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 φ 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]: $\varphi ::= \texttt{true} \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc\varphi \mid \varphi_1 \cup \varphi_2$, where $a \in AP$ and $\bigcirc$ (*next*), $\cup$ (*until*). For brevity, we omit the derivations

The authors are with the ACCESS Linnaeus Center, School of Electrical Engineering, KTH Royal Institute of Technology, SE-100 44, Stockholm, Sweden. mengg, dimos@kth.se. This work was supported by the Swedish Research Council (VR), NoE HYCON2 and EU STREP RECONFIG. The second author is also affiliated with the KTH Centre for Autonomous Systems and is supported by VR through contract 2009-3948.

of other useful operators like $\Box$ (*always*), $\Diamond$ (*eventually*), $\Rightarrow$ (*implication*) and refer the readers to Chapter 5 of [3].

Given an LTL formula φ over AP, there is a union of infinite words that satisfy φ: $\texttt{Words}(\varphi) = \{\sigma \in (2^{AP})^\omega \,|\, \sigma \models \varphi\}$, where $\models \subseteq (2^{AP})^\omega \times \varphi$ is the satisfaction relation. There exists a Nondeterministic Büchi automaton (NBA) $\mathcal{A}_\varphi$ over 2^{AP} corresponding to φ, which is defined as:

$$\mathcal{A}_\varphi = (Q,\, 2^{AP},\, \delta,\, Q_0,\, \mathcal{F}), \tag{1}$$

where Q is a finite set of states; Q_0 is the initial state, 2^{AP} is an alphabets; $\delta \subseteq Q \times 2^{AP} \times Q$ is a transition relation and $\mathcal{F} \subseteq Q$ is a set of accepting states. Denote by $\chi(q_m,\, q_n) = \{l \in 2^{AP} |\, (q_m,\, l,\, q_n) \in \delta\}$ the set of all input alphabets that enable the transition from q_m to q_n. An infinite run r of a NBA is an infinite sequence of states and is called accepting if $\texttt{Inf}(r) \cap \mathcal{F} \neq \emptyset$ where $\texttt{Inf}(r)$ is the set of states that appear in r infinitely often. Denote by $\mathcal{L}_\omega(\mathcal{A}_\varphi)$ the accepted language of $\mathcal{A}_\varphi$, which is the set of infinite words that have an accepting run in $\mathcal{A}_\varphi$, i.e., $\texttt{Words}(\varphi) = \mathcal{L}_w(\mathcal{A}_\varphi)$. There are fast translation algorithms [28] from an LTL formula to NBA. This process can be done in time and space $2^{\mathcal{O}(|\varphi|)}$ [3].

B. Discretized Abstraction

A labeled finite transition system [3] is used to describe the behavior of a robot within a workspace. The workspace we consider is geometrically partitioned into N regions, denoted by the set $\Pi = \{\pi_0,\, \pi_1,\, \ldots,\, \pi_N\}$. 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 $\mathcal{T} = (\Pi,\, \longrightarrow_c,\, \Pi_0,\, AP,\, L,\, W_c)$, where $\Pi = \{\texttt{the robot is in region } \pi_i,\, i = 1, 2 \cdots, N\}$; $\longrightarrow_c \subseteq \Pi \times \Pi$ is the transition relation; $\Pi_0 \subseteq \Pi$ is the set of initial states; AP is the set of APs; $L : \Pi \to 2^{AP}$ is a labeling function, giving the subset of AP which are true at state π_i; $W_c : \longrightarrow_c \to \mathbb{R}^+$ 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 $(\pi_i,\, \pi_{i-1}) \in \longrightarrow_c$ for all $i > 0$. Its trace is the sequence of APs that are true at the states along the path, i.e., $\texttt{trace}(\tau) = L(\pi_0)L(\pi_1)L(\pi_2)\cdots$. Given φ is an LTL formula over the same AP the satisfaction relation $\tau \models \varphi$ if and only if $\texttt{trace}(\tau) \in \texttt{Words}(\varphi)$. The infinite path τ that satisfies φ is called a motion plan for the task φ.

C. Motion Plan Synthesis

A valid motion plan τ 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 $\mathcal{A}_p = \mathcal{T} \otimes \mathcal{A}_\varphi = (Q_p,\, \delta_p,\, Q_{p,0},\, \mathcal{F}_p)$, where $Q_p = \Pi \times Q$; $Q_{p,0} = \Pi_0 \times Q_0$ are the initial states; $\mathcal{F}_p = \Pi \times \mathcal{F}$ are the accepting states; $\delta_p \subseteq Q \times Q$ is the transition relation.

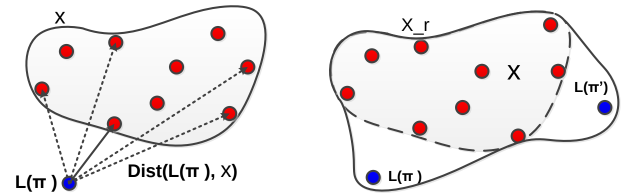

Fig. 1. Left: the distance of a labeling function $L(\pi)$ to a set of input alphabets χ (the solid line). Right: the set of the input alphabets is revised by adding some labeling functions to the set.

$(\langle\pi_i,\, q_m\rangle,\, \langle\pi_j,\, q_n\rangle) \in \delta_p$ if and only if $(\pi_i,\, \pi_j) \in \longrightarrow_c$ and $(q_m,\, L(\pi_i),\, q_n) \in\, \delta$. There exists an motion plan satisfying φ if and only if $\mathcal{A}_p$ has at least one accepting run [3].

