The International Journal of Robotics Research
32(8) 889–911

DOI: 10.1177/0278364913487931
ijr.sagepub.com

Optimality and Robustness in Multi-Robot Path Planning with Temporal Logic Constraints

Alphan Ulusoy[1], Stephen L. Smith[2], Xu Chu Ding[3], Calin Belta[1], and Daniela Rus[4]

Abstract

In this paper we present a method for automatic planning of optimal paths for a group of robots that satisfy a common high-level mission specification. The motion of each robot is modeled as a weighted transition system, and the mission is given as a linear temporal logic (LTL) formula over a set of propositions satisfied at the regions of the environment. In addition, an optimizing proposition must repeatedly be satisfied. The goal is to minimize a cost function that captures the maximum time between successive satisfactions of the optimizing proposition while guaranteeing that the formula is satisfied. When the robots can follow a given trajectory exactly, our method computes a set of optimal satisfying paths that minimize the cost function and satisfy the LTL formula. However, if the traveling times of the robots are uncertain, then the robots may not be able to follow a given trajectory exactly, possibly violating the LTL formula during deployment. We handle such cases by leveraging the communication capabilities of the robots to guarantee correctness during deployment and provide bounds on the deviation from the optimal values. We implement and experimentally evaluate our method for various persistent surveillance tasks in a road network environment.

Keywords

optimal path planning, optimal multi-robot path planning, temporal logic, formal methods

1. Introduction

In the classical *reach–avoid* robotic path planning problem (Choset et al., 2005; LaValle, 2006), the aim is to steer a robot from a given initial position to some final position while avoiding any obstacles along the way. Many methods based on the configuration space approach (Lozano-Perez, 1983) have been proposed to find such collision-free paths. If the dimension of the configuration space permits, one can use discretized approaches that utilize various graph search algorithms (Choset et al., 2005; LaValle, 2006) or continuous methods (Rimon and Koditschek, 1992) to solve this problem. Alternatively, randomized sampling-based algorithms such as probabilistic road map (PRM) (Kavraki et al., 1996) or rapidly-exploring random trees (RRT) (Kuffner and LaValle, 2000) can be used to find admissible paths. However, due to the limited scope of the problem that they address, classical path planning algorithms cannot handle more complex temporal and logic mission requirements.

Complex robotic missions need a precise as well as user-friendly language for requirement specification. In this regard, linear temporal logic (LTL) provides a very attractive formalism that can capture the infinite behavior of a dynamic system in an intuitive but mathematically precise manner (Baier and Katoen, 2008). Using LTL one can easily specify complex robotic missions such as "Repeatedly visit region 1. Go to region 3 before each visit to region 1. Always avoid region 2.". Current literature on path planning and control synthesis using LTL specifications considers finite systems, which may be abstractions of their infinite counterparts (Tabuada and Pappas, 2006; Yordanov et al., 2012). Given a finite system and an LTL mission specification, paths and control strategies that satisfy the mission can be automatically computed for deterministic (Kloetzer and Belta, 2010; Kress-Gazit et al., 2011), non-deterministic (Thomas, 2002; Kress-Gazit et al., 2007; Kloetzer and Belta, 2008; Yordanov et al., 2012), and probabilistic systems (Bianco and de Alfaro, 1995; Kwiatkowska et al., 2002; Ding et al., 2011). Nevertheless, finding a path

[1]Division of Systems Engineering, Boston University, Boston, MA, USA
[2]Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada
[3]Embedded Systems and Networks, United Technologies Research Center, East Hartford, CT, USA
[4]Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA

Corresponding author:
Alphan Ulusoy, Division of Systems Engineering, Boston University, Boston, MA 02215, USA.
Email: alphan@bu.edu

that accomplishes a mission is only part of the robotic path planning problem, as there remains the question of picking a particular path from all of those paths that satisfy given specifications. In this case, one can either break the tie by making an arbitrary choice or pick the best alternative in terms of safety, speed, efficiency, or some other relevant metric.

The goal of this paper is to compute optimal paths for a group of robots subject to general LTL specifications. Our approach is motivated by persistent monitoring and pickup–delivery problems, where there is an *optimizing task* that must be repeatedly completed. We aim to compute paths that satisfy the LTL specification while minimizing the maximum time between successive completions of this *optimizing task*. Previously, we provided a method that solves this problem for a single robot (Smith et al., 2011). Then, we extended our approach to multiple robots by utilizing timed automata (Ulusoy et al., 2011), and provided improved methods that are robust to uncertainties in the speeds of robots (Ulusoy et al., 2012b,a). Moving from a single robot to multiple robots requires special care, as the model of the robotic team must capture the asynchronous motion of its members. Kloetzer and Belta (2010) proposed a method for decentralized motion of multiple robots subject to LTL specifications. Their method, however, results in sub-optimal performance as it requires the robots to travel synchronously, blocking the execution of the mission before each transition until all robots are synchronized. The vehicle routing problem (VRP) (Toth and Vigo, 2001) and its extensions to more general classes of temporal constraints (Karaman and Frazzoli, 2008a,b) also deal with finding satisfying optimal paths for a given specification. Karaman and Frazzoli (2008b) considered optimal vehicle routing with metric temporal logic specifications by converting the problem to a mixed integer linear program (MILP). However, their method does not apply to the missions where robots must repeatedly complete some task, as it does not allow for specifications of the form "always eventually". Furthermore, none of these methods are robust to timing errors that can occur during deployment, as they rely on the ability of the robots to follow generated trajectories exactly for satisfaction of the mission specification. Quottrup et al. (2004) proposed a method for synthesizing controls for a team of robots subject to a computational tree logic (CTL) formula. But, they do not consider optimizing the paths of the robots. Chen et al. (2012) proposed a method for automatic synthesis of control and communication strategies for a team of robots. However, they consider finite horizon tasks given as regular expressions as opposed to infinite horizon tasks expressed in LTL that are of our interest. Moreover, their method does not consider the costs of the generated team trajectories and thus, in general, does not provide optimal solutions. Even though Chen et al. (2011) consider LTL as the specification language for the same problem, they again do not consider optimal solutions.

The contribution of this paper is threefold. First, we provide an algorithm to capture the asynchronous motion of a group of robots. Given a team of robots modeled as weighted transition systems, this algorithm constructs a new transition system that models the joint behavior of all members as a whole. Second, we provide an algorithm to compute communication strategies for a team of robots so that we can still guarantee correctness even if the robots cannot follow generated trajectories exactly during deployment. Finally, building on these two algorithms, we present a method for generating optimal paths for a group of robots satisfying general LTL formulas. Our method is general enough to address problems involving robotic teams with different capabilities. The first case that we consider is when the members of the robotic team can follow generated paths arbitrarily closely and their models have exact timing information. One such example would be a team of robots that have accurate position information and can regulate their speeds to track moving set-points that correspond to generated paths. We address such problems with our *exact* solution that generates optimal satisfying paths. However, there might also be cases where the robots lack accurate speed control and traveling times between the regions of the environment is an unknown quantity within a given interval. If this is the case, one cannot generally guarantee satisfaction of the LTL formula without additional measures. Intuitively, if during deployment the robot speeds differ from those used for planning, then the order of events can switch, which may result in the violation of the global mission specification. For such cases we propose a *robust* solution that leverages the communication capabilities of the robots to guarantee correctness and to maintain field performance in the presence of timing errors. Paths generated using this approach are robust to uncertainties in the speeds (traveling times) of robots. In addition, we characterize the performance of the robust paths with respect to the exact solutions. Preliminary versions of parts of our approach appeared in conference proceedings (Ulusoy et al., 2012b, 2011, 2012a). Here, we extend these preliminary works by presenting a unified approach that can handle cases with both exact and non-deterministic traveling times. We also provide full proofs, new case studies, and experiments.

The organization of the paper is as follows. In Section 2, we give some preliminaries in formal methods and trace-closed languages. In Section 3, we formally state the optimal motion planning problem for a team of robots and give an overview of our approach. In Section 4, we present the parts of our approach that are common to the two cases that we consider in this paper. We present our *exact* solution in Section 5, which applies to the cases where the models of the robots have exact timing information and the robots can follow generated trajectories exactly. In Section 6, we present our *robust* solution, which applies to the cases where the traveling times of the robots are uncertain

and the robots communicate to guarantee correctness during deployment and maintain field performance. In Section 7, we present experimental case studies for a team of robots performing persistent data gathering missions in a road network environment followed by numerical case studies that investigate the scalability of our approach considering a small academic example. We conclude with final remarks in Section 8.

