( j \) if \( AP_{\varphi_i} \land AP_{\varphi_j} \neq \emptyset \),
2. agent \( j \) depends on agent \( i \) if \( AP_{\varphi_j} \land AP_{\varphi_i} \neq \emptyset \).

The above conditions can be checked by comparing the elements within \( AP_{\varphi_i} \) and \( AP_{\varphi_j} \) (also \( AP_{\varphi_i} \) and \( AP_{\varphi_j} \)) [12]. Based on the dependency relation, we may define the dependency graph of the multi-agent system associated with task specifications \( \varphi_i, i = 1, 2, \ldots, N \).

Definition 4 (Dependency Graph): The dependency graph \( G_d = (V, E) \) consists of: the set of vertices \( V = 1, 2, \ldots, N \) representing the agents; the set of edges \( E \subseteq V \times V \) where \( (i, j) \in E \) if and only if \( i \) and \( j \) are dependent by Definition 3, \( \forall i \neq j \) and \( i, j \in V \).

Definition 5 (Dependency Cluster): \( \Theta \subseteq V \) forms a dependency cluster if and only if \( \forall i, j \in \Theta \) there is a path from \( i \) to \( j \) in the dependency graph \( G_d \).

A closure contains at least one agent, which happens when this single agent is not dependent on any of the other agents.

Loosely speaking, two agents belong to the same cluster when they are directly dependent or transitively dependent by a dependency chain. An example of a dependency graph and dependency clusters are shown in Figure 5. Without loss of generality, we first solve the reconfiguration problem within one cluster \( \Theta = \{1, 2, \ldots, M\} \). Each agent’s transition system and specification automaton are given in (8) and (9).

Given the individual FTS \( T_i, \forall i \in \Theta \), the composed FTS for this cluster \( \Theta \) is constructed by

\[
T_\Theta = (\Pi_{\Theta}, \rightarrow_{\Theta}, \Pi_{\Theta} \cup AP_{\Theta}, L_{\Theta}, W_{\Theta}),
\]

where \( \Pi_{\Theta} = \Pi_1 \cup \Pi_2 \ldots \cup \Pi_M \), \( \varphi_{\Theta} \) and \( AP_{\Theta} \) is the set of all APs appearing in \( \varphi_{\Theta} \).

Fig. 4. Left: the total cost of the optimal accepting run when \( \gamma = 5 \) under different \( \alpha \) (note the same accepting run under different \( \alpha \)). Right: the unique optimal runs, located by their \( \text{cost}_r \) and \( \text{dist}_r \).

Fig. 5. Left: the dependency graph of a system with 9 agents and 4 clusters. Right: different relative distances between \( L(\pi) \) and \( \chi_1, \chi_2 \).

Denote by \( AP_{\varphi_{\Theta}} = AP_{\varphi_1} \cup AP_{\varphi_2} \ldots \cup AP_{\varphi_M} \) the set of all APs appearing in the mutual specification \( \varphi_{\Theta} \). Note that \( AP_{\varphi_{\Theta}} \subseteq AP_{\Theta} \). Since \( \varphi_{\Theta} \) is infeasible over \( T_\Theta \), we need to relax the requirement that every \( \varphi_i \) has to be fulfilled simultaneously. Thus we define the relaxed intersection of the individual specification automaton \( A_{\varphi_i} \).