Lemma 1 (Feasibility and Projection): An LTL specification φ is feasible over the FTS $\mathcal{T}$ if and only if $\mathcal{A}_p = \mathcal{T} \otimes \mathcal{A}_\varphi$ has an accepting run. Furthermore, for any accepting run $R = \langle\pi_0, q_0\rangle\langle\pi_1, q_1\rangle\ldots$ of $\mathcal{A}_p$, its projection onto $\mathcal{T}$ the sequence $\tau = \pi_0\pi_1\ldots$ satisfies φ [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 $\mathcal{A}'_\varphi$ which is close to $\mathcal{A}_\varphi$ (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 $\mathcal{T} \otimes \mathcal{A}'_\varphi$ and they may fulfill the original φ to different extents. We instead aim to find the motion plan that fulfills φ the most with respect to certain criterion, based on which then the relaxed specification is constructed.

A. Relaxed Product Automaton

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

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

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

Two differences between $\mathcal{A}_r$ and $\mathcal{A}_p$ defined in Section II-C are: (i) the constraint "$(q_m,\, L(\pi_i),\, q_n) \in \delta$" when defining

δ_p is relaxed to "$\exists\, l \in 2^{AP}$ such that $(q_m,\, l,\, q_n) \in \delta$" when defining δ' here; (ii) the weight function W_r is only introduced for $\mathcal{A}_r$. Firstly we introduce the evaluation function $\texttt{Eval} : 2^{AP} \to \{0,\, 1\}^K$:

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

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

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

where $\nu = \texttt{Eval}(l)$, $\nu' = \texttt{Eval}(l')$ and $l,\, l' \in 2^{AP}$. $\|\cdot\|_1$ is the ℓ_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]:

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

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

$$\begin{aligned} &W_r((\langle \pi_i,\, q_m\rangle,\, \langle \pi_j,\, q_n\rangle)) \\ &= W_c(\pi_i,\, \pi_j) + \alpha\, \cdot \texttt{Dist}(L(\pi_i),\, \chi(q_m,\, q_n)), \end{aligned} \tag{5}$$

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 $\mathcal{A}_\varphi$. 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 π_i to π_j in $\mathcal{T}$. $\texttt{Dist}(L(\pi_i),\, \chi(q_m,\, q_n))$ measures how much the transition from π_i to π_j violates the constraints imposed by the transition from q_m to q_n. Being 0 means that $\mathcal{A}_\varphi$ is not violated, while the larger the distance is the more $\mathcal{A}_\varphi$ is violated. The design parameter α 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 $\mathcal{A}_\varphi$ is increased when α is larger.

B. Problem Statement

Note that $\mathcal{A}_r$ is more connected than the conventional product automaton $\mathcal{A}_p$ in Section II-C. Since $\mathcal{A}_p$ does not have an accepting run, we instead search for an accepting run within $\mathcal{A}_r$. 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:

$$\begin{aligned} R &= q'_0\, q'_1 \cdots [\, q'_k\, q'_{k+1} \cdots\cdots q'_n\,]^\omega \\ &= \langle \pi_0, q_0\rangle\, \langle \pi_1, q_1\rangle \cdots [\,\langle \pi_k, q_k\rangle \cdots\cdots \langle \pi_n, q_n\rangle\,]^\omega\ , \end{aligned} \tag{6}$$

where $q'_0 = \langle \pi_0, q_0\rangle \in Q'_0$ and $q'_k = \langle \pi_k, q_k\rangle \in \mathcal{F}'$. 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 $(q'_k\, q'_{k+1} \cdots\cdots q'_n)$ 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]):

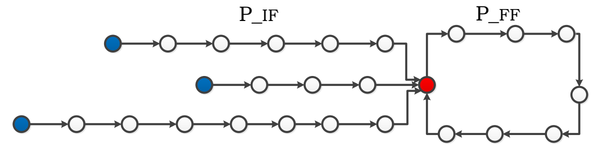

Fig. 2. For every accepting state $q'_f \in \mathcal{F}'$ (in red), P_{IF} contains the paths from every initial state $q'_0 \in Q'_0$ (in blue) to p_f with the minimal costs; P_{FF} contains the path from q'_f back to itself with the minimal cost.

$$\begin{aligned} \texttt{Cost}(R) &= \sum_{i=0}^{k-1} W_r(q'_i,\, q'_{i+1}) + \gamma \sum_{i=k}^{n-1} W_r(q'_i,\, q'_{i+1}) \\ &= \texttt{cost}_\tau + \alpha \cdot \texttt{dist}_\varphi\, , \end{aligned} \tag{7}$$

where $\texttt{cost}_\tau = (\sum_{i=0}^{k-1} + \gamma \sum_{i=k}^{n-1})\, W_c(\pi_i,\, \pi_{i+1})$ is the accumulated implementation cost of the motion plan τ, i.e., the projection of R onto $\mathcal{T}$; $\texttt{dist}_\varphi = (\sum_{i=0}^{k-1} + \gamma \sum_{i=k}^{n-1})\, \texttt{Dist}(L(\pi_i), \chi(q_i, q_{i+1}))$ is the accumulated distance of τ to $\mathcal{A}_\varphi$. 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 in it 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 the an infeasible specification φ over the FTS $\mathcal{T}$, find the accepting run of $\mathcal{A}_r$ that minimizes the cost by (7) and the corresponding motion plan τ.