2. Preliminaries

In this section, we introduce the notation that we use in the rest of the paper and give some definitions. We refer the reader to Clarke et al. (1999), Hopcroft et al. (2007), Baier and Katoen (2008) and references therein for a more complete and rigorous treatment of these topics.

For a set Π, we use $|\Pi|$ and 2^{Π} to denote its cardinality and power set, respectively.

Definition 2.1 (Transition system). *A (weighted) transition system (TS) is a tuple* $\mathbf{T} := (\mathcal{Q}_{\mathrm{T}}, q^0_{\mathrm{T}}, \delta_{\mathrm{T}}, \Pi_{\mathrm{T}}, \mathcal{L}_{\mathrm{T}}, w_{\mathrm{T}})$, *where:*

(i) $\mathcal{Q}_{\mathrm{T}}$ *is a finite set of states;*

(ii) $q^0_{\mathrm{T}} \in \mathcal{Q}_{\mathrm{T}}$ *is the initial state;*

(iii) $\delta_{\mathrm{T}} \subseteq \mathcal{Q}_{\mathrm{T}} \times \mathcal{Q}_{\mathrm{T}}$ *is the transition relation;*

(iv) Π_{T} *is a finite set of atomic propositions;*

(v) $\mathcal{L}_{\mathrm{T}} : \mathcal{Q}_{\mathrm{T}} \to 2^{\Pi_{\mathrm{T}}}$ *is a map giving the set of atomic propositions satisfied in a state;*

(vi) $w_{\mathrm{T}} : \delta_T \to \mathbb{N}_{>0}$ *is a map that assigns a positive integer weight to each transition.*

We define a *run* of $\mathbf{T}$ as an infinite sequence of states $r_{\mathrm{T}} = q^0, q^1, \ldots$ such that $q^0 = q^0_{\mathrm{T}}$, $q^k \in \mathcal{Q}_{\mathrm{T}}$ and $(q^k, q^{k+1}) \in \delta_{\mathrm{T}}$ for all $k \geq 0$. A run generates an infinite *word* $\omega_{\mathrm{T}} = \mathcal{L}_{\mathrm{T}}(q^0), \mathcal{L}_{\mathrm{T}}(q^1), \ldots$ where $\mathcal{L}_{\mathrm{T}}(q^k)$ is the set of atomic propositions satisfied at state q^k. A *prefix* of a run is a finite path from an initial state to a state q. A periodic *suffix* is an infinite run originating at the state q reached by the prefix, and periodically repeating a finite path, which we call the *suffix cycle*, originating and ending at q. A run is in *prefix–suffix form* if it consists of a prefix followed by a periodic suffix.

Definition 2.2 (LTL formula). *An LTL formula* ϕ *over a set of atomic propositions* Π *is defined inductively as follows (Clarke et al., 1999; Baier and Katoen, 2008):*

$$\phi := \top \mid \mathtt{p} \mid \phi \vee \phi \mid \phi \wedge \phi \mid \neg \phi \mid \mathbf{X}\phi \mid \phi \,\mathcal{U}\, \phi$$

where $\top$ *is a predicate true in each state of a system,* $\mathtt{p} \in \Pi$ *is an atomic proposition,* $\neg$ *(negation),* $\vee$ *(disjunction) and* $\wedge$ *(conjunction) are standard Boolean connectives, and* $\boldsymbol{X}$ *and* $\mathcal{U}$ *are temporal operators.*

LTL formulas are interpreted over infinite words (generated by the TS $\mathbf{T}$ from Definition 2.1 with $\Pi_{\mathrm{T}} = \Pi$). Informally, $\mathbf{X}\,\mathtt{p}$ states that at the next position of a word, proposition $\mathtt{p}$ is true. Formula $\mathtt{p}_1 \,\mathcal{U}\, \mathtt{p}_2$ states that there is a future position of the word when proposition $\mathtt{p}_2$ is true, and proposition $\mathtt{p}_1$ is true at least until $\mathtt{p}_2$ is true. From these temporal operators we can construct two other temporal operators: Eventually (future), $\mathbf{F}$, defined as $\mathbf{F}\,\phi := \top \,\mathcal{U}\, \phi$, and Always (globally), $\mathbf{G}$, defined as $\mathbf{G}\,\phi := \neg\, \mathbf{F} \neg \phi$. Formula $\mathbf{G}\,\phi$ states that ϕ is true at all positions of the word; formula $\mathbf{F}\,\phi$ states that ϕ eventually becomes true in the word. More expressivity can be achieved by combining the temporal and Boolean operators. We say a run r_{T} satisfies ϕ if and only if the word generated by r_{T} satisfies ϕ. An LTL formula ϕ over a set Π can be represented by a *Büchi automaton*, which is defined next.

Definition 2.3 (Büchi Automaton). *A Büchi automaton is a tuple* $\boldsymbol{B} := (\mathcal{Q}_{\mathrm{B}}, \mathcal{Q}^0_{\mathrm{B}}, \Pi_{\mathrm{B}}, \delta_{\mathrm{B}}, \mathcal{F}_{\mathrm{B}})$, *where:*

(i) $\mathcal{Q}_{\mathrm{B}}$ *is a finite set of states;*

(ii) $\mathcal{Q}^0_{\mathrm{B}} \subseteq \mathcal{Q}_{\mathrm{B}}$ *is the set of initial states;*

(iii) Π_{B} *is the input alphabet;*

(iv) $\delta_{\mathrm{B}} \subseteq \mathcal{Q}_{\mathrm{B}} \times \Pi_{\mathrm{B}} \times \mathcal{Q}_{\mathrm{B}}$ *is a non-deterministic transition relation;*

(v) $\mathcal{F}_{\mathrm{B}} \subseteq \mathcal{Q}_{\mathrm{B}}$ *is the set of accepting (final) states.*

A *run* of $\mathbf{B}$ over an input word $\omega = \omega^0, \omega^1, \ldots$ is a sequence $r_{\mathrm{B}} = q^0, q^1, \ldots$, such that $q^0 \in \mathcal{Q}^0_{\mathrm{B}}$, and $(q^k, \omega^k, q^{k+1}) \in \delta_{\mathrm{B}}$, for all $k \geq 0$. A Büchi automaton $\mathbf{B}$ accepts a word over Π_{B} if and only if at least one of the corresponding runs intersects with $\mathcal{F}_{\mathrm{B}}$ infinitely many times. For any LTL formula ϕ over a set Π, one can construct a Büchi automaton with input alphabet $\Pi_{\mathrm{B}} = 2^{\Pi}$ accepting all and only words over 2^{Π} that satisfy ϕ. The set of all of the words accepted by a Büchi automaton $\mathbf{B}$ is called the *language* recognized by the automaton and is denoted by L_{B}.

Given a set Π, the collection of subsets $\Pi_i \subseteq \Pi$, $\forall\, i = 1, \ldots, m$ is called a *distribution* of Π if $\cup_{i=1}^{m} \Pi_i = \Pi$. For a word ω over 2^{Π} generated by m TSs $\{\mathbf{T}_1, \ldots, \mathbf{T}_m\}$ with $\cup_{i=1}^{m} \Pi_i = \Pi$, $\omega{\restriction_i}$ denotes the *projection* of ω onto $\mathbf{T}_i$, which is the portion of ω generated by $\mathbf{T}_i$ over 2^{Π_i}.

Definition 2.4 (Trace-closed language). *Given m TSs* $\{\mathbf{T}_1, \ldots, \mathbf{T}_m\}$ *such that* $\{\Pi_1, \ldots, \Pi_m\}$ *is a distribution of* Π *and words* $\omega, \omega' \in 2^{\Pi}$, ω' *is trace-equivalent to* ω, *denoted* $\omega' \sim \omega$, *iff their projections onto each one of the TSs are equal, i.e.* $\omega \restriction_i = \omega' \restriction_i$ *for each* $i = 1, \ldots, m$. *For* $\{\mathbf{T}_1, \ldots, \mathbf{T}_m\}$, *the trace-equivalence class of* ω *is given by* $[\omega] = \{\omega' : \omega' \in 2^{\Pi}, \omega' \restriction_i = \omega \restriction_i \ \forall\ i = 1, \ldots, m\}$. *Finally, a trace-closed language is a language L such that* $[\omega] \subseteq L$, $\forall\, \omega \in L$.

