Sampling-Based Path Planning for Multi-robot Systems with Co-safe Linear Temporal Logic Specifications

Felipe J. Montana[1(✉)], Jun Liu[2], and Tony J. Dodd[1]

[1] Department of Automatic Control and Systems Engineering, University of Sheffield, Sheffield, UK
{fjmontanagonzalez1,t.j.dodd}@sheffield.ac.uk
[2] Department of Applied Mathematics, University of Waterloo, Waterloo, Canada
j.liu@uwaterloo.ca

Abstract. This paper addresses the problem of path planning for multiple robots under high-level specifications given as syntactically co-safe linear temporal logic formulae. Most of the existing solutions use the notion of abstraction to obtain a discrete transition system that simulates the dynamics of the robot. Nevertheless, these solutions have poor scalability with the dimension of the configuration space of the robots. For problems with a single robot, sampling-based methods have been presented as a solution to alleviate this limitation. The proposed solution extends the idea of sampling methods to the multiple robot case. The method samples the configuration space of the robots to incrementally constructs a transition system that models the motion of all the robots as a group. This transition system is then combined with a Büchi automaton, representing the specification, in a Cartesian product. The product is updated with each expansion of the transition system until a solution is found. We also present a new algorithm that improves the performance of the proposed method by guiding the expansion of the transition system. The method is demonstrated with examples considering different number of robots and specifications.

1 Introduction

Motion planning based on high-level temporal specifications has become an important area of research. Several methods have been developed for single robots, e.g., [5, 15, 18, 22]; and for multiple robots, e.g., [2, 6, 19]. The multi-robot path planning problem with linear temporal logic (LTL) specifications can be categorised into two areas depending on the final goal: (i) each robot has its own task, or (ii) all the robots act as a team trying to accomplish a global specification. In general, to find a path that satisfies an LTL specification, most of the methods use the notion of equivalent abstraction [1] to create a finite transition system that models the motion of the robot. Then, a product automaton is created using this transition system and a Büchi automaton that represents the LTL specification. In this product automaton, a graph search is performed to find

Felipe J. Montana is supported by the Mexican National Council of Science and Technology (CONACyT). Jun Liu is supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada and the Canada Research Chairs (CRC) program.

a path satisfying the specification. When a single task has to be completed by all the robots, a parallel composition of the individual transition systems can be created to model the motion of all the robots as a group. Then, this composition is used to create a product automaton with the Büchi automaton as in the single robot case. Although this method can find a solution, it is computationally expensive and scales poorly with the number of robots [10].

To avoid the parallel composition, in [2], the authors present a method to decompose the global specification into local specifications. Then, individual strategies are computed for the robots. Using a similar approach, in [12], the problem of gathering information from an environment while the motion of the robots is constrained by a temporal logic specification is solved. Distributability has been also used to find robust paths when the travelling time of the robots is uncertain [20] and for nonholonomic robots [23]. Although these methods avoid the parallel composition by decomposing the specification, the approaches fail to find a solution, even if one exists, when the global specification is not distributable among the robots.

A common similarity of the works aforementioned is the assumption of a transition system obtained by the process of abstraction described above. A limitation of this approach is its complexity. They scale at least exponentially with the dimension of the configuration space of the robots [21]. Using sampling-based methods, this problem has been addressed by sampling the continuous configuration space and incrementally constructing a transition system until the specified task can be accomplished. In [8], the authors use an incremental model checking method to solve the problem when μ-calculus formulae are used to express the specifications. In [21], a method that uses a sparse sampling to reduce the number of states in the transition system is presented. These methods scale well since all the operations performed to find a path increment only with the number of samples. The previous methods only consider a single robot. For the multi-robot problem, in [7], a sampling-based method is used to create a tree that approximates the product automaton. This approximation permits to solve large problems, in terms of the number of states in the product automaton, that are not solvable considering the product automaton itself. Nevertheless, in contrast to the solution proposed in this paper, they sample states from a transition system representing regions of the environment and not from the configuration space of the robots.

In this paper we present a sampling-based method that explores the configuration space of a group of robots to find a path such that a global specification is satisfied. The proposed method explores an implicit representation of a composite roadmap that models the motion of all the robots as a group. During the exploration, a transition system is incrementally expanded by adding new states from individual roadmaps. With each expansion, the product automaton of the transition system and a Büchi automaton is updated. Although a solution can be found by naively exploring the composite roadmap, this process could require long time. To improve this time, we also present an algorithm that uses the Büchi automaton of the specification to guide the exploration of the composition. The main contribution of this paper is a novel method that combines a sampling-based method for multiple robots with a new algorithm that allows fast computation of solutions.

The rest of the paper is organised as follows. Preliminaries and a formal definition of the problem addressed are presented in Sect. 2. A detailed presentation of the proposed method is found in Sect. 3. The method is demonstrated with three examples in Sect. 4 and the conclusion is presented in Sect. 5.

2 Preliminaries and Problem Definition

2.1 Deterministic Transition System

A deterministic transition system is a tuple $\mathscr{T} = (S, s_0, \delta_{\mathscr{T}}, \Pi, L)$, where:

- S is a finite set of states,
- $s_0 \in S$ is an initial state,
- $\delta_{\mathscr{T}} \subseteq S \times S$ is a transition relation,
- Π is a finite set of atomic propositions,
- $L : S \to 2^{\Pi}$ is a labelling function.

A run on $\mathscr{T}$ is a sequence $\sigma = s_0 s_1 \ldots$ such that for every $i \geq 0$, $(s_i, s_{i+1}) \in \delta_{\mathscr{T}}$. The trace of a run σ, $\omega = L(s_0)L(s_1)\ldots$, is a word over the power set of Π that defines the atomic propositions that evaluate true in the states of the run.

2.2 Linear Temporal Logic

We use a segment of LTL, called syntactically co-safe LTL (sc-LTL) [11], to express the desired system behaviour. LTL formulae are built from atomic propositions $\pi \in \Pi$ that indicate whether a property of the system is true or false.