Given $\mathcal{A}_r$ and a value of α, we call the solution to Problem 1 as the *optimal* accepting run R_{opt}under that α. Algorithm 1 takes as input arguments the weighted state graph [3] $G(\mathcal{A}_r) = (Q',\, \delta',\, W_r)$, the set of initial vertices $I = Q'_0$ and the set of accepting vertices $F = \mathcal{F}'$. 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 $\texttt{MinPath}$ takes $(G,\, I,\, F)$ as inputs and outputs a $L \times M$ matrix D_{IF}, with the $(i_{\text{th}},\, j_{\text{th}})$ element containing the value of the minimal cost from I_i to F_j; and a $L \times M$ cell P_{IF}, with the $(i_{\text{th}},\, j_{\text{th}})$ cell containing the sequence of vertices appearing in the path with minimal cost from I_i to F_j. Function $\texttt{MinCycl}$ is a variant of function $\texttt{MinPath}$,

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.

$$(D_{IF}, P_{IF}) = \texttt{MinPath}(G,\, I,\, F).$$

2. Compute the path with minimal cost from every accepting vertex in F and back to itself:

$$(D_{FF}, P_{FF}) = \texttt{MinCycl}(G,\, F).$$

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 f_{min}.
5. Optimal accepting run R_{opt}, prefix: the f_{min}-th element of P_{iF}; suffix: the f_{min}-th element 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 α is chosen in $\mathcal{A}_r$. Then Algorithm 2 takes as inputs R_{opt}, the FTS $\mathcal{T}$ and the original specification automaton $\mathcal{A}_\varphi$. While iterating through the transitions along R_{opt} in sequence, it projects R_{opt} into $\mathcal{T}$ to obtain the corresponding motion plan τ; it constructs the revised specification automaton $\mathcal{A}'_\varphi$ by adding new transitions to $\mathcal{A}_\varphi$ (as shown in Figure 1); it computes the implementation cost $\texttt{cost}_\tau$ and the accumulated distance to $\mathcal{A}_\varphi$ $\texttt{dist}_\varphi$ defined in (7). It can be verified that the obtained $\mathcal{A}'_\varphi$ is a valid relaxation of $\mathcal{A}_\varphi$ [16]. Note each accepting run R_{opt} corresponds to a unique motion plan τ and a revised specification automaton $\mathcal{A}'_\varphi$.

Remark 1: Although $\mathcal{A}_r$ may allow more transitions compared with $\mathcal{A}_p$, any run of $\mathcal{A}_r$ 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 $\mathcal{A}_r$. Thus the motion plan derived from Algorithm 2 is always implementable.

Lemma 2: Assume τ and $\texttt{dist}_\varphi$ are the derived from Algorithm 2. Then $\texttt{dist}_\varphi = 0$ implies that τ satisfies φ.

Proof: Since $\texttt{Dist}() \geq 0$, the accumulated distance $\texttt{dist}_\varphi = 0$ 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 $\mathcal{A}_p$. Its projection τ satisfies φ by Lemma 1. ■

Algorithms 1 and 2 solve Problem 1 under a given α. However it may not be trivial to determine the appropriate value of α. As an extension, Algorithm 1 could be called under different α 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 τ and the associated $\texttt{cost}_\tau$, $\texttt{dist}_\varphi$ as the feedback.

Algorithm 2: Function `MP-SA-single`(R_{opt}, $\mathcal{T}$, $\mathcal{A}_\varphi$)

Input: an optimal accepting run R_{opt}, $\mathcal{T}$, $\mathcal{A}_\varphi$.
Output: the corresponding motion plan τ, the revised $\mathcal{A}'_\varphi$, $\texttt{cost}_\tau$ and $\texttt{dist}_\varphi$.
1. Initialization: $\mathcal{A}'_\varphi = \mathcal{A}_\varphi$. $\texttt{cost}_\tau = \texttt{dist}_\varphi = 0$.
2. Follow the transitions along R_{opt} by (6), namely $(q'_i,\, q'_{i+1})$, $i = 1, \cdots, n-1$, perform Steps 3-5:
3. Let $q'_i = \langle\pi,\, q_m\rangle$ and $q'_{i+1} = \langle\pi',\, q_n\rangle$.
4. Save $(\pi,\, \pi')$ in τ. $\texttt{cost}_\tau = \texttt{cost}_\tau + W_c(\pi,\, \pi')$.
5. Check if $(q_m,\, L(\pi),\, q_n) \in \delta$ holds. If so, $\mathcal{A}'_\varphi$ remains unchanged. Otherwise, add $(q_m,\, L(\pi),\, q_n)$ to δ of $\mathcal{A}'_\varphi$. $\texttt{dist}_\varphi = \texttt{dist}_\varphi + \texttt{Dist}(L(\pi),\, \chi(q_m,\, q_n))$.

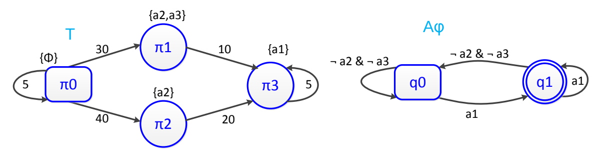

Fig. 3. The FTS $\mathcal{T}$ has four states, labeled by the labeling functions. Edges are labeled by the costs. The task specification is $\varphi = \square\Diamond a_1 \wedge \square\neg(a_2 \wedge a_3)$, the corresponding $\mathcal{A}_\varphi$ is shown by the diagram on the right.