Remark 2.5 (Optimal-Run Algorithm (Smith et al., 2011)). *The approach that we present in this paper utilizes the* OPTIMAL-RUN *algorithm that we developed previously in Smith et al. (2011). The algorithm takes as input a weighted TS modeling the motion of a robot and an LTL*

formula of the form $\phi := \varphi \wedge \boldsymbol{GF}\pi$. In formula ϕ, π is the optimizing task *that must be repeatedly satisfied and φ is an arbitrary LTL formula for capturing other mission requirements. The* OPTIMAL-RUN *algorithm outputs an optimal satisfying run that satisfies ϕ and minimizes the maximum time between successive satisfying instances of π. We refer the interested reader to Smith et al. (2011) for more details on the* OPTIMAL-RUN *algorithm.*

3. Problem formulation and approach

In this section we introduce the optimal multi-robot path planning problem and motivate the need for solutions that are robust to uncertain robot speeds. Let

$$\mathcal{E} = (V, \rightarrow_{\mathcal{E}}, \Pi, \mathcal{L})$$

be a directed graph, where V is the set of vertices, $\rightarrow_{\mathcal{E}} \subseteq V \times V$ is the set of edges, Π is a finite set of atomic propositions, and $\boldsymbol{\mathcal{L}}$ is a map giving the set of atomic propositions satisfied at a vertex. In this paper, $\mathcal{E}$ is the quotient graph of a partitioned environment, where V is a set of labels for the regions in the partition and $\rightarrow_{\mathcal{E}}$ is the corresponding adjacency relation. For example, V can be a set of labels for the regions and intersections for a road network and $\rightarrow_{\mathcal{E}}$ can give their connections (see Figure 4).

Consider a team of m robots moving in an environment modeled by $\mathcal{E}$. The motion capabilities of robot $i \in \{1, \ldots, m\}$ are represented by a TS $\mathbf{T}_i = (\mathcal{Q}_i, q_i^0, \delta_i, \Pi_i, \mathcal{L}_i, w_i)$, where $\mathcal{Q}_i \subseteq V$; q_i^0 is the initial vertex of robot i; $\delta_i \subseteq \rightarrow_{\mathcal{E}}$ is a relation modeling the capability of robot i to move among the vertices; $\Pi_i \subseteq \Pi$ is the set of propositions that can be satisfied by robot i and $\{\Pi_1, \ldots, \Pi_m\}$ is a distribution of Π; $\mathcal{L}_i$ is a mapping from $\mathcal{Q}_i$ to 2^{Π_i} showing how the propositions are satisfied at vertices; $w_i(q, q')$ captures the time for robot i to go from vertex q to q', which we assume to be a positive integer. In this model, each robot travels along the edges of the corresponding TS $\mathbf{T}_i$, and spends zero time at its vertices. We assume that the robots are equipped with motion primitives that allow them to deterministically move from q to q' for each $(q, q') \in \delta_i$.

We consider the case where this robotic team has a mission in which some particular task must be repeatedly completed and the maximum time in between successive completions of this task must be minimized. For instance, in a persistent surveillance mission (Smith et al., 2011), the global mission could be to *keep gathering data while obeying traffic rules at all times*, and the repeating task could be *gathering data*. For this example, the robots would operate according to the mission specification while ensuring that the maximum time between successive data gatherings is minimized. Consequently, we assume that there is an optimizing proposition $\pi \in \Pi$ corresponding to this particular repeating task and consider missions specified by LTL formulae of the form

$$\phi := \varphi \wedge \mathbf{GF}\pi, \tag{1}$$

where φ can be any LTL formula over Π, and $\mathbf{GF}\pi$ means that the proposition π must be repeatedly satisfied. Our aim is to plan multi-robot paths that satisfy the mission specified by ϕ and minimize the maximum time between successive satisfying instances of π.

To state this problem formally, we assume that each run $r_i = q_i^0, q_i^1, \ldots$ of $\mathbf{T}_i$ (robot i) starts at $t = 0$ and generates a word $\omega_i = \omega_i^0, \omega_i^1, \ldots$ and a corresponding sequence of time instances $\mathbb{T}_i := t_i^0, t_i^1, \ldots$ such that $\omega_i^k = \mathcal{L}_i(q_i^k)$ is satisfied at t_i^k. To define the behavior of the team as a whole, we interpret the sequences $\mathbb{T}_i$ as sets, take the union $\bigcup_{i=1}^{m} \mathbb{T}_i$ and order this set in an ascending order to obtain the sequence $\mathbb{T} := t^0, t^1, \ldots$. Next, we define $\omega_{team} = \omega_{team}^0, \omega_{team}^1, \ldots$ to be the word generated by the team of robots where ω_{team}^k is the union of all propositions satisfied at t^k. Then, we define the infinite sequence $\mathbb{T}^\pi = \mathbb{T}^\pi(1), \mathbb{T}^\pi(2), \ldots$ where $\mathbb{T}^\pi(k)$ stands for the time instance when π is satisfied for the kth time by the team.[1] Finally, we define the cost function

$$J(\mathbb{T}^\pi) = \limsup_{k \to +\infty} (\mathbb{T}^\pi(k+1) - \mathbb{T}^\pi(k)). \tag{2}$$

The form of the cost function given in (2) is motivated by persistent surveillance and pickup–delivery missions, where one is interested in the long-term behavior of the team. Given a sequence $\mathbb{T}^\pi$ corresponding to a run of the team, the cost function in (2) captures the maximum time between satisfying instances of π once the team behavior reaches a steady-state, which is achieved in finite time as we will discuss in Section 4.2.

In this paper we are particularly interested in the implementability and robustness of our solutions. Thus, we consider two cases for the traveling times given by the models of the robots: the first case that we consider is when the weight $w_i(q, q')$ of each transition $(q, q') \in \delta_i$ is exactly the time it takes for robot i to go from q to q' for $i = 1, \ldots, m$. This corresponds to the case when the robots can follow any given run exactly when deployed in the environment and $\mathbb{T}^\pi$ observed during deployment is identical to the planned $\mathbb{T}^\pi$. The second case that we consider is when the robots lack accurate speed control and the actual time it takes for robot i to go from q to q' is an uncertain quantity $\tilde{w}_i(q, q')$ taking values in known intervals non-deterministically. The interval of each $\tilde{w}_i(q, q')$ is given by $[\underline{\rho_i} w_i(q, q'), \overline{\rho_i} w_i(q, q')]$, where $w_i(q, q')$ is the weight of the transition $(q, q') \in \delta_i$, $\underline{\rho_i}$ and $\overline{\rho_i}$ are the lower and upper *deviation values* of robot i, and $0 < \underline{\rho_i} \leq 1 \leq \overline{\rho_i}$. In this setting, we treat the weight $w_i(q, q')$ given by $\mathbf{T}_i$ as a nominal value, which determines the bounds of the uncertain traveling time $\tilde{w}_i(q, q')$ along with $\underline{\rho_i}$ and $\overline{\rho_i}$. We further assume that $\underline{\rho_i}$ and $\overline{\rho_i}$ of each robot i are known *a priori*. In the following, we use x and $\tilde{x}$ to denote the *nominal* and *actual* values of some variable x, and use the expression "*in the field*" to refer to the model with uncertain traveling times. Note that, for the case of uncertain traveling times, $J(\mathbb{T}^\pi)$ corresponds to the *nominal* value of the cost function, whereas $J(\tilde{\mathbb{T}}^\pi)$ is the *actual*

maximum time between any two successive satisfactions of π during deployment, i.e.

$$J(\tilde{\mathbb{T}}^{\pi}) = \limsup_{k \to +\infty} \left(\tilde{\mathbb{T}}^{\pi}(k+1) - \tilde{\mathbb{T}}^{\pi}(k) \right).$$

When the robots cannot follow generated trajectories exactly, the order in which the propositions are satisfied may switch during deployment. Then, the *actual* word $\tilde{\omega}_{team}$ generated by the robotic team during its infinite asynchronous run in the field may not be the *planned* word ω_{team}, but a trace equivalent of ω_{team} instead, i.e. $\tilde{\omega}_{team} \in [\omega_{team}]$. This leads to the definition of critical words.