Definition 6 (Relaxed Automata Intersection): Given \( M \) Büchi automata \( A_{\varphi_1}, A_{\varphi_2}, \ldots, A_{\varphi_M} \) by (9), their relaxed intersection is given by \( A_{\varphi_{\Theta}} = (Q, 2^{AP_{\varphi_{\Theta}}}, \delta, Q_0, F) \), where \( Q = Q_1 \times \cdots \times Q_M \times \{1, 2, \ldots, M\} \); \( Q_0 = Q_{1,0} \times Q_{2,0} \ldots \times Q_{M,0} \times \{1\} \); \( F = F_1 \times F_2 \ldots \times F_M \times \{1\} \); \( \delta \subseteq Q \times Q \times \{\tau_i, t_j\} \times \{q_i, q_j, t'\} \in \delta \) when

- \( \langle q_1, q_2, \ldots, q_M, t \rangle, \langle q_1', q_2', \ldots, q_M', t' \rangle \in Q \).
- \( \exists l_i \in 2^{AP_{\varphi_i}} \) such that \( (q_i, l_i, q_i') \in \delta_i, \forall i \in \Theta \).
- \( q_i \notin F_i \) and \( t' = t, \) or \( q_i \in F_i \) and \( t' = \text{mod} (t, M) + 1 \), where \( \text{mod} \) is the modulo operation.

The conventional definition of Büchi automaton intersection [9] is obtained by replacing the second constraint \( "\exists l_i \in 2^{AP_{\varphi_i}} \) such that \( (q_i, l_i, q_i') \in \delta_i, \forall i \in \Theta" \) by "\( \exists l \in 2^{AP_{\varphi_{\Theta}}} \) such that \( (q_i, l, q_i') \in \delta_i, \forall i \in \Theta\). Namely,
we relax the requirement that there should exist a common input alphabet that enable the transitions from $q_i$ to $q'_i$ in $A_{\varphi_i}, \forall i \in \Theta$. The last component $t \in \{1, 2, \ldots, M\}$ in the state ensures that at least one accepting state of every $A_{\varphi_i}$ is visited infinitely often.

Definition 7 (Relaxed Product Automaton): The relaxed product automaton $A_i = T_{\Theta} \times A_{\varphi_i} = (Q', \delta', Q'_0, F', W_r)$ is defined as follows:

- $Q' = \Pi_\Theta \times Q$, $q' = (\pi_\Theta, q), \forall \pi_\Theta \in \Pi_\Theta$ and $\forall q \in Q$.
- $\delta' \subseteq Q' \times Q'$. $(\pi_\Theta, q_0), (\pi'_\Theta, q_0) \in \delta'$ iff $(\pi_\Theta, \pi'_\Theta) \longrightarrow \pi_\Theta$ and $(q_0, q_0) \in \delta$.
- $Q'_0 = \Pi_{\Theta, 0} \times Q_0$ is the set of initial states.
- $F' = Q \times F$ is the set of accepting states.
- $W_r : \delta' \rightarrow \mathbb{R}^+$ is the weight function, defined as

$$W_r((\pi_\Theta, q_1, \ldots, q_M, t), (\pi'_\Theta, q'_1, \ldots, q'_M, t')) = W_\Theta(\pi_\Theta, \pi'_\Theta) + \sum_{i=1}^M \beta_i \text{Dist}(L_\Theta(\pi_\Theta), \chi_i(q_i, q'_i))$$

where $\alpha, \beta_1, \beta_2, \ldots, \beta_M \geq 0$ are design parameters; the function Dist is defined in (4); $((\pi_\Theta, q_1, \ldots, q_M, t), (\pi'_\Theta, q'_1, \ldots, q'_M, t')) \in \delta'$; $\chi_i(q_i, q'_i) = \{ l \in 2^{A_{\varphi_i}} \mid (q_i, l, q'_i) \in \delta_i \}$ consists of all input alphabets that enable the transition from $q_i$ to $q'_i$ in $A_{\varphi_i}, \forall i \in \Theta$.