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

An example system is shown in Figure 3. The agent has to go from region π_0 to π_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 α, as shown in Figure 4: (i) when the penalty on violating φ is low, $\mathcal{A}_\varphi$ is revised by adding $(q_0,\, \emptyset,\, q_1)$ and $(q_1,\, \emptyset,\, q_1)$ to δ and the corresponding motion plan is $[\pi_0]^\omega$ (black hexagram, $\texttt{cost}_\tau$ 30, $\texttt{dist}_\varphi$ 6); (ii) when the penalty is increased, $\mathcal{A}_\varphi$ is revised by adding $(q_0,\, \{a_2,\, a_3\},\, q_1)$ to δ, where the motion plan is $\pi_0\, \pi_1\, [\pi_3]^\omega$ (blue square, $\texttt{cost}_\tau$ 65, $\texttt{dist}_\varphi$ 2); (iii) when the penalty is severe, $\mathcal{A}_\varphi$ is revised by adding $(q_0,\, \{a_2\},\, q_1)$ to δ, where the motion plan is $\pi_0\, \pi_2\, [\pi_3]^\omega$ (cyan triangle, $\texttt{cost}_\tau$ 85, $\texttt{dist}_\varphi$ 1). Note that in plan (iii) the agent passes through π_2 which satisfies only a_2, instead of π_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

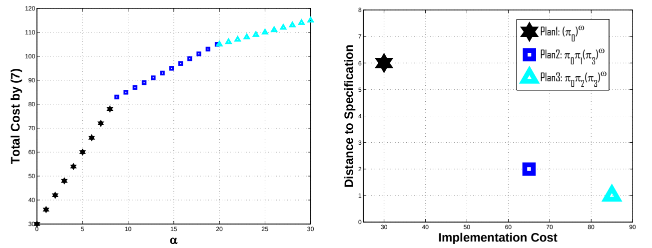

Fig. 4. Left: the total cost of the optimal accepting run when $\gamma = 5$ under different α (note the same accepting run under different α). Right: the unique optimal runs, located by their $\texttt{cost}_{\mathcal{T}}$ and $\texttt{dist}_{\varphi}$.

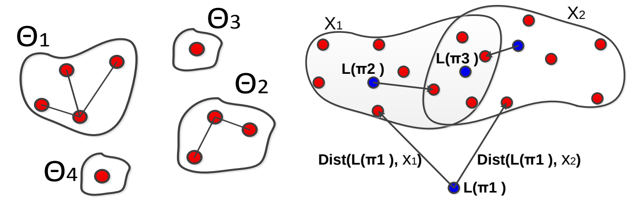

Fig. 5. Left: the dependency graph of a system with 9 agents and 4 clusters. Right: different relative distances between $L(\pi)$ and χ_1, χ_2.

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 $i = 1, 2\cdots, N$. Moreover, we denote the finite transition system of agent i by

$$\mathcal{T}_i = (\Pi_i,\, \longrightarrow_i,\, \Pi_{i,0},\, AP_i,\, L_i,\, W_i); \tag{8}$$

its LTL specification by φ_i; the specification automaton by

$$\mathcal{A}_{\varphi_i} = (Q_i,\, 2^{AP_{\varphi_i}},\, \delta_i,\, Q_{i,0},\, \mathcal{F}_i), \tag{9}$$

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 $\mathcal{T}$ and $\mathcal{A}_\varphi$ introduced in Section II. $\mathcal{T}_i$ abstracts agent i's behavior within its workspace Π_i. AP_i reflects the properties concerning agent i in $\mathcal{T}_i$. Note that AP_{φ_i} is the set of APs appearing in φ_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 $AP_{\varphi_i} \wedge AP_j \neq \emptyset$,
(2) agent j depends on agent i if $AP_{\varphi_j} \wedge AP_i \neq \emptyset$.

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

Definition 4 (Dependency Graph): The dependency graph $G_d = (V, E)$ consists of: the set of vertices $V = 1, 2\cdots, N$ representing the agents; the set of edges $E \subseteq V \times V$ where $(i, j) \in E$ and $(j, i) \in E$ if agent 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 $\Theta = \{1, 2, \cdots, M\}$. Each agent's transition system and specification automaton are given in (8) and (9).

Given the individual FTS $\mathcal{T}_i$, $\forall i \in \Theta$, the composed FTS for this cluster Θ is constructed by

$$\mathcal{T}_\Theta = (\Pi_\Theta,\, \longrightarrow_\Theta,\, \Pi_{\Theta,0},\, AP_\Theta,\, L_\Theta,\, W_\Theta), \tag{10}$$

where $\Pi_\Theta = \Pi_1 \times \Pi_2 \cdots \times \Pi_M$; $\langle \pi_1, \pi_2 \cdots, \pi_M\rangle \longrightarrow_\Theta \langle \pi'_1, \pi'_2 \cdots, \pi'_M\rangle$ if and only if $\pi_i \longrightarrow_i \pi'_i$, $i = 1, 2, \cdots, M$; $\Pi_{\Theta,0} = \Pi_{1,0} \times \Pi_{2,0} \cdots \times \Pi_{M,0}$; $AP_\Theta = AP_1 \cup AP_2 \cdots \cup AP_M$; $L_\Theta(\langle \pi_1, \pi_2 \cdots, \pi_M\rangle) = L_1(\pi_1) \cup L_2(\pi_2) \cdots \cup L_M(\pi_M)$; $W_\Theta(\langle \pi_1, \pi_2 \cdots, \pi_M\rangle, \langle \pi'_1, \pi'_2 \cdots, \pi'_M\rangle) = W_1(\pi_1, \pi'_1) + W_2(\pi_2, \pi'_2) \cdots + W_M(\pi_M, \pi'_M)$.