Syntax: The syntax of sc-LTL over Π is defined as follows:

$$\varphi := \pi \mid \neg\pi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc \varphi \mid \varphi_1 \mathscr{U} \varphi_2 \ ,$$

where $\pi \in \Pi$ is an atomic proposition, φ is a sc-LTL formula; and $\neg$, $\vee$, $\wedge$, $\bigcirc$ and $\mathscr{U}$ represent the operators *negation*, *disjunction*, *conjunction*, *next* and *until*, respectively. Other operators such as the temporal operator *eventually*, $\Diamond\pi = True\mathscr{U}\,\pi$, can be derived from the operators presented above.

Semantics: The semantics of LTL are defined over words ω. Given an LTL specification φ, a run $\sigma = s_i s_{i+1} \ldots$, and the satisfaction relation $\models$, the semantics are defined inductively as follows:

$$s_i \models \pi \text{ iff } \pi \in L(s_i) \ ,$$
$$s_i \models \varphi_1 \wedge \varphi_2 \text{ iff } s_i \models \varphi_1 \text{ and } s_i \models \varphi_2 \ ,$$
$$s_i \models \varphi_1 \vee \varphi_2 \text{ iff } s_i \models \varphi_1 \text{ or } s_i \models \varphi_2 \ ,$$
$$s_i \models \bigcirc\varphi \text{ iff } s_{i+1} \models \varphi \ ,$$
$$s_i \models \varphi_1 \mathscr{U} \varphi_2 \text{ iff } \exists j \geq i : s_j \models \varphi_2 \text{ and } s_k \models \varphi_1, \forall\, i \leq k < j \ .$$

LTL formulae in positive normal form, where negations only occur in front of atomic propositions, and which only use the operators $\bigcirc$, $\mathscr{U}$ and $\Diamond$, are co-safe formulae [11].

2.3 Büchi Automaton

Given an LTL specification, it is possible to construct a Büchi automaton, which accepts only words ω that satisfy the specification. A Büchi automaton $\mathscr{B}$ is a tuple $\mathscr{B} = (\Sigma, Q, q_0, \delta_{\mathscr{B}}, Q_F)$, where:

- $\Sigma = 2^{\Pi}$ is a finite alphabet,
- Q is a finite set of states,
- $q_0 \in Q$ is an initial state,
- $\delta_{\mathscr{B}} : Q \times \Sigma \to Q$ is a transition function,
- $Q_F \subseteq Q$ is a set of accepting states.

A run on $\mathscr{B}$, produced by a word ω over the alphabet Σ, is a sequence $\rho = q_0 q_1 \ldots$ such that for every $i \geq 0$, there exists $\pi_i \in \Sigma$ and $\delta_{\mathscr{B}}(q_i, \pi_i) = q_{i+1}$. For sc-LTL formulae, an infinite word ω is accepted if it starts with a prefix such that the produced run ρ reaches the set Q_F of final states. An example of a specification and its Büchi automaton is shown in Fig. 1.

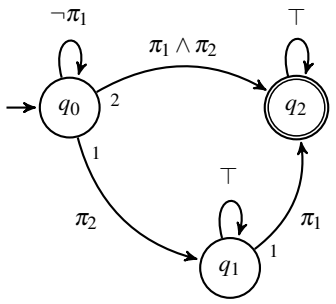

Fig. 1. Büchi automaton of formula $\varphi = (\neg\pi_1 \mathscr{U} \pi_2) \wedge \Diamond\pi_1$, where $\pi_1, \pi_2 \in \Sigma$ are atomic propositions, $\top$ is unconditionally true and $\mathscr{U}$, $\Diamond$ are the operator *until* and *eventually*, respectively. The formula indicates that the atomic propositions π_2 and π_1 have to be satisfied in that specific order or at the same time. The small numbers on the edges are used to identify each transition (see Sect. 3.4). The initial and final states, q_0 and q_2, are indicated with an arrow and a double circle, respectively.

Given a Büchi automaton $\mathscr{B} = (\Sigma, Q, q_0, \delta_{\mathscr{B}}, Q_F)$, let $|Q|$,$|q|$ and $AP(q, q')$ denote the cardinality of Q, the number non self-transitions from state q and the set of atomic propositions required for a transition from q to q', i.e., $AP(q, q') = \pi$ if $\delta_{\mathscr{B}}(q, \pi) = q'$.

2.4 Product Automaton

Given a transition system $\mathscr{T}$ and a Büchi automaton $\mathscr{B}$, the product automaton $\mathscr{P} = \mathscr{T} \times \mathscr{B}$ is defined by the tuple $\mathscr{P} = (S_{\mathscr{P}}, s_{\mathscr{P},0}, \delta_{\mathscr{P}}, S_{\mathscr{P},F})$, where:

- $S_{\mathscr{P}} = S \times Q$ is a finite set of states,

- $s_{\mathscr{P},0} = s_0 \times q_0$ is an initial state,
- $\delta_{\mathscr{P}} \subseteq S_{\mathscr{P}} \times S_{\mathscr{P}}$ is a transition relation, where $((s,q),(s',q')) \in \delta_{\mathscr{P}}$ iff $(s,s') \in \delta_{\mathscr{T}}$ and $\delta_{\mathscr{B}}(q, L(s')) = q'$,
- $S_{\mathscr{P},F} = S \times Q_F$ is a set of accepting states.

2.5 Problem Formulation

We consider R robots operating in a workspace containing obstacles and areas of interest, defined by disjoint regions, that are associated with atomic propositions. These atomic propositions are used to define sc-LTL formulae such as $\varphi_1 = \Diamond \pi_1$ or $\varphi_2 = \neg(\pi_1 \vee \pi_2) \mathscr{U} (\pi_1 \wedge \pi_2)$. Formula φ_1 indicates that a robot has to visit the area associated with the atomic proposition π_1 while the formula φ_2 indicates that the areas π_1 and π_2 cannot be visited until they are reached at the same time step.