Definition 3.1 (Critical words). *Given the language L_{B} of the Büchi automaton that corresponds to the LTL formula ϕ over Π, and a team of m robots modeled as TSs $\{\mathbf{T}_1, \ldots, \mathbf{T}_m\}$ such that $\{\Pi_1, \ldots, \Pi_m\}$ is a distribution of Π, the word ω_{team} over 2^{Π} is a* critical word *if $\exists\, \tilde{\omega}_{team} \in [\omega_{team}]$ such that $\tilde{\omega}_{team} \notin L_{\mathrm{B}}$, where $[\omega_{team}]$ is the trace-equivalence class of ω_{team} (Definition 2.4).*

Thus, we see that if the planned word is critical and the traveling times of the robots are non-deterministic, then we may not satisfy the specification in the field. This can be formalized by noting that the optimal runs that satisfy (1) are always in a prefix–suffix form (Smith et al., 2011), where the suffix cycle is repeated infinitely often. Using this observation and Definition 3.1 we can formally define the words that can violate the LTL formula during the deployment of a robotic team with uncertain traveling times.

Proposition 3.2. *If the suffix cycle of the word ω_{team} is a critical word and the traveling times of the robots are non-deterministic, then the correctness of the motion of the robotic team during its deployment cannot be guaranteed.*

Proof. We denote the actual word generated by the robotic team in the field by $\tilde{\omega}_{team}$, whereas ω_{team} stands for the planned word. Suppose that for each robot $\underline{\rho_i} = 1 - \epsilon$, $\overline{\rho_i} = 1 + \epsilon$, and in the suffix cycle of ω_{team} we have $\alpha \subseteq \omega_{team}^{k}$ and $\beta \subseteq \omega_{team}^{(k+\tau)}$, where α and β are the propositions generated by robots i and j at positions k and $k+\tau$ of ω_{team}, respectively. Further assume that β must not occur before α, because if it does, $\tilde{\omega}_{team}$ violates ϕ. Note that we are guaranteed to find such α and β as we assume the suffix cycle to be a critical word. In the worst case, for $\tilde{\omega}_{team}$ to violate ϕ, we must have $(1+\epsilon)\, t^{k} > (1-\epsilon)\, t^{k+\tau}$, where t^{k} is the time at which ω_{team}^{k} is satisfied. Solving for ϵ, we get $\epsilon > (t^{k+\tau} - t^{k}) / (t^{k} + t^{k+\tau})$. However, as the suffix is an infinite repetition of the suffix cycle, $\lim_{k \to \infty} (t^{k+\tau} - t^{k}) / (t^{k} + t^{k+\tau}) = 0$ and ϕ is violated for any $\epsilon > 0$. ■

Remark 3.3 (Worst-case performance in the field under uncertain traveling times). *In addition, we can consider the performance of the team during deployment in terms of the value of the cost function* (2) *observed in the field. Using the same arguments presented in Proposition 3.2, it can be easily shown that the worst-case field value of* (2) *will be the minimum of $(J(\tilde{\mathbb{T}}_1^{\pi}), \ldots, J(\tilde{\mathbb{T}}_m^{\pi}))$, where $\tilde{\mathbb{T}}_i^{\pi}$ is the time sequence of satisfactions of π by robot i and $J(\tilde{\mathbb{T}}_i^{\pi})$ is the maximum duration between any two successive satisfactions of π by robot i in the field. This effectively means that, in the worst case, there is no benefit in executing the task with multiple robots, as at some point in the future the overall performance of the team will be limited by that of a single member.*

Proposition 3.2 shows that we cannot solely rely on the planned runs to satisfy the mission when the traveling times are uncertain and the suffix cycle of the word ω_{team} is a critical word. Thus, for such cases, it is relevant to consider the communication capabilities of the robots as one may leverage them to guarantee correctness during deployment. We can now formulate the problem that we consider in this paper.

Problem 3.4. *Given an LTL formula ϕ over Π of the form* (1) *and a team of m robots modeled as TSs $\{\mathbf{T}_1, \ldots, \mathbf{T}_m\}$, possibly with uncertain traveling times characterized by deviation values $\overline{\rho_i}, \underline{\rho_i}, i = 1, \ldots, m$; generate individual runs and communication strategies for each robot such that $\mathbb{T}^{\pi}$ minimizes the cost function* (2) *subject to the constraint that ω_{team}, or $\tilde{\omega}_{team}$ in case of uncertain traveling times, satisfies ϕ.*

Since we consider LTL formulas containing $\mathbf{GF}\pi$, this optimization problem is always well-posed. An overview of our approach is given in Figure 1. Note that the exact steps we take to solve Problem 3.4 depend on whether the traveling times of the robots are uncertain or not. Nevertheless, in both solutions, we first construct the team TS $\mathbf{T}$ that captures the joint asynchronous motion of the robots in the environment (Section 4.1). Then, we find an optimal satisfying run on $\mathbf{T}$ using the OPTIMAL-RUN algorithm we previously developed in Smith et al. (2011), and project this run back to the individual $\mathbf{T}_i$, $i = 1, \ldots, m$ (Section 4.2). In the next section, we discuss these common parts of our approach before presenting our *exact* and *robust* solutions in the sections that follow.

Remark 3.5 (Complexity of multi-robot optimal path planning). *LTL model checking is the problem of automatically checking a given system model against some LTL specification ψ. Sistla and Clarke (1985) showed that the complexity of LTL model checking is PSPACE-complete. The single-robot version of Problem 3.4, where the aim is to find an optimal path that satisfies a given LTL specification of the form* (1) *and minimizes* (2)*, was previously considered by Smith et al. (2011). Note that any instance of the LTL model checking problem can be transformed to a single-robot optimal path planning problem in polynomial time by letting $\phi := \neg\psi \wedge \mathbf{GF}\pi$ and defining π on all states of the model. Then, if one can find an optimal path that satisfies ϕ, the system model violates ψ, and vice versa. Thus, the single-robot version of Problem 3.4 is PSPACE-hard. Since the multi-robot optimal path planning problem is at*

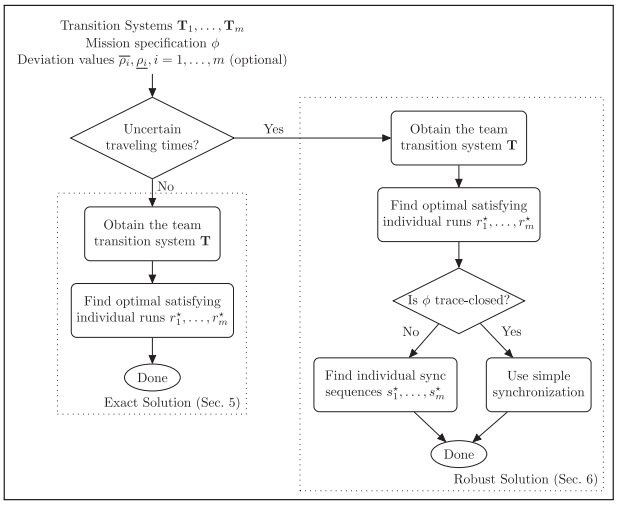

Fig. 1. An overview of our approach.

least as hard as the single-robot case, Problem 3.4 is also PSPACE-hard.

Remark 3.6 (Optimization objective). *Another interesting optimization objective would be to compute robot paths that give the best performance for the worst case, i.e.* $\min \max J(\mathbb{T}^\pi)$*, where minimization is over all paths that satisfy* ϕ*, and maximization is over all possible realizations of traveling times within the given intervals. However, it appears that this would entail the solution of an additional optimization problem over a high-dimensional continuous space (for discovering the worst-case traveling times), potentially resulting in a further increase in the complexity of this problem.*

4. Modeling the team and finding optimal satisfying runs

As given in Figure 1, there are two operations common to both of our solutions: construction of the team TS $\mathbf{T}$ and finding optimal satisfying runs for individual robots. In the following, we discuss these operations.

4.1. Constructing the team transition system

In order to be able to optimize the motion of the team, we must capture the joint asynchronous behavior of its members as they move in the environment. Since traveling times between regions are typically not identical, we need a way to capture the states, or relative positions, of the robots regardless of whether they are at the regions in the environment or traveling between the regions. This leads to the definition of traveling states.

Definition 4.1 (Traveling state). *Given the TS* $\mathbf{T}_i := (\mathcal{Q}_i, q_i^0, \delta_i, \Pi_i, \mathcal{L}_i, w_i)$ *modeling robot i, we refer to a state of the form* $q_i q_i' x_i$*, where* $q_i, q_i' \in \mathcal{Q}_i$ *and* $x_i > 0$*, as a traveling state, and use it to represent the instant where robot i has traveled from* q_i *to* q_i' *for* x_i *time units.*

Algorithm 1: CONSTRUCT-TEAM-TS

Input: $\{\mathbf{T}_1, \ldots, \mathbf{T}_m\}$.
Output: Corresponding team transition system $\mathbf{T}$.