We denote the mutual specification by $\varphi_\Theta = \varphi_1 \wedge \varphi_2 \cdots \wedge \varphi_M$, i.e., the conjunction of all individual task specifications. $\mathcal{A}_{\varphi_\Theta}$ is the NBA associated with φ_Θ. Then $\{\varphi_i, \forall i \in \Theta\}$ are called *mutually infeasible* if φ_Θ is infeasible over $\mathcal{T}_\Theta$ by Definition 1. Thus the question of how to synthesize the cooperative motion plans that fulfill the mutual specification the most arises.

B. Problem Statement

Denote by $AP_{\varphi_\Theta} = AP_{\varphi_1} \cup AP_{\varphi_2} \cdots \cup AP_{\varphi_M}$ the set of all APs appearing in the mutual specification φ_Θ. Note that $AP_{\varphi_\Theta} \subseteq AP_\Theta$. Since φ_Θ is infeasible over $\mathcal{T}_\Theta$, we need to relax the requirement that every φ_i has to be fulfilled simultaneously. Thus we define the relaxed intersection of the individual specification automaton $\mathcal{A}_{\varphi_i}$.

Definition 6 (Relaxed Automata Intersection): Given M Büchi automata $\mathcal{A}_{\varphi_1}, \mathcal{A}_{\varphi_2} \cdots, \mathcal{A}_{\varphi_M}$ by (9), their relaxed intersection is given by $\tilde{\mathcal{A}}_{\varphi_\Theta} = (Q,\, 2^{AP_{\varphi_\Theta}},\, \delta,\, Q_0,\, \mathcal{F})$, where $Q = Q_1 \times \cdots \times Q_M \times \{1, 2\cdots, M\}$; $Q_0 = Q_{1,0} \times Q_{2,0} \cdots \times Q_{M,0} \times \{1\}$; $\mathcal{F} = \mathcal{F}_1 \times Q_2 \cdots \times Q_M \times \{1\}$; $\delta \subseteq Q \times Q$. $(\langle q_1, \cdots, q_M, t\rangle, \langle q'_1, \cdots, q'_M, t'\rangle) \in \delta$ when

- $\langle q_1, q_2 \cdots, q_M, t\rangle, \langle q'_1, q'_2 \cdots, q'_M, t'\rangle \in Q$.
- $\exists\, l_i \in 2^{AP_{\varphi_\Theta}}$ such that $(q_i, l_i, q'_i) \in \delta_i$, $\forall i \in \Theta$.
- $q_t \notin \mathcal{F}_t$ and $t' = t$, *or* $q_t \in \mathcal{F}_t$ and $t' = \texttt{mod}(t, M) + 1$, where $\texttt{mod}$ is the modulo operation.

The conventional definition of Büchi automaton intersection [9] is obtained by replacing the second constraint "$\exists\, l_i \in 2^{AP_{\varphi_\Theta}}$ 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 $\mathcal{A}_{\varphi_i}$, $\forall i \in \Theta$. The last component $t \in \{1, 2, \cdots, M\}$ in the state ensures that at least one accepting state of every $\mathcal{A}_{\varphi_i}$ is visited infinitely often.

Definition 7 (Relaxed Product Automaton): The relaxed product automaton $\mathcal{A}_r = \mathcal{T}_\Theta \times \tilde{A}_{\varphi_\Theta} = (Q', \delta', Q'_0, \mathcal{F}', W_r)$ is defined as follows:

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

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

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

Denote by $\beta = \{\beta_i, i \in \Theta\}$. As "$\exists l_i \in 2^{AP_{\varphi_\Theta}}$ 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 χ_1, χ_2. The definition of W_r can be interpreted similarly as the one in (5). However, β plays the role as the 'priority' index for each agent, i.e., the larger β_i is, the higher the priority agent i has. For example, if agent i has the highest priority with important tasks, β_i can be set very large such that the penalty of violating $\mathcal{A}_{\varphi_i}$ is severe. On the other hand, if it plays the role as an assisting robot, β_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 φ_Θ is infeasible over the composed FTS $\mathcal{T}_\Theta$, find the accepting run of $\mathcal{A}_r$ that minimizes the cost by (7) and the corresponding individual motion plan for each agent i.

Given the value of α and β, $\mathcal{A}_r$ 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 α according to the intended relative weighting between the implementation cost and the distance to the mutual tasks, and also the value of β based on their priorities within the cluster. Thus in the absence of a central authority, α and β can either be determined by the designer prior to the deployment or a consensus algorithm on the value of α and β within the cluster might be needed. Then Algorithm 1 is called to generate the optimal accepting run R_{opt}. The cooperative motion plan τ_Θ is the projection of R_{opt} onto $\mathcal{T}_\Theta$. Then Algorithm 3 is used to interpret R_{opt} of $\mathcal{A}_r$ for each agent i: (i) its individual motion plan τ_i; (ii) the associated revised specification automaton $\mathcal{A}'_{\varphi_i}$; (iii) the implementation cost of τ_i $\mathtt{cost}_{\tau_i}$; (iv) the accumulated distance of τ_Θ to its original task specification φ_i $\mathtt{dist}_{\varphi_i}$. Note that τ_i is the projection of τ_Θ onto $\mathcal{T}_i$. $\mathcal{A}'_{\varphi_i}$ is obtained by adding new transitions to $\mathcal{A}_{\varphi_i}$. $\mathtt{cost}_{\tau_i}$ and $\mathtt{dist}_{\varphi_i}$ are defined similarly as in (7). As an extension, Algorithm 1 could be applied under different α and β 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.