Let $X^i \subset \mathbb{R}^n$ be a compact set defining the configuration space of a robot i, where i is an element of the set $\mathfrak{R} = \{1, \ldots, R\}$ that indexes the robots and $\mathbb{R}^n$ is the n-dimensional Euclidean space. Each robot has an obstacle-free configuration space X^i_{free}. The configuration space of the full system is denoted as $X = \prod_{i \in \mathfrak{R}} X^i$. The obstacle-free space $X_{\text{free}} = \prod_{i \in \mathfrak{R}} X^i_{\text{free}}$ does not include states where collision between robots occurs. Let $\boldsymbol{x} = x_0 x_1 \ldots$, where $x_j = (x^1_j, \ldots, x^R_j)$ for all $j \geq 0$, be a sequence of configurations describing a path followed by the full system. A path is collision free if $x_j \in X_{\text{free}}$ for all $j \geq 0$. To interpret atomic propositions over the configuration space X, let $L : X \to 2^{\Pi}$ be a function that maps a configuration x to the atomic propositions satisfied by the configuration. Hence, a word $\omega = L(x_0)L(x_1)\ldots$ expresses a path $\boldsymbol{x}$ in terms of the atomic propositions. We say that the path $\boldsymbol{x}$ satisfies the sc-LTL specification φ if the word ω, produced by $\boldsymbol{x}$, is accepted by the Büchi automaton that accepts words satisfying φ.

Problem definition: Given a group of R robots with initial configuration x^i_0 for $i \in \mathfrak{R}$ and a sc-LTL specification φ, find a collision-free path $\boldsymbol{x}$ such that φ is satisfied.

3 Solution

This section firstly presents an overview of the proposed method followed by a detailed explanation. The main idea of the method is to create a graph, called transition system, modelling the motion of all the robots as a single system. Each vertex of the graph represents a combination of single configurations of all the robots. Edges represent collision-free paths between these configurations. Initially, the graph contains only one vertex, the initial configuration of all the robots. Then, this graph is incrementally expanded by adding a new vertex and transitions. To obtain the new vertex, individual graphs, called roadmaps, that model the motion of each robot are used. The process of expanding the graph is repeated until the specification can be satisfied by a path in the graph. To improve the required time to find a solution, the method uses an algorithm that guides the expansion of the graph. Section 3.1 explains the creation of the individual roadmaps. In Sects. 3.2 and 3.3, the incremental construction of the transition system and the search for a path satisfying the specification are presented. In Sect. 3.4, the algorithm that guides the expansion is explained in detail. Finally, illustrative examples and conclusions are presented in Sect. 4 and 5, respectively.

3.1 Probabilistic Roadmap

The first step of the proposed method consists of creating probabilistic roadmaps [9] for each robot $i \in \mathfrak{R}$. A roadmap of a robot i models a subset of the possible trajectories of the robot and is formed by a set of sampled configurations $x \in X^i_{\text{free}}$ connected by collision-free paths. A graph $G^i = (V^i, E^i)$ is used to represent the roadmap of the robot i. Each vertex $v \in V^i$ is associated with an unique robot configuration $x \in X^i_{\text{free}}$. This association is given by the function $\chi : V^i \to X^i$. Connectivity between two configurations is represented by an edge $(v, v') \in E^i$. We refer to all the vertices v' that share an edge with v as neighbours of v. To verify the satisfaction of a specification using only the atomic propositions that are true in each state of the transition system, see Sect. 3.3, we limit the edges between vertices to those edges that intersect the boundary of a region in the workspace at most once [21]. Moreover, we reduce the size of the roadmaps by constructing sparse roadmaps [4]. Since each vertex $v \in V^i$ is associated with a configuration $x \in X^i_{\text{free}}$, with abuse of notation, we use $L(v)$ to denote the atomic propositions satisfied by $\chi(v)$. The set of vertices on a roadmap G^i that satisfy an atomic proposition $\pi \in \Pi$ is denoted by $[\![\pi]\!]_i$.

To consider the configuration of all the robots, a composite roadmap [17] $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ is constructed as the tensor product of the individual roadmaps $\{G^i\}_{i=1}^R$. Formally, $\nu = (v^1, \ldots, v^R)$ is a vertex of $\mathbb{G}$ if $v^i \in V^i$ for all $i \in \mathfrak{R}$ and $\chi(\nu) \in X_{\text{free}}$. Let $\nu = (v^1, \ldots, v^R)$ and $\nu' = (v'^1, \ldots, v'^R)$ be two vertices in $\mathbb{G}$. In a tensor product, an edge $(\nu, \nu') \in \mathbb{E}$ is defined if for every $i \in \mathfrak{R}$, $(v^i, v'^i) \in E^i$. The projection of a composite vertex $\nu \in \mathbb{V}$ onto the vertex $v^i \in V^i$ of robot i is denoted by $\nu \downarrow_i$, i.e., $\nu \downarrow_i = v^i$. The atomic propositions satisfied by a vertex $\nu = (v^1, \ldots, v^R)$ is the union of the atomic propositions satisfied by the individual vertices forming ν, i.e., $L(\nu) = \cup_{i=1}^R L(v^i)$, where $v^i = \nu \downarrow_i$.

As explained in Sect. 1, it is possible to find a path for each robot satisfying a specification by creating a product automaton of the composite roadmap $\mathbb{G}$ and the Büchi automaton $\mathscr{B}$ of the specification φ. Nevertheless, this procedure is only applicable for small problems due to its poor scalability; the number of vertices in $\mathbb{G}$ is $|V|^R$. Instead, we implicitly represent $\mathbb{G}$ and perform a sampling of it until a solution is found. Algorithm 1, explained in the rest of Sect. 3, shows this procedure.