Denote by $\beta = \{ \beta_i, i \in \Theta \}$. As “$\exists l_i \in 2^{A_{\varphi_i}}$ such that $(q_i, l_i, q'_i) \in \delta_i, \forall i \in \Theta$” by Definition 6, $\chi_i(q_i, q'_i) \neq \emptyset$. Figure 5 illustrates the relative distances between $L_\Theta(\pi_\Theta)$ and two sets of input alphabets $\chi_1, \chi_2$. The definition of $W_r$ can be interpreted similarly as the one in (5). However, $\beta$ plays the role as the ‘priority’ index for each agent, i.e., the larger $\beta_i$ is, the higher the priority agent $i$ has. For example, if agent $i$ has the highest priority with important tasks, $\beta_i$ can be set very large such that the penalty of violating $A_{\varphi_i}$ is severe. On the other hand, if it plays the role as an assisting robot, $\beta_i$ can be chosen close to zero. Now we would like to state the problem for multi-agent systems:

Problem 2: Given that the mutual specification $\varphi_\Theta$ is infeasible over the composed FTS $T_\Theta$, find the accepting run of $A_i$ that minimizes the cost by (7) and the corresponding individual motion plan for each agent $i$.

Given the value of $\alpha$ and $\beta, A_r$ results in a weighted graph, with the sets of initial and accepting states. Algorithm 1 can be directly applied to find the optimal accepting run, with the prefix-suffix structure (6) and the total cost (7).

Remark 3: It is possible to split $W_\Theta(\pi_\Theta, \pi'_\Theta)$ in $W_r$ into $M$ parts, i.e., the implementation cost of each agent. Relative weighting among these costs can also be added in case of different energy capacities among the agents.

C. Individual Motion Plan and Feedback

Agents within one cluster should agree on the value of $\alpha$ according to the intended relative weighting between the implementation cost and the distance to the mutual tasks, and also the value of $\beta$ based on their priorities within the cluster. Thus in the absence of a central authority, $\alpha$ and $\beta$ can either be determined by the designer prior to the deployment or a consensus algorithm on the value of $\alpha$ and $\beta$ within the cluster might be needed. Then Algorithm 1 is called to generate the optimal accepting run $R_{opt}$. The cooperative motion plan $\tau_\Theta$ is the projection of $R_{opt}$ onto $T_\Theta$. Then Algorithm 3 is used to interpret $R_{opt}$ of $A_i$ for each agent $i$: (i) its individual motion plan $\tau_i$; (ii) the associated revised specification automaton $A'_{\varphi_i}$; (iii) the implementation cost of $\tau_i$; (iv) the accumulated distance of $\tau_\Theta$ to its original task specification $\varphi_1$ dist$\varphi_1$. Note that $\tau_i$ is the projection of $\tau_\Theta$ onto $T_i, A'_{\varphi_i}$ is obtained by adding new transitions to $A_{\varphi_i}$. cost$\tau_i$, and dist$\varphi_i$ are defined similarly as in (7). As an extension, Algorithm 1 could be applied under different $\alpha$ and $\beta$ to derive several optimal accepting run candidates, of which the unique ones are saved. Then Algorithm 3 gives feedback about their implementation cost and their distances to individual specifications.

Lemma 3: Assume $\tau_\Theta$ and dist$\varphi_i$ are the derived from Algorithm 3. Then dist$\varphi_i = 0$ implies that $\tau_i$ satisfies $\varphi_i$.

Proof: The proof is omitted as it is similar to that of Lemma 2.

An example of a two-agent system is shown in Figures 6 and 7. Agent 1 needs to visit $\pi_1$ and $\pi_2$ infinitely often, but never be at $\pi_1$ with agent 2 at the same time. Agent 2 needs to visit $\pi_1$ and stay there. Six different motion plans are obtained by Algorithm 3 under different $\alpha$ and $\beta$, as in Figure 8. The same color indicates that the same optimal accepting run is found. Here we list two motion plan candidates: (i) agent 1: $\pi_0 \pi_1 \pi_2 \pi_1 \pi_2$, agent 2: $\pi_0 \pi_0 \pi_1 \pi_2 \pi_1 \pi_2$ (which has dist$\varphi_1 2$, dist$\varphi_2 0$, cost$\tau_1 12$, cost$\tau_2 8$); (ii) agent 1: $\pi_0 \pi_1 \pi_2 \pi_1 \pi_2 \pi_1 \pi_2$, agent 2: $\pi_0 \pi_0 \pi_1 \pi_0 \pi_1 \pi_2 \pi_1 \pi_2 \pi_1 \pi_2 \pi_1 \pi_2$ (which has dist$\varphi_1 0$, dist$\varphi_2 1$, cost$\tau_1 20$, cost$\tau_2 16$).