Algorithm 3: Function $\mathtt{MP\text{-}SA\text{-}multi}(R_{opt},\ \mathcal{T}_i,\ \mathcal{A}_{\varphi_i})$

Input: an optimal accepting run R_{opt} of $\mathcal{A}_r$; $\mathcal{T}_i$ and the original specification automata $\mathcal{A}_{\varphi_i}$.

Output: agent i's τ_i, $\mathtt{cost}_{\tau_i}$, $\mathtt{dist}_{\varphi_i}$ and $\mathcal{A}'_{\varphi_i}$.

1. Initialization: $\mathcal{A}'_{\varphi_i} = \mathcal{A}_{\varphi_i}$, $\mathtt{cost}_{\tau_i} = \mathtt{dist}_{\varphi_i} = 0$.
2. For all transitions along the accepting path R_{opt} (in sequence), namely (q'_j, q'_{j+1}), $j = 1, 2, \cdots, n-1$, perform Steps 3-5 as follows:
3. Let $q'_j = \langle \pi_\Theta, q_1, \cdots, q_M, t \rangle$ and $q'_{j+1} = \langle \pi'_\Theta, q'_1, \cdots, q'_M, t' \rangle$.
4. Project $(\pi_\Theta, \pi'_\Theta)$ onto $\mathcal{T}_i$ and save the projection $(\pi_{i,\Theta}, \pi'_{i,\Theta})$ in τ_i. $\mathtt{cost}_{\tau_i} = \mathtt{cost}_{\tau_i} + W_i(\pi_{i,\Theta}, \pi'_{i,\Theta})$;
5. Check if $(q_i, L_\Theta(\pi_\Theta), q'_i) \in \delta_i$. If so, $\mathcal{A}'_{\varphi_i}$ remains unchanged. Otherwise, add $(q_i, L_\Theta(\pi_\Theta), q'_i)$ to δ_i of $\mathcal{A}'_{\varphi_i}$. $\mathtt{dist}_{\varphi_i} = \mathtt{dist}_{\varphi_i} + \mathtt{Dist}(L_\Theta(\pi_\Theta), \chi_i(q_i, q'_i))$.

Lemma 3: Assume τ_Θ and $\mathtt{dist}_{\varphi_i}$ are the derived from Algorithm 3. Then $\mathtt{dist}_{\varphi_i} = 0$ implies that τ_Θ satisfies φ_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 π_1 and π_2 infinitely often, but never be at π_1 with agent 2 at the same time. Agent 2 needs to visit π_1 and stay there. Six different motion plans are obtained by Algorithm 3 under different α and β, 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: $\pi_0\pi_1[\pi_2]^\omega$, agent 2: $\pi_0\pi_0[\pi_1]^\omega$ (which has $\mathtt{dist}_{\varphi_1}$ 2, $\mathtt{dist}_{\varphi_2}$ 0, $\mathtt{cost}_{\tau_1}$ 12, $\mathtt{cost}_{\tau_2}$ 8); (ii) agent 1: $\pi_0\pi_1[\pi_2\pi_1\pi_2]^\omega$, agent 2: $\pi_0\pi_0[\pi_1\pi_0\pi_1]^\omega$ (which has $\mathtt{dist}_{\varphi_1}$ 0, $\mathtt{dist}_{\varphi_2}$ 1, $\mathtt{cost}_{\tau_1}$ 20, $\mathtt{cost}_{\tau_2}$ 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 α and β 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

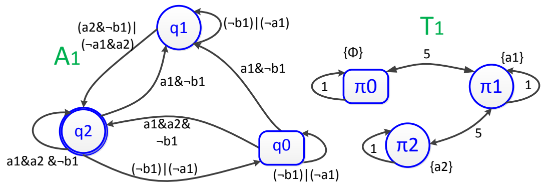

Fig. 6. Agent 1: its FTS $\mathcal{T}_1$ and NBA $\mathcal{A}_{\varphi_1}$ associated with $\varphi_1 = (\Box\Diamond a_1) \wedge (\Box\Diamond a_2) \wedge (\Box\neg(a_1 \wedge b_1))$.

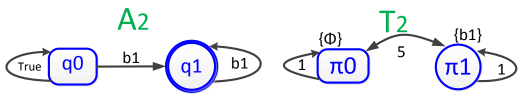

Fig. 7. Agent 2: its FTS $\mathcal{T}_2$ and NBA $\mathcal{A}_{\varphi_2}$ associated with $\varphi_2 = \Diamond\Box\, b_1$.

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 $\varphi = \varphi_1 \wedge \varphi_2 \cdots \wedge \varphi_N$. Then the sub-specification φ_i can be modeled as the individual specification of an "imaginary" agent which has identical movements as the "real" agent. β 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 $|\mathcal{T}_i|$ and $|\mathcal{A}_{\varphi_i}|$ denote the size of agent i's FTS and the NBA. The size of $\mathcal{A}_r$ by Definition 7 for one cluster with M members is $|\mathcal{A}_r| = M \cdot \prod_{i=1}^{M} |\mathcal{T}_i| \cdot |\mathcal{A}_{\varphi_i}|$. Algorithm 1 runs in $\mathcal{O}(|\mathcal{A}_r| \log |\mathcal{A}_r| \cdot |Q'_0| \cdot |\mathcal{F}'|)$. Algorithms 2 and 3 have the complexity linear to the length of R_{opt}.