1 $q_\mathrm{T}^0 := (q_1^0, \ldots, q_m^0)$.
2 **dfsT**(q_T^0).

Function dfsT(state tuple $q \in \mathcal{Q}_\mathrm{T}$)

4 Define $q[i]$ as the ith element of q.
5 Define $\rightarrow_i$ as a transition of $\mathbf{T}_i$, such that $\rightarrow_i \in \{(q[i], q_i') \mid (q[i], q_i') \in \delta_i\}$ for $q[i] \in \mathcal{Q}_i$ and $\rightarrow_i = (q_i, q_i')$ for $q[i] = q_i q_i' x_i$.
6 $\mathcal{T}$ is the set of all possible transition tuples $(\rightarrow_1, \ldots, \rightarrow_m)$ at q.
7 **foreach** *transition tuple* $(\rightarrow_1, \ldots, \rightarrow_m) \in \mathcal{T}$ **do**
8 $w =$ Shortest time until a robot is at a vertex.
9 Find the q' that corresponds to the new state of the team.
10 **if** $q' \notin \mathcal{Q}_\mathrm{T}$ **then**
11 Add state q' to $\mathcal{Q}_\mathrm{T}$.
12 Set $\mathcal{L}_\mathrm{T}(q') = \cup_{i=1}^m \mathcal{L}_i(q'[i])$.
13 Add (q, q') to δ_T with weight w.
14 Continue search from q': **dfsT**(q').
15 **else if** $(q, q') \notin \delta_\mathrm{T}$ **then**
16 Add (q, q') to δ_T with weight w.

To model the asynchronous motion of the team in the environment, we use a *team* TS $\mathbf{T} = (\mathcal{Q}_\mathrm{T}, q_\mathrm{T}^0, \delta_\mathrm{T}, \Pi_\mathrm{T}, \mathcal{L}_\mathrm{T}, w_\mathrm{T})$, where $\mathcal{Q}_\mathrm{T}$ is the set of states of the form $q = (q[1], \ldots, q[m])$ where q is a tuple and its ith element $q[i]$ is the state of robot i; $q_\mathrm{T}^0 = (q_1^0, \ldots, q_m^0)$ is the initial state of the team; δ_T is the set of transitions; $\Pi_\mathrm{T} = \cup_{i=1}^m \Pi_i$ is the set of propositions; $\mathcal{L}_\mathrm{T}$ is a mapping from $\mathcal{Q}_\mathrm{T}$ to 2^{Π_T}; $w_\mathrm{T}(q, q')$ is the weight of the transition from q to q'. The states of $\mathbf{T}$ correspond to the instants where at least one member of the team has completed a transition on its individual TS and is currently at a vertex while other robots may still be traveling. When robot i is at some region in the environment, we have $q[i] \in \mathcal{Q}_i$. If, on the other hand, robot i is traveling from q_i to q_i' and it has been x_i time units since it left q_i, we have $q[i] = q_i q_i' x_i$. Using this, we construct $\mathbf{T}$ by running a depth first search on the TSs of the individual members of the team as given in Algorithm 1.

Algorithm 1 is essentially a recursive depth first search (lines 4–16) that starts at the initial state of the team TS $\mathbf{T}$ (line 2). The initial state q_T^0 of $\mathbf{T}$ is defined as the tuple of the initial states of the m TSs (line 1). Given a state q of $\mathbf{T}$, the function `dfsT` first generates all possible tuples of transitions that can be taken at the current states of the TSs $\{\mathbf{T}_1, \ldots, \mathbf{T}_m\}$ (lines 4–6). The current state of TS $\mathbf{T}_i$ is given by the ith element $q[i]$ of the current state q of $\mathbf{T}$. At line 5 of Algorithm 1, we consider all possible transitions out of the current states of all TSs $\{\mathbf{T}_1, \ldots, \mathbf{T}_m\}$. If $q[i] \in \mathcal{Q}_i$, i.e.

$q[i]$ is a regular state of $\mathbf{T}_i$, then all transitions going out of this state in $\mathbf{T}_i$ will be considered in the transition tuples that we will construct. Otherwise, $q[i]$ is a traveling state of $\mathbf{T}_i$ of the form $q_i q_i' x_i$, and the only transition that can be taken is the one that is being taken, i.e. the transition from q_i to q_i'. Then, we construct the set of all possible tuples of transitions that can be taken at the current states of the TSs (line 6) and process each tuple one by one (lines 7–16). In a transition tuple $(\rightarrow_1, \ldots, \rightarrow_m)$, the ith element $\rightarrow_i$ gives the transition that is being taken at the current state of $\mathbf{T}_i$. In lines 8 and 9, we find the next instant where at least one transition from the current tuple $(\rightarrow_1, \ldots, \rightarrow_m)$ has been completed and the next state q' of $\mathbf{T}$ has been reached. The ith element $q'[i]$ of the next state q' of $\mathbf{T}$ corresponds to the next state of $\mathbf{T}_i$ w time units after starting taking the transition $\rightarrow_i$ at $q[i]$. Suppose that, the source and target states of transition $\rightarrow_i$ are q_i and q_i', respectively. If the transition $\rightarrow_i$ has been completed at this point, then $q'[i] = q_i'$, i.e. we set the next state of $\mathbf{T}_i$ to the target state of $\rightarrow_i$. Otherwise, $q'[i]$ is a traveling state of the form $q_i q_i' x_i$ such that $x_i = w$ if $q[i] = q_i$, and $x_i = n + w$ if $q[i] = q_i q_i' n$. If q' is a new state (lines 10–14), we accordingly add it to $\mathcal{Q}_\mathrm{T}$ and define its propositions. Then, we add the transition that has just been completed to δ_T and continue our search from this new state q'. Otherwise, we add the transition that has just been completed to δ_T if required and proceed to the next transition tuple in $\mathcal{T}$. The algorithm concludes when all states and transitions of $\mathbf{T}$ have been discovered.

The following proposition provides a bound on the size of the team TS $\mathbf{T}$.

Proposition 4.2. *The number of states $|\mathcal{Q}_\mathrm{T}|$ of $\mathbf{T}$ is bounded by*

$$\prod_{i=1}^{m} |\mathcal{Q}_i| + (w_{max} - 1) \prod_{i=1}^{m} |\delta_i| \tag{3}$$

where w_{max} is the largest edge weight in all TSs $\{\mathbf{T}_1, \ldots, \mathbf{T}_m\}$.

Proof. The first term in (3) is the maximum number of states that we can have in the Cartesian product of T_i, $i = 1, \ldots, m$. The second term in (3) is an upper-bound on the number of traveling states (Definition 4.1) that we can define as we construct $\mathbf{T}$. Here, $\prod_{i=1}^{m} |\delta_i|$ is the maximum number of different transition tuples that we can consider (Algorithm 1, line 7) and $(w_{max} - 1)$ is the upper bound on the number of new traveling states per transition tuple. Thus, $|\mathcal{Q}_\mathrm{T}|$ is bounded by the sum of these two terms as given in (3). ■

Remark 4.3 (Comparison with naive construction). *One can avoid going through Algorithm 1 and capture the joint behavior of the team by discretizing each transition in $\mathbf{T}_i$, $i = 1, \ldots, m$ to unit-length edges and taking the synchronous product of these m TSs. This approach, however, yields a much larger model whose state count is bounded by*

$$\prod_{i=1}^{m} \left(|\mathcal{Q}_i| + \sum_{(q,q') \in \delta_i} w_i(q, q') - |\delta_i| \right).$$

For the case where we have m identical robots in an environment with Q vertices, Δ edges and a largest edge weight of w_{max}, the above given bound is $O((Q + \Delta w_{max})^m)$, whereas the bound given by Proposition 4.2 is $O(Q^m + \Delta^m w_{max})$.

4.2. Finding optimal satisfying runs for individual robots

Once we have the TS $\mathbf{T}$ modeling the team, we can use the OPTIMAL-RUN algorithm (Smith et al., 2011) to obtain an optimal run $r_{team}^\star$ on $\mathbf{T}$ that minimizes the cost function (2) and satisfies any mission specification ϕ of the form (1). The optimal run $r_{team}^\star$ always consists of a finite sequence of states of $\mathbf{T}$ (prefix), followed by infinite repetitions of another finite sequence of states of $\mathbf{T}$ (suffix).

Given a run r_{team} of $\mathbf{T}$, we can finally project it onto individual robots to obtain their individual runs $\{r_1, \ldots, r_m\}$.