The above approach can be applied to any other clusters within the multi-agent system. In particular, the following procedures are carried out: (i) all agents need to confirm their dependency relation, i.e., which cluster they belong to; (ii) within each cluster an agreement on the value of $\alpha$ and $\beta$ should be achieved; (iii) every agent calls Algorithm 1 to derive the optimal accepting run. If there are more than one with the equal total cost by (7), another consensus needs
to be reached regarding which optimal accepting run to choose; (iv) each agent computes the individual motion plan by Algorithm 3; (v) all agents within one cluster implements their motion plans in a synchronized way [10].

**Remark 4:** This multi-agent framework can be modified and applied to the single-agent case where the specification has the “conjunction” form \( \varphi = \varphi_1 \land \varphi_2 \land \cdots \land \varphi_N \). Then the sub-specification \( \varphi_i \) can be modeled as the individual specification of an “imaginary” agent which has identical movements as the “real” agent. \( \beta \) could represent different priorities among these sub-tasks.

V. CORRECTNESS AND COMPLEXITY

The correctness of the proposed solutions in Section III-C and IV-C follows from the problem formulation and the correctness of the Dijkstra’s shortest path algorithm. Let \( |T_i| \) and \( |A_{\varphi_1}| \) denote the size of agent \( i \)'s FTS and the NBA. The size of \( A_r \) by Definition 7 for one cluster with \( M \) members is \( |A_r| = M \prod_{i=1}^{M} |T_i| \cdot |A_{\varphi_i}| \). Algorithm 1 runs in \( O(|A_r| \log |A_r| \cdot |Q_0| \cdot |F^T|) \). Algorithms 2 and 3 have the complexity linear to the length of \( R_{opt} \).

VI. SIMULATION — ASSEMBLY ROBOTS

Consider a team of four unicyclic robots that satisfy: \( \dot{x}_i = v_i \cos \theta_i, \dot{y}_i = v_i \sin \theta_i, \dot{\theta}_i = \omega_i \), where \( p_i = (x_i, y_i)^T \in \mathbb{R}^2 \) is the center of mass for agent \( i \); \( \theta_i \in [0, 2\pi] \) is the orientation; and \( v_i, \omega_i \in \mathbb{R} \) are the transition and rotation velocities, \( i = 1, 2, 3, 4 \). The whole workspace is shown in Figure 9, which consists of 26 polygonal regions. The continuous controller that drives the robots from an region to any geometrically adjacent region is based on [27] by constructing vector fields over each cell for each face. The controller design is not stated here for brevity. All simulations are carried out in MATLAB on a desktop computer (3.06 GHz Duo CPU and 8GB of RAM).

A. Local Specifications

Robots 2, 3 and 4 are confined in rooms 2, 3 and 4 as shown in Figure 9. Each room has six regions, some of which are obstacle-occupied (in grey). They repetitively carry different goods from the storage region to the unloading region within each room, while avoiding obstacles. After picking up goods at the storage region, they have to drop the goods at unloading region before they return to the storage region. The storage, unloading and obstacle-occupied regions are labeled by \( a_{i,s}, a_{i,u} \) and \( a_{i,o} \) respectively for agent \( i = 2, 3, 4 \). Robot 1 has to collect these goods at the regions labeled by \( a_{1,c1}, a_{1,c2} \) and \( a_{1,c3} \) repetitively. In addition, robot 4 needs to meet robot 1 at region labeled by \( a_{4,u'} \). The obstacle-occupied regions for agent 1 are labeled by \( a_{1,o} \). These tasks are specified as LTL formulas by