VI. Simulation — Assembly Robots

Consider a team of four unicycle 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 $\mathbf{p}_i = (x_i,\, y_i)^T \in \mathbb{R}^2$ 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

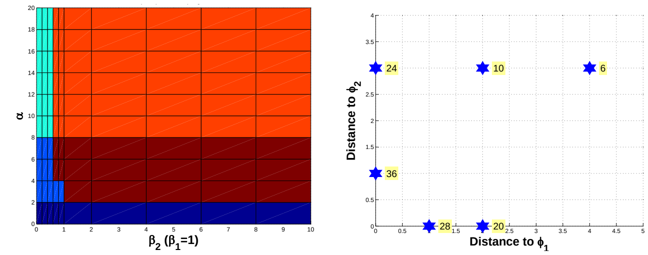

Fig. 8. Left: the optimal accepting runs generated under different α and β ($\gamma = 5$). Right: the alternative motion plans located by their distance to φ_1 and distance to φ_2 (y-axis), and labeled by the total implementation cost.

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: $\varphi_1 = \Box\Diamond(a_{1,c1}) \wedge \Box\Diamond(a_{1,c2}) \wedge \Box\Diamond(a_{1,c3} \wedge a_{4,u'}) \wedge \Box(\neg a_{1,o})$
- robot i: $\varphi_i = \Box\Diamond a_{i,s} \wedge \Box\Diamond a_{i,u} \wedge \Box(a_{i,s} \Rightarrow \bigcirc(\neg a_{i,s} \cup a_{i,u})) \wedge \Box(\neg a_{i,o}),\ \ i = 2, 3, 4.$

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 φ_1 and φ_4 are mutually infeasible. Moreover, φ_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 with the cost 1. $\mathcal{T}_1$ has 13 states while $\mathcal{T}_i$ has 6 states; $\mathcal{A}_{\varphi_1}$ has 4 states and $\mathcal{A}_{\varphi_i}$ has 5 states by [28], $i = 2,\, 3,\, 4$.

B. Simulation Results

Algorithm 3 is applied to the cluster formed by robots 1 and 4. The composed FTS $\mathcal{T}_g$ has 78 states. The relaxed product automaton $\mathcal{A}_r$ consists of 3120 states and 1364 edges, which has three weighting parameters α, β_1 and β_2. By choosing $\alpha = 0,\, 20,\, 100$; $\beta_1 = 1$; $\beta_2 = 0,\, 0.5,\, 1,\, 10$, six unique motion plan candidates are found. Here we choose three of them: (P1) $\alpha = 100$, β_1, $\beta_2 = 1$. Robot 4 travels more distance from its unloading region to meet robot 1 at the collecting region ($\texttt{dist}_{\varphi_1}$ 0, $\texttt{dist}_{\varphi_4}$ 0, $\texttt{cost}_{\tau_1}$ 140, $\texttt{cost}_{\tau_4}$ 48); (P2) $\alpha = 100$, $\beta_1 = 1$, $\beta_2 =$ 0. Robots 1 and 4 meet at robot 1's collecting region ($\texttt{dist}_{\varphi_1}$ 0, $\texttt{dist}_{\varphi_4}$ 8, $\texttt{cost}_{\tau_1}$ 140, $\texttt{cost}_{\tau_4}$ 21); (P3) $\alpha = 30$, $\beta_1, \beta_2 = 1$. Robots 1 and 4 do not meet ($\texttt{dist}_{\varphi_1}$ 2, $\texttt{dist}_{\varphi_4}$ 0, $\texttt{cost}_{\tau_1}$ 126, $\texttt{cost}_{\tau_4}$ 20). On the other hand, Algorithm 2

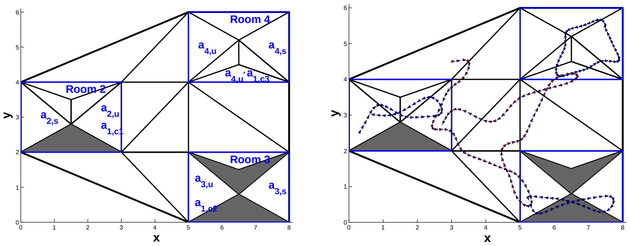

Fig. 9. Left: the workspace model, where blue boxes indicate the confined rooms for robots 2, 3 and 4; Right: both φ_1 and φ_2 are fulfilled (corresponds to P1), and robot 3 chooses the plan that violates φ_3 the least.

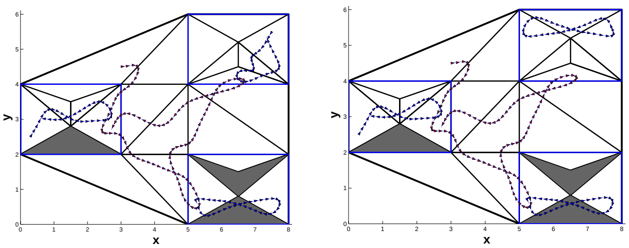

Fig. 10. Left: robots 1 and 4 meet at $a_{1,u'}$ and φ_1 is fulfilled but not φ_4 (corresponds to P2). Right: robots 1 and 4 do not meet while φ_4 is fulfilled but not φ_1 (corresponds to P3).

is applied for robot 3 to find the motion plan that violates φ_3 the least. We choose the motion plan under $\alpha = 2$, of which the implementation cost is 30 and the distance to φ_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

[1] A. Bhatia, L. E. Kavraki, M. Y. Vardi. Sampling-based motion planning with temporal goals. *IEEE International Conference on Robotics and Automation*, 2010.