Algorithm 1 IncrementalExpansion ($\{G^i\}_{i=1}^R$,$\mathscr{B}$)

1: $S \leftarrow s_0 = (v_0^1, v_0^2, \ldots, v_0^R)$
2: $\mathscr{P} = \mathscr{T} \times \mathscr{B}$
3: **while** $s_{\mathscr{P}} = (s, q) \notin S_{\mathscr{P}} : q \in Q_F$ **do**
4: $\quad \mathscr{T} \leftarrow$ Explore($\{G^i\}_{i=1}^R$,$\mathscr{T}$)
5: $\quad \mathscr{P}, S'_{\mathscr{P}} \leftarrow$ Update($\mathscr{P}$,$\mathscr{B}$,$\mathscr{T}$)
6: $\quad \mathscr{T} \leftarrow$ LocalConnector($\{G^i\}_{i=1}^R$,$\mathscr{B}$,$\mathscr{T}$,$S'_{\mathscr{P}}$)
7: $\quad$ **while** new connection **do**
8: $\quad\quad \mathscr{P}, S'_{\mathscr{P}} \leftarrow$ Update($\mathscr{P}$,$\mathscr{B}$,$\mathscr{T}$)
9: $\quad\quad \mathscr{T} \leftarrow$ LocalConnector($\{G^i\}_{i=1}^R$,$\mathscr{B}$,$\mathscr{T}$,$S'_{\mathscr{P}}$)

3.2 Composite Roadmap Exploration

In this subsection, the incremental exploration of the composite configuration space $\mathbb{G}$ is presented. First, a transition system $\mathscr{T}$ is initialised with only the vertex corresponding to the initial configuration of all robots, i.e., $s_0 = v_0 = (v_0^1, v_0^2, \ldots, v_0^R)$, where $\chi(v_0^i) = x_0^i \; \forall i \in \mathfrak{R}$ (Alg. 1, line 1). Vertices v added to the transition system are represented as s. In each iteration of Alg. 1, a new state is added to $\mathscr{T}$ using the idea of discrete rapidly-exploring random trees [16] as follows (Alg. 1, line 4).

Unless some conditions, explained in Sect. 3.4, are satisfied, a state $s = (v^1, \ldots, v^R) \in S$ is randomly selected from the transition system $\mathscr{T}$. Consider a single vertex v^i forming s and recall that $v^{i,j}$ is a neighbour of v^i in G^i if $(v^i, v^{i,j}) \in E^i$. The rays $\rho_{v^i, v^{i,j}}$, for all $j \in \{1, \ldots, l\}$, that start from v^i and pass through the l neighbours of v^i are computed. Then, a configuration x_{sample} is sampled from X^i_{free} and the ray $\rho_{v^i, x_{\text{sample}}}$ is calculated. To choose a neighbour of v^i in direction of x_{sample}, the angles between the ray $\rho_{v^i, x_{\text{sample}}}$ and each of the rays $\rho_{v^i, v^{i,j}}$ are computed. The neighbour vertex that generates the ray with the smallest angle is selected and denoted as v^i_{new}, Fig. 2. This process is repeated for all the vertices forming s, resulting in a candidate state $s_{\text{new}} = (v^1_{\text{new}}, \ldots, v^R_{\text{new}})$.

Before adding s_{new} to $\mathscr{T}$, it is verified whether collision between robots exists. To avoid collisions during the transitions (v^i, v^i_{new}) for all $i \in \mathfrak{R}$, priorities are assigned to each robot according to the following rules [3]: (i) if robot i, transitioning from v^i to v^i_{new}, causes a collision with robot j, located in v^j_{new}, the robot i receives higher priority than j; (ii) if robot i collides with robot j placed in v^j during the transition

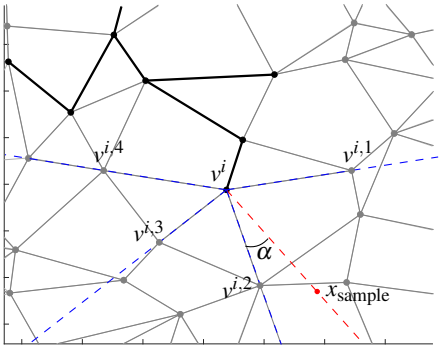

Fig. 2. Selection of vertex and edge in roadmap G^i. The states and transitions in $\mathscr{T}$ are illustrated with black vertices and edges. The roadmap G^i is shown in grey. To choose which neighbour $\{v^{i,j}\}_{j=1}^4$ of v^i is added to the $\mathscr{T}$, a configuration x_{sample} is randomly sampled from X^i_{free}. The rays starting from v^i and passing through x_{sample} and the neighbours are shown as red and blue dotted lines, respectively. The angles between the ray of the sample and the rest of the rays are computed. The smallest angle, α in the figure, determines which neighbour and edge are added to $\mathscr{T}$, neighbour $v^{i,2}$ in this example.

(v^i, v^i_{new}), then, robot i receives lower priority than j. The state s_{new} is discarded if there is no ordering such that collisions are avoided. Otherwise, the state is added to $\mathscr{T}$ with the transitions (s, s_{new}) and (s_{new}, s). Note that by choosing only neighbours of each individual vertex v^i, $(v^1_{\text{new}}, \ldots, v^R_{\text{new}})$ is an element of the composite roadmap $\mathbb{G}$. Intuitively, the transition system $\mathscr{T}$ represents the explored part of $\mathbb{G}$. An example of such exploration, for the case $R = 1$, is shown in Fig. 3.

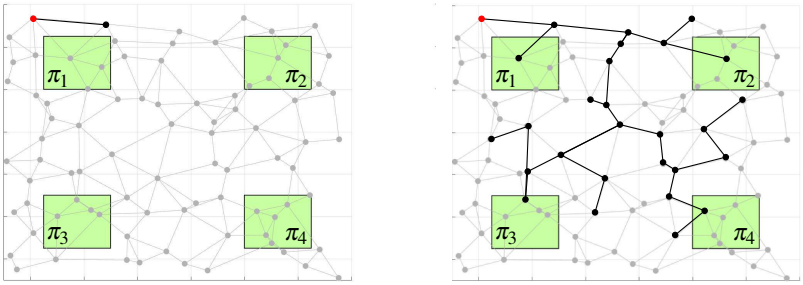

Fig. 3. Incremental construction of a transition system. The roadmap of the robot is shown in grey. The transition system, representing the explored part of the roadmap, is shown in black. The green areas, identified by the atomic propositions π_1, π_2, π_3 and π_4, are regions of interest. The proposed method iteratively adds vertices and edges from the roadmap to the transition system until the specifications can be satisfied. In this example, the specification is to visit the four green areas. The initial configuration of the robot is shown as a red vertex.