- **robot 1:** \( \varphi_1 = \Box (a_{1,c1}) \land \Box (a_{1,c2}) \land \Box (a_{1,c3}) \land \Box (a_{4,u'}) \land \Box (a_{4,o}) \)
- **robot 2:** \( \varphi_2 = \Box (a_{i,s} \land \Box (a_{i,u} \land \Box (a_{i,s} \Rightarrow \Box (\neg a_{i,s} \cup a_{i,u})))) \land \Box (\neg a_{i,o}), i = 2, 3, 4 \).

**Dependency and Potential Infeasibility:** By Definition 3, robots 1 and 4 are dependent while robots 2 and 3 run independently. There is a misunderstanding between robots 1 and 4 about the location of robot 4’s unloading region, namely, \( a_{4,u'} \) and \( a_{4,u} \) indicate two different regions, as shown in Room 4 of Figure 9. But this does not necessarily mean that \( \varphi_1 \) and \( \varphi_2 \) are mutually infeasible. Moreover, \( \varphi_3 \) is infeasible for agent 3 because of the obstacles in room 3.

We omit here the detailed diagrams of each robot’s FTS and its associated specification automaton, due to limited space. Each robot can transit between any two geometrically adjacent regions within their confined workspace, of which the costs are uniformly 5. They could also stay at any region independently. There is a misunderstanding between robots 1 and 4 about the location of robot 4’s unloading region, namely, \( a_{4,u'} \) and \( a_{4,u} \) indicate two different regions, as shown in Room 4 of Figure 9. But this does not necessarily mean that \( \varphi_1 \) and \( \varphi_2 \) are mutually infeasible. Moreover, \( \varphi_3 \) is infeasible for agent 3 because of the obstacles in room 3.

B. Simulation Results

Algorithm 3 is applied to the cluster formed by robots 1 and 4. The composed FTS \( T_g \) has 78 states. The relaxed product automaton \( A_r \) consists of 3120 states and 1364 edges, which has three weighting parameters \( \alpha, \beta_1 \) and \( \beta_2 \). By choosing \( \alpha = 0, 20, 100; \beta_0 = 1; \beta_2 = 0, 2, 5, 10, 10\), six unique motion plan candidates are found. Here we choose three of them: (P1) \( \alpha = 100, \beta_1 = 1, \beta_2 = 1 \). Robot 4 travels more distance from its unloading region to meet robot 1 at the collecting region \( \text{dist}_{r1} 0, \text{dist}_{r4} 0, \text{cost}_{r1} 140, \text{cost}_{r4} 48 \); (P2) \( \alpha = 100, \beta_1 = 1, \beta_2 = 0 \). Robots 1 and 4 meet at robot 1’s collecting region \( \text{dist}_{r1} 0, \text{dist}_{r4} 8, \text{cost}_{r1} 140, \text{cost}_{r4} 21 \); (P3) \( \alpha = 30, \beta_1 = 1, \beta_2 = 1 \). Robots 1 and 4 do not meet \( \text{dist}_{r1} 2, \text{dist}_{r4} 0, \text{cost}_{r1} 126, \text{cost}_{r4} 20 \). On the other hand, Algorithm 2
is applied for robot 3 to find the motion plan that violates \( \varphi_3 \) the least. We choose the motion plan under \( \alpha = 2 \), of which the implementation cost is 30 and the distance to \( \varphi_3 \) is 3. In particular, Figures 9 and 10 present the final motion of the composed system when the above motion plans are implemented by the lower-level hybrid controllers.

VII. CONCLUSION AND DISCUSSION

In this paper, we propose a reconfiguration method for the motion planning of multi-agent systems under infeasible local LTL specifications. Algorithms are provided to derive optimal motion plan candidates that are sorted by their implementation costs and their distances to individual task specifications. Future work could include the consideration of limited communications.

REFERENCES