Definition 4.4 (Projection of a run on T to $\mathbf{T}_i$). *Given a run r_{team} on $\mathbf{T}$ where $r_{team} = q^0, q^1, \ldots$, we define its projection on $\mathbf{T}_i$ as run $r_i = q_i^0 q_i^1 \ldots$ for all $i = 1, \ldots, m$, such that q_i^k appears in r_i only if $q^k[i] \in \mathcal{Q}_i$ where $q^k[i]$ is the ith element of tuple q^k.*

It can be easily seen that the set of runs $\{r_1, \ldots, r_m\}$ obtained from r_{team} using Definition 4.4 and the run r_{team} on $\mathbf{T}$ agree with each other: the projection given in Definition 4.4 simply breaks down a sequence of tuples of states into a tuple of sequences of states, while preserving the order of the states and filtering out the traveling states. Thus, the word ω and the time sequence $\mathbb{T}$ generated by $\{r_1, \ldots, r_m\}$ are exactly the word ω_{team} and the time sequence $\mathbb{T}_{team}$ generated by r_{team}. Moreover, if the run r_{team} is in prefix–suffix form, all individual runs r_i projected from r_{team} are also in prefix–suffix form. Therefore, the individual runs projected from the optimal run $r_{team}^\star$ are always in prefix–suffix form.

5. Exact solution

In this section we consider the case where the models of the robots have exact timing information and the time it takes for the robots to travel between regions during deployment is exactly the time captured in their models. Consequently, if we plan a run based on the models of the robots, the run that we will observe when the robots are deployed will be exactly the planned run in the sense that the times at which robots reach the regions in the run will be exactly as planned.

To solve Problem 3.4 in this case, we first create a model of the motion of the team in the environment. Given the

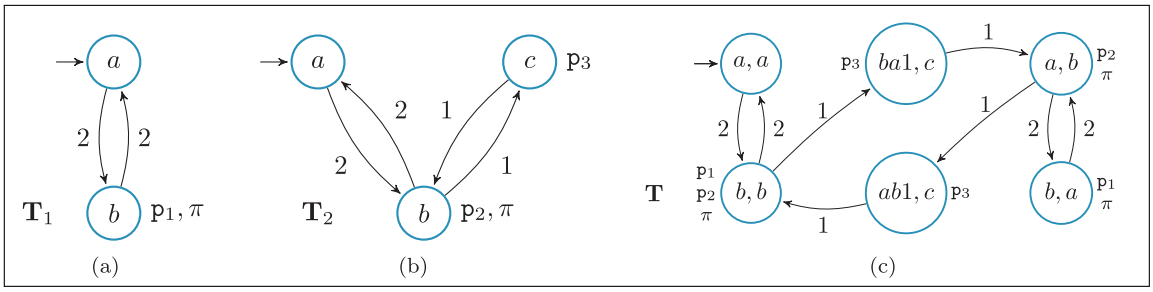

Fig. 2. (a), (b) The TSs $\mathbf{T}_1$ and $\mathbf{T}_2$ of two robots in an environment with three vertices. The states of the TSs correspond to vertices $\{a, b, c\}$ and the edges represent the motion capabilities of each robot. The weights of the edges represent the traveling times between any two vertices. Propositions $\mathtt{p}_1, \mathtt{p}_2, \mathtt{p}_3$, and π are shown next to the vertices where they can be satisfied by the robots. (c) The team TS capturing the joint behavior of the robots in 6 states. A state labeled (a, b) means robot 1 is at region a and robot 2 is at region b, whereas a state labeled $(ba1, c)$ means robot 1 has traveled from b to a for 1 time unit and robot 2 is at c.

individual TSs $\{\mathbf{T}_1, \ldots, \mathbf{T}_m\}$ of the robots, we use Algorithm 1 to construct the team TS $\mathbf{T}$ that captures the joint asynchronous behavior of the robots.

Example 5.1. *Figures 2(a) and 2(b) illustrate the TSs of two robots, where* $\Pi_1 = \{\mathtt{p}_1, \pi\}$, $\Pi_2 = \{\mathtt{p}_2, \mathtt{p}_3, \pi\}$, *and* $\Pi = \{\mathtt{p}_1, \mathtt{p}_2, \mathtt{p}_3, \pi\}$. *Using Algorithm 1 we construct the team TS* $\mathbf{T}$ *(Figure 2(c)) that captures the joint asynchronous behavior of the team in six states.*

Next, given an LTL mission specification ϕ of the form (1), we use our previous OPTIMAL-RUN algorithm (Smith et al., 2011) to generate an optimal satisfying run $r^\star_{team}$ on the team TS $\mathbf{T}$. Then, we project the optimal satisfying run $r^\star_{team}$ on $\mathbf{T}$ onto individual TSs using Definition 4.4 to obtain individual optimal satisfying runs $\{r^\star_1, \ldots, r^\star_m\}$ of the robots.

Example 5.1 Revisited. *Running the* OPTIMAL-RUN *algorithm (Smith et al., 2011) for the team TS* $\mathbf{T}$ *given in Figure 2(c), and the formula* $\phi := \mathbf{GF}\pi$ *results in the optimal run*

$\mathbb{T}$	0	2	3	4	6	8	10	...
$r^\star_{team}$	a,a	b,b	ba1,c	a,b	b,a	a,b	b,a	...
$\mathcal{L}_{\mathbf{T}}(\cdot)$		$\mathtt{p}_1, \mathtt{p}_2, \pi$	$\mathtt{p}_3$	$\mathtt{p}_2, \pi$	$\mathtt{p}_1, \pi$	$\mathtt{p}_2, \pi$	$\mathtt{p}_1, \pi$	...
$r^\star_1$	a	b		a	b	a	b	...
$r^\star_2$	a	b	c	b	a	b	a	...

where the first row corresponds to the times when transitions occur, the second row corresponds to the run $r^\star_{team}$, *the third row shows the propositions satisfied at each position, and the last two rows correspond to the individual runs of the robots. For this run, we see that* $(a, a), (b, b), (ba1, c), (a, b)$ *is the prefix and* $(a, b), (b, a)$ *is the suffix cycle and will be repeated an infinite number of times. Also, the time sequence of satisfactions of* π *is* $\mathbb{T}^\pi = 2, 4, 6, 8, 10, \ldots$ *and the cost as defined in* (2) *is* $J(\mathbb{T}^\pi) = 2$. *Note that, at time* $t = 3$, *the second robot has arrived at c while the first robot is still traveling from b to a, therefore* $r^\star_1$ *has no state corresponding to time* $t = 3$.

We finally summarize our exact solution in Algorithm 2, and show that this algorithm indeed gives a solution to Problem 3.4 for the case where the models of the robots have exact timing information. We analyze the overall complexity of Algorithm 2 in Proposition 5.3.

Algorithm 2: EXACT-MULTI-ROBOT-OPTIMAL-RUN

Input: Transition systems $\{\mathbf{T}_1, \ldots, \mathbf{T}_m\}$ and an LTL specification ϕ of the form (1).

Output: A set of runs $\{r^\star_1, \ldots, r^\star_m\}$ that both satisfies ϕ and minimizes (2).

1 Construct the team TS $\mathbf{T}$ using CONSTRUCT-TEAM-TS (Algorithm 1).
2 Find the optimal run $r^\star_{team}$ on $\mathbf{T}$ using OPTIMAL-RUN (Smith et al., 2011).
3 Project $r^\star_{team}$ onto $\{\mathbf{T}_1, \ldots, \mathbf{T}_m\}$ to obtain runs $\{r^\star_1, \ldots, r^\star_m\}$ (Definition 4.4).

Proposition 5.2. *Algorithm 2 solves Problem 3.4.*

Proof. Note that Algorithm 2 combines all steps outlined in this section. Run $r^\star_{team}$ obtained from Algorithm OPTIMAL-RUN both satisfies ϕ and minimizes (2) among all runs of $\mathbf{T}$ (Smith et al., 2011). As discussed in Section 4.2, there is a one-to-one correspondence between a set of runs $\{r_1, \ldots, r_m\}$ obtained using Definition 4.4 and a run r_{team} of $\mathbf{T}$. Therefore, $\{r^\star_1, \ldots, r^\star_m\}$ as a projection of $r^\star_{team}$ onto $\{\mathbf{T}_1, \ldots, \mathbf{T}_m\}$ is a solution to Problem 3.4. ■

Proposition 5.3. *For the case where a group of m identical robots are expected to satisfy an LTL specification* ϕ *in a common environment with Q vertices,* Δ *edges and a largest edge weight of* w_{max}, *the worst-case complexity of Algorithm 2 is* $O((Q^m + \Delta^m w_{max})^3 \cdot 2^{O(|\phi|)})$.