Since each vertex v of $\mathbb{G}$ is associated with a configuration $x \in X_{\text{free}}$, a run $\sigma = s_0 s_1 \ldots$ on $\mathscr{T}$ represents a path of the full system in the configuration space X_{free}. Hence, this exploration continues until a path that satisfies the specification φ is found. The procedure to determine whether the current transition system contains such a path is presented in the next subsection.

3.3 Product Automaton Update

Based on model checking techniques, the verification of a run σ satisfying the sc-LTL specification φ is made on the Cartesian product $\mathscr{P} = (S_{\mathscr{P}}, s_{\mathscr{P},0}, \delta_{\mathscr{P}}, S_{\mathscr{P},F})$ of $\mathscr{T}$ and the Büchi automaton $\mathscr{B}$ obtained from φ. States $s_{\mathscr{P}} \in S_{\mathscr{P}}$ are formed by pairs (s, q), where $s = (v^1, \ldots, v^R) \in S$ is a state of $\mathscr{T}$ and $q \in Q$ is a state of $\mathscr{B}$.

The product automaton $\mathscr{P}$ is firstly created when the transition system contains only the initial state s_0 (Alg. 1, line 2). Since the transition system $\mathscr{T}$ changes with each new state s_{new}, the product $\mathscr{P}$ requires to be updated (Alg. 1, line 5). The procedure to incrementally update $\mathscr{P}$ [21] and to search for a path satisfying φ is now presented.

When a new state s_{new} is added to $\mathscr{T}$ with the transition (s, s_{new}) and (s_{new}, s), the set $S'_{\mathscr{P}}$ of states $s'_{\mathscr{P}} = (s_{\text{new}}, q')$, such that $\delta_{\mathscr{B}}(q, L(s_{\text{new}})) = q'$ and $(s, q) \in S_{\mathscr{P}}$, is

computed. Then, for each state $s'_{\mathscr{P}} \in S'_{\mathscr{P}}$, it is verified if $s'_{\mathscr{P}}$ is already in $S_{\mathscr{P}}$. If that is not the case, the state is added to $\mathscr{P}$ and is removed from $S'_{\mathscr{P}}$. Moreover, the set of states $s''_{\mathscr{P}} = (s', q'')$, such that $(s_{\text{new}}, s') \in \delta_{\mathscr{T}}$ and $\delta_{\mathscr{B}}(q', L(s')) = q''$, is computed. If a state $s''_{\mathscr{P}}$ is not already in $S_{\mathscr{P}}$, $s''_{\mathscr{P}}$ is added to $S_{\mathscr{P}}$ and to $S'_{\mathscr{P}}$. This recursive procedure continues until the set $S'_{\mathscr{P}}$ is empty.

By construction, if a run on $\mathscr{P}$, starting from $s_{\mathscr{P},0}$, reaches the set $S_{\mathscr{P},F}$ of accepting states, the language produced by the run σ on $\mathscr{T}$ generates a word ω that is accepted by the Büchi automaton $\mathscr{B}$ computed from the sc-LTL formula. In other words, a run σ on $\mathscr{T}$ satisfies the specification if a run on $\mathscr{P}$ reaches the set of accepting states. Hence, the process of exploring $\mathscr{B}$ and updating $\mathscr{P}$ continues until a state $s_{\mathscr{P}} = (s, q)$ is added to $\mathscr{P}$ such that $q \in Q_F$ (Alg. 1, line 3).

A solution to the problem defined in Section 2.5 would be eventually found by repeating the process described above. Nevertheless, depending on the number of robots and the specification, this process could take an impractical amount of time. In the next subsection, an algorithm that improves the required time by guiding the exploration of $\mathbb{G}$ is presented.

3.4 Guided Expansion

In this subsection, we present a new algorithm, called local connector, which selects the states in $\mathscr{T}$ that must be expanded in order to satisfy the sc-LTL specification. The main idea is to find the shortest path, in terms of transitions, in the Büchi automaton to an accepting state and to search for vertices in the individual roadmaps $\{G^i\}_{i=1}^R$ satisfying the atomic propositions required to progress in such a path.

Before explaining the algorithm, some concepts and notation are introduced. The algorithm monitors which transitions of the Büchi automaton have been satisfied by the current transition system. To achieve this, each state and non-self transition of the Büchi automaton are identified by an index, Fig. 1. When one of the $|q_i|$ outgoing transitions from state $q_i \in Q$ is satisfied, the index that identifies the transition is added to the set D^i_δ. Once all the outgoing transitions of a state $q_i \in Q$ are satisfied, i.e., $|D^i_\delta| = |q_i|$, the index i is added to the set D_q.

Depending on the transition, one or more atomic propositions have to be satisfied at the same time. Since each robot can satisfy one atomic proposition at a time, when more than one proposition is required, collaboration between robots is needed. Consider the transition (q_0, q_2) in the Büchi automaton shown in Fig. 1 as an example. To make this transition, the atomic propositions π_1 and π_2 have to be satisfied, i.e., $AP(q_0, q_2) = \{\pi_1, \pi_2\}$. The algorithm verifies if a robot i is able to satisfy any of the propositions by finding a local path from its current configuration to a configuration in G^i satisfying one of the atomic propositions. If, for instance, a path exists to a vertex $v \in V^i$ satisfying π_1, i.e., $v \in [\![\pi_1]\!]_i$, the atomic proposition π_1 is added to the set D_Π and the index that identifies the robot is added to the set W_R. The set W_R identifies the robots that are waiting for other robots to satisfy the remaining atomic propositions required to make the transition in the Büchi automaton. Moreover, this set is used to guide the expansion of $\mathscr{T}$ as presented below. To identify which transition is tried to be satisfied when a robot is added to W_R, the index identifying the state q and its outgoing transition are

Algorithm 2 LocalConnector ($\{G_i\}_{i=1}^{R}$,$\mathscr{B}$,$\mathscr{T}$,$S'_{\mathscr{P}}$)