[2] A. Bhatia, M. R. Maly, L. E. Kavraki, M. Y. Vardi. Motion planning with complex goals. *IEEE Robotics & Automation Magazine*, 18(3): 55-64, 2011.

[3] C. Baier, J.-P Katoen. Principles of model checking. *The MIT Press*, 2008.

[4] K. E. Bekris, K. I. Tsianos, L. E. Kavraki. Safe and Distributed Kinodynamic Replanning for Vehicular Networks. *ACM/Springer Mobile Networks and Applications*, 14(3): 292-308, 2009.

[5] C. Belta, V. Isler, G. J. Pappas. Discrete abstractions for robot motion planning and control in polygonal environments. *IEEE Transactions on Robotics*, 21(5): 864-874, 2005.

[6] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, G. J. Pappas. Symbolic planning and control of robot motion. *IEEE Robotics and Automation Magazine*, 14: 61-71, 2007.

[7] S. Boyd, L. Vandenberghe. *Convex Optimization*, Cambridge University Press, 2009.

[8] Y. Chen, J. Tumova, C. Belta. LTL Robot motion control based on automata learning of environmental dynamics. *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[9] E. M. Clarke, O. Grumberg, D. A. Peled. Model checking. *The MIT Press*, 1999.

[10] X. Ding, M. Kloetzer, Y. Chen, C. Belta. Automatic deployment of robotic teams. *IEEE Robotics Automation Magazine*, 18: 75-86, 2011.

[11] G. E. Fainekos, A. Girard, H. Kress-Gazit, G. J. Pappas. Temporal Logic Motion Planning for Dynamic Mobile Robots. *Automatica*, 45(2): 343-352, 2009.

[12] I. F. Filippidis, D. V. Dimarogonas, K. J. Kyriakopoulos. Decentralized Multi-Agent Control from Local LTL Specifications. *IEEE Conference on Decision and Control*, 2012.

[13] M. Guo, K. H. Johansson, D. V. Dimarogonas. Revising Motion Planning under Linear Temporal Logic Specifications in Partially Known Workspaces. *IEEE International Conference on Robotics and Automation*, 2013. to appear.

[14] S. Karaman, E. Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30(7): 846-894, 2011.

[15] S. Karaman, E. Frazzoli. Vehicle routing with linear temporal logic specifications: Applications to Multi-UAV Mission Planning. *Navigation, and Control Conference in AIAA Guidance*, 2008.

[16] K. Kim, G. E. Fainekos, S. Sankaranarayanan. On the revision problem of specification automaton. *IEEE International Conference on Robotics and Automation*, 5171-5176, 2012.

[17] K. Kim, G. E. Fainekos. Approximate solutions for the minimal revision problem of specification automata *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[18] M. Kloetzer, C. Belta. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Transactions on Automatic Control*, 53(1): 287-297, 2008.

[19] M. Kloetzer, C. Belta. Automatic deployment of distributed teams of robots from temporal logic specifications. *IEEE Transactions on Robotics*, 26(1): 48-61, 2010.

[20] M. Kloetzer, X. C. Ding, C. Belta. Multi-robot deployment from LTL specifications with reduced communication, *IEEE Conference on Decision and Control*, 2011

[21] D. E. Koditschek, E. Rimon. Robot navigation functions on manifolds with boundary. *Advances Appl. Math.*, 11:412-442, 1990.

[22] H. Kress-Gazit, T. Wongpiromsarn, U. Topcu. Correct, reactive robot control from abstraction and temporal logic specifications. *IEEE Robotics and Automation Magazine*, 2011.

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

[24] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli. Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on Control Systems*, 17(5): 1105-1118, 2009.

[25] S. M. LaValle. Planning algorithms. *Cambridge University Press*, 2006.

[26] S. C. Livingston, R. M. Murray, J. W. Burdick. Backtracking temporal logic synthesis for uncertain environments. *IEEE International Conference on Robotics and Automation*, 51635170, 2012.

[27] S. R. Lindemann, I. I. Hussein, S. M. LaValle. Real time feedback control for nonholonomic mobile robots with obstacles. *IEEE Conference on Decision and Control*, 2406-2411, 2006.

[28] D. Oddoux, P. Gastin. LTL2BA software: fast translation from LTL formulae to Büchi automaton. `http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/index.php`.

[29] A. S. Oikonomopoulos, S. G. Loizou, K. J. Kyriakopoulos. Coordination of multiple non-holonomic agents with input constraints. *IEEE International Conference on Robotics and Automation*, 869-874, 2009.

[30] T. Wongpiromsarn, U. Topcu, R. Murray. Receding horizon temporal logic planning, *IEEE Transactions on Automatic Control*, 2012.

[31] V. Raman, H. Kress-Gazit. Analyzing Unsynthesizable Specifications for high-Level robot behavior using LTLMoP. *In the Proceedings of Computer Aided Verification*, 2011.

[32] A. Ulusoy, S. L. Smith, C. Belta. Optimal Multi-robot path planning with LTL constraints: guaranteeing correctness through synchronization. *International Symposium on Distributed Autonomous Robotic Systems*, 2012.

[33] S. L. Smith, J. Tumova, C. Belta, D. Rus. Optimal path planning for surveillance with temporal logic constraints. *International Journal of Robotics Research*, 30(14): 1695-1708, 2011.