Proof. For the above mentioned case, the worst-case size of $\mathbf{T}$ as given in (3) is $O(Q^m + \Delta^m w_{max})$. Smith et al. (2011) gave the worst-case complexity of the OPTIMAL-RUN algorithm as $O(|T|^3 \cdot 2^{O(|\phi|)})$ where $|T|$ is the number of states of the input TS and $|\phi|$ is the length of the LTL specification. Then, the worst-case complexity of Algorithm 2 becomes $O((Q^m + \Delta^m w_{max})^3 \cdot 2^{O(|\phi|)})$. ■

6. Robust solution

In this section we consider the case where the actual traveling times of the robots observed during deployment, denoted by $\tilde{w}_i(q,q')$, are uncertain quantities taking values in known intervals non-deterministically. Recall from Section 3 that, $\tilde{w}_i(q,q')$ lies in the interval $[\underline{\rho_i} w_i(q,q'), \overline{\rho_i} w_i(q,q')]$, where $w_i(q,q')$ is the nominal value given by $\mathbf{T}_i$, $\underline{\rho_i}$ and $\overline{\rho_i}$ are the lower and upper *deviation values* of robot i, and $0 < \underline{\rho_i} \leq 1 \leq \overline{\rho_i}$. Thus, when the robots execute a planned run in the field, the run observed during deployment may be different from the one planned, possibly violating the mission specification. As discussed previously in Section 3, our solution in this case will also comprise a communication strategy so that the satisfaction of the mission specification will be guaranteed and the deviation of the field performance from optimality will be bounded.

6.1. Optimal satisfying runs and transition systems with traveling states

Given the TSs $\{\mathbf{T}_1, \ldots, \mathbf{T}_m\}$ of the robots and the mission specification ϕ, we first construct the team TS $\mathbf{T}$ using Algorithm 1 to model the team. Then, we use the OPTIMAL-RUN algorithm (Smith et al., 2011) to obtain a run $r^\star_{team}$ on $\mathbf{T}$ that satisfies ϕ and minimizes the cost function (2).

Example 6.1. *Running the* OPTIMAL-RUN *algorithm (Smith et al., 2011) on* $\mathbf{T}$ *given in Figure 2(c) for the formula* $\phi = \mathbf{G}(\mathrm{p}_1 \Rightarrow \mathbf{X}(\neg \mathrm{p}_1 \,\mathcal{U}\, \mathrm{p}_3)) \wedge \mathbf{GF}\pi$ *results in the optimal run*

$\mathbb{T}$	0	2	3	4	5	6	...
$r^\star_{team}$	a,a	b,b	ba1,c	a,b	ab1,c	b,b	...
$\mathcal{L}_{\mathbf{T}}(\cdot)$		$\mathrm{p}_1, \mathrm{p}_2, \pi$	p_3	p_2, π	p_3	$\mathrm{p}_1, \mathrm{p}_2, \pi$	...

where the first row shows when transitions occur, the second row corresponds to the run $r^\star_{team}$*, and the last row shows the satisfying atomic propositions. For this run,* $(a,a),(b,b)$ *is the finite prefix and* $(b,b),(ba1,c),(a,b),(ab1,c)$ *is the suffix cycle, which will be repeated an infinite number of times. Also, the time sequence* $\mathbb{T}^\pi$ *of satisfactions of* π *is* $\mathbb{T}^\pi = 2,4,6,8,\ldots$ *and the cost as defined in* (2) *is* $J(\mathbb{T}^\pi) = 2$.

Since $\mathbf{T}$ captures the asynchronous motion of the robots, the optimal satisfying run $r^\star_{team}$ on $\mathbf{T}$ may contain some traveling states (Definition 4.1) which do not appear in the individual TSs $\{\mathbf{T}_1, \ldots, \mathbf{T}_m\}$ that we started with. In our *exact* solution (Section 5), we pruned such states as we projected $r^\star_{team}$ onto $\{\mathbf{T}_1, \ldots, \mathbf{T}_m\}$ to obtain $\{r^\star_1, \ldots, r^\star_m\}$. But we cannot ignore such traveling states in this case, as each one of them is a candidate synchronization point for the corresponding robot as we discuss in the following subsections. Instead, we insert those traveling states into individual TSs so that the robots will be able to synchronize with each other at those points if needed. In the following, we use

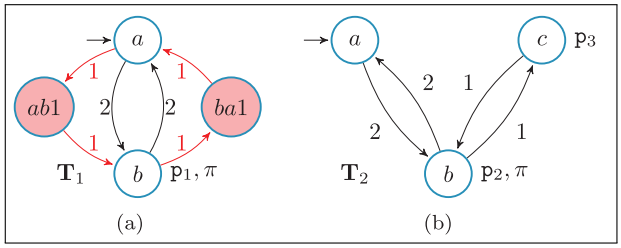

Fig. 3. (a), (b) The TSs with new traveling states and transitions that correspond to the optimal run $r^\star_{team}$ that we compute for Example 6.1. (a) The new traveling states and transitions of $\mathbf{T}_1$ are highlighted in red.

$q^k[i]$ to denote the ith element of the kth state tuple in $r^\star_{team}$, which is also the state of robot i at that position of $r^\star_{team}$. As given in Definition 4.1, a traveling state of robot i has the form $q_i q'_i x_i$. First, we construct the set $\mathcal{S} = \{(i, q^k[i]) \mid q^k[i] = q_i q'_i x_i \,\forall k,i\}$ of all traveling states that appear in $r^\star_{team}$. Elements of $\mathcal{S}$ are ordered pairs where the second element is a traveling state and the first element gives the TS this new traveling state will be added to. Next, we construct the set $\mathcal{T} = \{(i,(q^k[i], q^{k+1}[i]), x) \mid ((i, q^k[i]) \in \mathcal{S}) \vee ((i, q^{k+1}[i]) \in \mathcal{S}),\ x = w_{\mathrm{T}}(q^k, q^{k+1})\ \forall k,i\}$ of all transitions that involve any of the traveling states in $r^\star_{team}$. Elements of $\mathcal{T}$ are triplets where the second element is a transition, the third element is the weight of this transition, and the first element shows the TS that this new transition will be added to. Then, we add the traveling states in $\mathcal{S}$ and the transitions in $\mathcal{T}$ to their corresponding TSs. Finally, using Definition 4.4, we project the run $r^\star_{team}$ onto $\{\mathbf{T}_1, \ldots, \mathbf{T}_m\}$ to obtain the individual runs $r^\star_i, i = 1, \ldots, m$.

Example 6.1 Revisited. *For the optimal run* $r^\star_{team}$ *we obtained for this example, we have* $\mathcal{S} = \{(1, ab1), (1, ba1)\}$ *and* $\mathcal{T} = \{(1,(a, ab1), 1), (1,(ab1, b), 1), (1,(b, ba1), 1), (1,(ba1, a), 1)\}$. *Figure 3 illustrates the corresponding TSs with new traveling states and transitions highlighted in red. Then, we have runs of individual robots from Definition 4.4 as* $r^\star_1 = a, b, ba1, a, ab1, b, ba1, a, ab1, \ldots$ *and* $r^\star_2 = a, b, c, b, c, b, c, b, c, \ldots$.

Remark 6.2. *For most applications, adding new states and transitions to the models of the robots may imply introducing new waypoints or motion primitives at lower levels. Since the exact way in which these model changes are accommodated at lower levels is strictly application specific, we do not discuss these details here assuming that such necessary changes can be implemented.*

6.2. Synchronization for trace-closed specifications and optimality bounds

After obtaining individual runs of the robots, we proceed by checking whether the mission specification ϕ is trace-closed using an algorithm adapted from Peled et al. (1998).

We say an LTL formula ϕ is trace-closed if the language L_{B} of the corresponding Büchi automaton is trace-closed in the sense of Definition 2.4.