1: $g_i \leftarrow \{s : (s,q_i) \in S'_{\mathscr{P}},\ \forall i \in \{1,\dots,|Q|\}\}$
2: **for** $i \in sorted(q_i) : i \in \{1,\dots,|Q|\}, g_i \neq \emptyset$ and $i \notin D_q\}$ **do**
3: **for** $j \in \{\Re : j \notin W_R\}$ **do**
4: **for** $n = 1 \rightarrow |g_i|$ **do**
5: $v_s = s_n \downarrow_j$, $s_n \in g_i$ ▷ Vertex to be connected
6: **for** $k \in sorted(\delta_{\mathscr{B}}(q_i,\cdot)) : k \in \{1,\dots,|q_i|, W_\delta \neq \emptyset \rightarrow k \in W_\delta$ and $k \notin D^i_\delta\}$ **do**
7: $\Pi_{\text{req}} = AP(q_i,q_k)$ ▷ Set of AP required
8: **for** $m \in \{1,\dots,|\Pi_{\text{req}}| : m \notin D_\Pi\}$ **do**
9: **for** $h = 1 \rightarrow |[\![\pi_m]\!]_j|$ **do**
10: $v_t = v_h$, $v_h \in [\![\pi_m]\!]_j$ ▷ Target vertex
11: **if** Connect (v_s, v_t) **then**
12: **if** $|\Pi_{\text{req}}| = 1$ **then**
13: $s_{\text{new}} = s$, $s \in g_i$
14: $s_{\text{new}} \downarrow_j = v_t$ ▷ New state of $\mathscr{T}$
15: $D^i_\delta \leftarrow D^i_\delta \cup k$
16: **else if** $|D_\Pi| < |\Pi_{\text{req}}| - 1$ **then**
17: $W_R \leftarrow W_R \cup j$ ▷ robot j can satisfy π_m
18: $W_q \leftarrow W_q \cup i$
19: $W_\delta \leftarrow W_\delta \cup k$
20: $D_\Pi \leftarrow D_\Pi \cup m$
21: $v_{j,\text{next}} = v_t$ ▷ Vertex satisfying π_m
22: **else if** $|D_\Pi| = |\Pi_{\text{req}}| - 1$ **then**
23: $s_{\text{new}} = s$, $s \in g_i$
24: $s_{\text{new}} \downarrow_p = v_{p,\text{next}}$, $\forall p \in W_R$
25: $s_{\text{new}} \downarrow_j = v_t$ ▷ New state of $\mathscr{T}$
26: $D^i_\delta \leftarrow D^i_\delta \cup k$
27: $W_R = \emptyset, W_q = \emptyset, W_\delta = \emptyset, D_\Pi = \emptyset$

added to the sets W_q and W_δ, respectively. We now explain the algorithm in detail, see Alg. 2.

The algorithm receives as input the Büchi automaton $\mathscr{B}$ and the set $S'_{\mathscr{P}}$ of states added to the product automaton $\mathscr{P}$ after the last update. These states have the form (s,q), where $s = (v^1,\dots,v^R) \in S$ and $q \in Q$. The states are divided into different groups depending on their Büchi state component (Alg. 2, line 1). In other words, for each state q_i in the Büchi automaton, a group g_i containing states $s = (v^1,\dots,v^R)$ such that $s_{\mathscr{P}} = (s,q_i) \in S'_{\mathscr{P}}$ is created. The algorithm eliminates the group g_i if there is no remaining outgoing transitions from the Büchi state q_i to be satisfied, i.e., $i \in D_q$. Then, the algorithm sorts, from shortest to longest, the different paths from the initial state $q_0 \in Q$ to the closest accepting state $q \in Q_F$ in the the Büchi automaton $\mathscr{B}$.

Using these sorted paths, the algorithm tries to reach atomic propositions required in the paths, starting from the shortest path (Alg. 2, line 2). An exception to the order is made when of one the robots is waiting for another atomic proposition to be satisfied, i.e., $W_R \neq \emptyset$. In this case, all the Büchi states are ignored except the states in the set W_q.

For each state s in g_i, the individual vertices of non-waiting robots forming s, i.e., $s \downarrow_j$ for $j \in \Re \setminus W_R$, are considered to be connected to vertices in G^j satisfying the

required atomic propositions in the Büchi automaton transition. An individual vertex, denoted as v_s, is considered for connection in each iteration (Alg. 2, lines 3-5). If a robot is waiting, all the transitions are ignored except the transition indicated by the set W_δ. Otherwise, the transition is selected based on the sorted paths (Alg. 2, line 6). The required atomic propositions in the transition are assigned to the set Π_{req} (Alg. 2, line 7). Then, all the vertices in the roadmap G^j that satisfy an atomic proposition that cannot be satisfied by a waiting robot, i.e., $\pi_m \in \Pi_{\text{req}} \setminus D_\Pi$, are assigned as a target of the connection and are denoted as v_t (Alg. 2, lines 8-10). The algorithm then tries to find a path between the vertices v_s and v_t. By connecting the transition system to vertices satisfying atomic propositions required for the specification, the time needed to solve the proposed problem is reduced.

In order to find a path between the vertices v_s and v_t any method can be used. However, because this process is constantly repeated, a method that sacrifices completeness for speed is preferred. In this work, the algorithm attempts to connect two vertices if the Euclidean distance between them is less than a pre-established value. If the path, given by a line between the vertices, is collision free, the connection is considered successful (Alg. 2, line 11). Depending on the number of atomic propositions in the selected transition in $\mathscr{B}$, three different situations can occur:

Case 1: Only one atomic proposition is required in the transition, i.e., $|\Pi_{\text{req}}| = 1$ (Alg. 2, lines 12-15). In this case, if robot j can satisfy the required atomic proposition through the connection, a new state $s_{\text{new}} = (v^1, \ldots, v^R)$, where $v^i = s\downarrow_i$ for $i \in \{\Re : i \neq j\}$ and v_t otherwise, is created. Intuitively, the new state has the same components as the composite state s, except the element of robot j that is replaced by v_s. If the new state is not in the transition system $\mathscr{T}$, the state is added with the transitions (s, s_{new}) and (s_{new}, s). Finally, the index k of the satisfied transition is added to D^i_δ indicating that the transition k from state q_i has been satisfied.

Case 2: More than one atomic proposition is required and at least one more is still required after the connection (Alg. 2, lines 16-21). When a robot j can satisfy one of the required atomic propositions but at least another is needed for the transition in the Büchi automaton, the robot stays in the vertex v_s waiting for the remaining robots to satisfy the other atomic propositions. To indicate that the robot is waiting, the index j is added to the set W_R. The set W_R restricts the states that can be selected in the exploration of $\mathbb{G}$. The selected state in the exploration must be formed by the vertices v^i, where $i \in W_R$. Moreover, the next state s_{new} to be added to the transition system must have the same vertices. After adding the index j of the robot to the set W_R, the vertex that can be reached, i.e., v_t, is saved in $v_{j,\text{next}}$ to be used once all the atomic propositions of the transition are satisfied. Then, the index m of the atomic proposition that can be satisfied is added to the set D_Π to skip this atomic proposition the next iteration of Alg. 2. Note that the restriction explained above guides the sampling process of $\mathbb{G}$.

Case 3: The last required atomic proposition is satisfied with the connection (Alg. 2, lines 22-27). Similar to case 1, when a robot j can satisfy the last required atomic proposition, a new state s_{new} is created with the saved states $v_{i,\text{next}}$, i.e., $s_{\text{new}}\downarrow_i = v_{i,\text{next}}$ for all $i \in W_R$, v_t for $i = j$ and $s\downarrow_i$ otherwise. This state is added to $\mathscr{T}$ with the transitions (s, s_{new}) and (s_{new}, s). The index k of the satisfied transition is added to D^i_δ and the sets

W_R, W_q and W_δ and D_Π become empty indicating that all the robots can move again and any non-satisfied state and transition in the Büchi automaton can be selected.

Every time a new state s_{new} is added to $\mathscr{T}$, the product automaton $\mathscr{P}$ is update (Alg. 1, lines 7-9) and the process is repeated. As mentioned in Sect. 3.3, Alg. 1 stops once a product state with a final state $q \in Q_F$ is added to $\mathscr{P}$.

3.5 Implementation

This subsection presents how a solution is obtained from $\mathscr{P}$ and the implementation of it in the robots. Once the condition to stop Alg. 1 is satisfied, the shortest path $\sigma = s_{\mathscr{P},0} \dots s_{\mathscr{P},n}$ on $\mathscr{P}$, where $s_{\mathscr{P},n} \in S_{\mathscr{P},F}$, is sought. Since a state $s_{\mathscr{P}}$ is formed by the pair (s,q), only the first element of each state is considered to create the path $\boldsymbol{x}$ that satisfies the sc-LTL specification. The function χ is used to define the configurations in X that defines $\boldsymbol{x}$. Finally, each configuration in $\boldsymbol{x}$ is projected to the individual configuration spaces X^i to define a path for each robot.

To execute the path, each robot stores a list of the vertices to visit in its roadmap G^i together with the configurations where the robot has to wait for other robots before performing a transition. When a robot finishes a transition, it broadcasts a unique identifier number and a signal indicating that the transition has been completed. If a robot needs to wait for other robots, the transition is not performed until the robot receives the signal of all the robots with higher priority.

4 Examples

The proposed method is illustrated with different sc-LTL specifications and number of robots. We consider a differential wheeled robot, called e-puck [14], in a workspace with 4 areas associated with the atomic propositions π_1, π_2, π_3 and π_4, Fig. 4. The computation of the path is implemented in MATLAB on a computer with a 3.1 GHz i7 processor and 8GB of RAM. The dynamics of the e-pucks are simulated using Enki [13].

We present three examples, considering two, three and four robots, respectively:

1. Regions π_1, π_2 have to be visited at the same time as well as π_3, π_4 with the same restriction, Fig. 4.
2. Regions π_1, π_2 and π_3 must be visited in the presented order.
3. Regions π_1, π_2, π_3 and π_4 cannot be visited until all of them are visited at the same time.

Recall that the number of vertices in $\mathbb{G}$ is equal to $|V|^R$, where $|V|$ is the number of vertices in the roadmap and R is the number of robots. In the example with four robots, the parallel composition $\mathbb{G}$ has more than 96 million vertices. Nevertheless, Table 1 shows that only a small portion of $\mathbb{G}$ is explored before finding a solution. This result can be attributed to the guided search performed by the local connector algorithm. For comparison, we compute a solution to the first specification without Alg. 2. That is, we only expand the transition system using the idea presented in Sect.

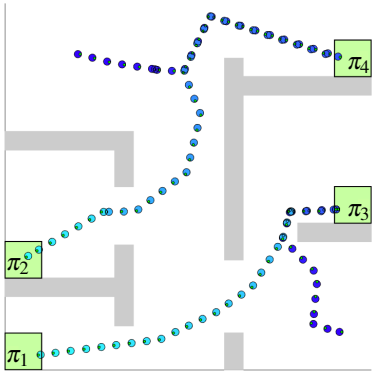

Fig. 4. Illustration of the path followed by two robots satisfying the specification $\varphi = \Diamond(\pi_1 \wedge \pi_2) \wedge \Diamond(\pi_3 \wedge \pi_4)$. This sc-LTL specification requires the robots to visit areas marked as π_1 and π_2 at the same time and the areas π_3 and π_4 with the same restriction. The colour of the robots changes, from darker to lighter blue, over time to show that the atomic propositions are satisfied at the same time step.

Table 1. Average number of states in $\mathscr{T}$ and required time to solve the problem over 20 different runs.

Specification	Robots	States in $\mathscr{T}$	Time (seconds)
$\Diamond(\pi_1 \wedge \pi_2) \wedge \Diamond(\pi_3 \wedge \pi_4)$	2	278.55	6.30
$\Diamond(\pi_1 \wedge \bigcirc\Diamond(\pi_2 \wedge \bigcirc\Diamond(\pi_3)))$	3	6457.9	372.37
$\neg(\pi_1 \vee \pi_2 \vee \pi_3 \vee \pi_4)\mathscr{U}(\pi_1 \wedge \pi_2 \wedge \pi_3 \wedge \pi_4)$	4	270.4	7.48

3.2. In average, the solution is found in 1057.91 seconds and require the exploration of 7242.43 vertices. This comparison shows that Alg. 2 reduces the exploration or $\mathbb{G}$ and, as a consequence, the required time to find a solution. A direct comparison with other sampling-based methods for multiple robots, e.g., [7], is not possible because our method samples the continuous configuration space instead of a discrete representation of the robots mobility.

5 Conclusions

In this paper, we have introduced a new method to find collision-free paths for a multi-robot system that satisfy syntactically co-safe linear temporal logic formulae. Most of the work in the literature consider methods with low scalability with respect to the dimension of the robot's configuration space. We extend sampling-based methods, previously proposed to alleviate the scalability problem, to multi-robot systems. The pro-

posed method explores a composite roadmap modelling the possible behaviour of all the robots. This exploration stops when a path satisfying the specification is found. Additionally, we have presented a new algorithm that guides the exploration to reduce the time required to find a solution. Numerical results show that only a small portion of the composite roadmap is explored as a result of using this algorithm. The proposed method is focused on obtaining a result in the shortest possible period of time regardless of its optimality. Hence, a possible direction for future work is the inclusion of a cost function to find optimal paths.

References

1. Alur, R., Henzinger, T.A., Lafferriere, G., Pappas, G.J.: Discrete abstractions of hybrid systems. Proc. of the IEEE 88(7), 971–984 (2000)
2. Chen, Y., Ding, X.C., Belta, C.: Synthesis of distributed control and communication schemes from global LTL specifications. In: Proc. of CDC-ECC. pp. 2718–2723. IEEE (2011)
3. van Den Berg, J., Snoeyink, J., Lin, M.C., Manocha, D.: Centralized path planning for multiple robots: Optimal decoupling into sequential plans. In: Robotics: Science and systems. vol. 2, pp. 2–3 (2009)
4. Dobson, A., Bekris, K.E.: Sparse roadmap spanners for asymptotically near-optimal motion planning. The International Journal of Robotics Research 33(1), 18–47 (2014)
5. Fainekos, G.E., Girard, A., Kress-Gazit, H., Pappas, G.J.: Temporal logic motion planning for dynamic robots. Automatica 45(2), 343–352 (2009)
6. Guo, M., Dimarogonas, D.V.: Multi-agent plan reconfiguration under local LTL specifications. The International Journal of Robotics Research 34(2), 218–235 (2015)
7. Kantaros, Y., Zavlanos, M.M.: Sampling-based control synthesis for multi-robot systems under global temporal specifications. In: Proc. of ICCPS. pp. 3–13. ACM (2017)
8. Karaman, S., Frazzoli, E.: Sampling-based motion planning with deterministic μ-calculus specifications. In: Proc. of CDC/CCC. pp. 2222–2229. IEEE (2009)
9. Kavraki, L.E., Svestka, P., Latombe, J.C., Overmars, M.H.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation 12(4), 566–580 (1996)
10. Kloetzer, M., Belta, C.: Automatic deployment of distributed teams of robots from temporal logic motion specifications. IEEE Transactions on Robotics 26(1), 48–61 (2010)
11. Kupferman, O., Vardi, M.Y.: Model checking of safety properties. Formal Methods in System Design 19(3), 291–314 (2001)
12. Leahy, K., Jones, A., Schwager, M., Belta, C.: Distributed information gathering policies under temporal logic constraints. In: Proc. of CDC. pp. 6803–6808. IEEE (2015)
13. Magnenat, S., Waibel, M., Beyeler, A.: Enki: The fast 2d robot simulator. URL http://home.gna. org/enki (2011)
14. Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J.C., Floreano, D., Martinoli, A.: The e-puck, a robot designed for education in engineering. In: Proc. of ICARSC. pp. 59–65. IPCB (2009)
15. Montana, F.J., Liu, J., Dodd, T.J.: Sampling-based stochastic optimal control with metric interval temporal logic specifications. In: Proc. of CCA. pp. 767–773. IEEE (2016)
16. Solovey, K., Salzman, O., Halperin, D.: Finding a needle in an exponential haystack: Discrete RRT for exploration of implicit roadmaps in multi-robot motion planning. The International Journal of Robotics Research 35(5), 501–513 (2016)
17. Švestka, P., Overmars, M.H.: Coordinated path planning for multiple robots. Robotics and autonomous systems 23(3), 125–152 (1998)

18. Svoreňová, M., Křetínský, J., Chmelík, M., Chatterjee, K., Černá, I., Belta, C.: Temporal logic control for stochastic linear systems using abstraction refinement of probabilistic games. In: Proc. of HSCC. pp. 259–268. ACM (2015)
19. Tumová, J., Dimarogonas, D.V.: A receding horizon approach to multi-agent planning from local LTL specifications. In: Proc. of ACC. pp. 1775–1780. IEEE (2014)
20. Ulusoy, A., Smith, S.L., Ding, X.C., Belta, C., Rus, D.: Optimality and robustness in multi-robot path planning with temporal logic constraints. The International Journal of Robotics Research 32(8), 889–911 (2013)
21. Vasile, C.I., Belta, C.: Sampling-based temporal logic path planning. In: Proc. of IROS. pp. 4817–4822. IEEE (2013)
22. Wolff, E.M., Topcu, U., Murray, R.M.: Optimal control of non-deterministic systems for a computationally efficient fragment of temporal logic. In: Proc. of CDC. pp. 3197–3204. IEEE (2013)
23. Zhang, Z., Cowlagi, R.V.: Motion-planning with global temporal logic specifications for multiple nonholonomic robotic vehicles. In: Proc. of ACC. pp. 7098–7103. IEEE (2016)