Proposition 6.3. *If the LTL formula ϕ over the set Π is a trace-closed formula with respect to the distribution $\{\Pi_1, \ldots, \Pi_m\}$ given by TSs $\{\mathbf{T}_1, \ldots, \mathbf{T}_m\}$, then it will not be violated in the field due to uncertain traveling times.*

Proof. From Definitions 2.4 and 3.1, we know that if we can find a run that satisfies a trace-closed LTL formula, then the word ω_{team} corresponding to the run will not be a critical word. We use $\tilde{\omega}_{team}$ to denote the actual word generated by the team during deployment. Since ω_{team} is not a critical word, $\nexists\ \tilde{\omega}_{team} \in [\omega_{team}]$ such that $\tilde{\omega}_{team} \notin L_{\mathrm{B}}$. Thus, regardless of the deviation values of the robots, ϕ will not be violated in the field due to uncertain traveling times as any $\tilde{\omega}_{team} \in [\omega_{team}]$ will also be in L_{B}. ■

Corollary 6.4. *If the LTL formula ϕ over the set Π is not trace-closed with respect to the distribution $\{\Pi_1, \ldots, \Pi_m\}$ given by TSs $\{\mathbf{T}_1, \ldots, \mathbf{T}_m\}$, then ϕ may be violated during deployment due to uncertain traveling times.*

Proof. The proof follows directly from Proposition 6.3. ■

If ϕ is not trace-closed, we cannot guarantee correctness during deployment in general as shown in Corollary 6.4. In cases where the traveling times of the robots are uncertain and ϕ is not trace-closed, we compute individual synchronization sequences $\{s_1, \ldots, s_m\}$ for the robots to guarantee correctness during deployment. We discuss how we generate these synchronization sequences in greater detail in Section 6.3. If, on the other hand, the mission specification ϕ is trace-closed, we can guarantee correctness in the field without any additional measures as shown in Proposition 6.3. Nevertheless, as given in Remark 3.3, the field performance of the team will invariably deviate from its planned value, and in the worst case, the field performance of the team will be limited by that of a single member. To address this issue, we propose a periodic synchronization protocol (Algorithm 3). As the robots execute their infinite runs in the field, they synchronize with each other periodically at the beginning of each repetition of the suffix cycle.

Using this protocol, we can define a bound on the deviation from optimality, i.e. the value of the cost function (2) observed in the field, as given in the following proposition.

Proposition 6.5. *Suppose that each robot's deviation values are bounded by $\underline{\rho}$ and $\overline{\rho}$ where $\overline{\rho} \geq 1 \geq \underline{\rho} > 0$ (i.e. $\underline{\rho_i} \geq \underline{\rho}$ and $\overline{\rho_i} \leq \overline{\rho}$ for each robot i). Let $J(\mathbb{T}^\pi)$ be the cost of the planned robot paths and let $J(\tilde{\mathbb{T}}^\pi)$ be the actual value of the cost observed during deployment. Then, if the robots follow the protocol given in Algorithm 3 the field value of the cost satisfies*

$$J(\tilde{\mathbb{T}}^\pi) \leq J(\mathbb{T}^\pi)\,\overline{\rho} + d_s(\overline{\rho} - \underline{\rho})$$

where d_s is the planned duration of the suffix cycle.

Algorithm 3: TRACE-CLOSED-SYNC-RUN

Input: A run $r_i = q_i^0, q_i^1, \ldots$ of robot i in prefix–suffix form.

```
q_sync ← First state in the suffix cycle.
k ← 0.
while True do
    if current state is q_sync then
        Notify all robots.
        Wait until notification messages of all robots are received.
    Make transition to r_i^{k+1}.
    k ← k + 1.
```

Proof. The suffix consists of an infinite number of repetitions of the suffix cycle, which we denote by S_{c}. As given in Algorithm 3, each repetition of S_{c} begins with a synchronization point where all robots synchronize with each other. Let d_s be the planned duration of S_{c}, let n_s be the number of optimizing propositions satisfied in S_{c}. Let us redefine $t = 0$ to be the time when the suffix starts, and let $\bar{\mathbb{T}}^\pi$ be a sequence of length n_s recording the n_s times that the optimizing proposition is satisfied in the first repetition of S_{c}. Note that, as we consider infinite runs and as the process restarts itself at the beginning of each S_{c} by means of the synchronization protocol given in Algorithm 3, we only need to consider the first two repetitions of S_{c}. We first define

$$\underline{T^i} = \bar{\mathbb{T}}^\pi(i)\,\underline{\rho}$$
$$\overline{T^i} = \bar{\mathbb{T}}^\pi(i)\,\overline{\rho}$$
$$t^w = d_s\overline{\rho}$$

where $\underline{T^i}$ and $\overline{T^i}$ are the earliest and latest times that the ith optimizing proposition can be satisfied, respectively. The value t^w is the latest time that the second repetition of S_{c} can begin. Then, for $0 < i \leq n_s$, the worst-case time between satisfying the ith optimizing proposition and the $(i+1)$th optimizing proposition is

$$\tau^{i,i+1} = \begin{cases} \overline{T^{i+1}} - \underline{T^i} & \text{if } 0 < i < n_s, \\ t^w + \overline{T^1} - \underline{T^{n_s}} & \text{if } i = n_s. \end{cases} \tag{4}$$

Next, in the planned paths, multiple robots may simultaneously satisfy the ith optimizing proposition. In the field, these satisfactions will not occur simultaneously. The maximum amount of time between the first and last of these satisfying instances for the ith proposition, for $0 < i \leq n_s$, is

$$\tau^i = \overline{T^i} - \underline{T^i}. \tag{5}$$

Finally, using (4) and (5) we obtain the upper bound on the value of the cost function (2) that will be observed during deployment as

$$\overline{J(\tilde{\mathbb{T}}^\pi)} = \max\{\max_i\{\tau^{i,i+1}\}, \max_i\{\tau^i\}\}. \tag{6}$$

Substituting the definitions for $\overline{T^i}$, $\underline{T^i}$, and t^w into (4) we obtain

$$\tau^{i,i+1} = \begin{cases} \bar{\mathbb{T}}^\pi(i+1)\,\overline{\rho} - \bar{\mathbb{T}}^\pi(i)\,\underline{\rho} & \text{if } 0 < i < n_s, \\ \big(d_s + \bar{\mathbb{T}}^\pi(1)\big)\overline{\rho} - \bar{\mathbb{T}}^\pi(n_s)\,\underline{\rho} & \text{if } i = n_s \end{cases}$$

But we have that $J(\mathbb{T}^\pi) \geq \bar{\mathbb{T}}^\pi(i+1) - \bar{\mathbb{T}}^\pi(i)$, and $J(\mathbb{T}^\pi) \geq d_s + \bar{\mathbb{T}}^\pi(1) - \bar{\mathbb{T}}^\pi(n_s)$. In addition, $\bar{\mathbb{T}}^\pi(1) \leq J(\mathbb{T}^\pi)$ and $\bar{\mathbb{T}}^\pi(i) \leq d_s$ for all $i \in \{2, \ldots, n_s\}$. Using these expressions we obtain $\tau^{i,i+1} \leq J(\mathbb{T}^\pi)\,\overline{\rho} + d_s(\overline{\rho} - \underline{\rho})$. Similarly, we get $\tau^i \leq d_s(\overline{\rho} - \underline{\rho})$, and thus $J(\tilde{\mathbb{T}}^\pi) \leq J(\mathbb{T}^\pi)\,\overline{\rho} + d_s(\overline{\rho} - \underline{\rho})$. ■

Remark 6.6 (Exact bound on $J(\tilde{\mathbb{T}}^\pi)$). *In Proposition 6.5, we have provided a conservative bound for ease of presentation. However, we can also calculate an exact bound on the field value of the cost $J(\tilde{\mathbb{T}}^\pi)$ using a treatment similar to the proof of Proposition 6.5.*

6.3. Synchronization for general specifications and guarantee of correctness

If the traveling times of the robots are uncertain and ϕ is not trace-closed, we compute individual synchronization sequences $\{s_1, \ldots, s_m\}$ for the robots to guarantee correctness during deployment. As the robots execute their infinite runs in the field, they synchronize with each other according to the synchronization sequences that we generate using Algorithm 4. The synchronization sequence s_i of robot i is an infinite sequence of pairs of sets. The kth element of s_i, denoted by s_i^k, corresponds to the kth element q_i^k of $r_i^\star$. Each s_i^k is a pair of two sets of robots: $s_i^k = (s_{i,wait}^k, s_{i,notify}^k)$, where $s_{i,wait}^k$ and $s_{i,notify}^k$ are the *wait-set* and *notify-set* of s_i^k, respectively. The *wait-set* of s_i^k is the set of robots that robot i must wait for at state q_i^k before satisfying its propositions and proceeding to the next state q_i^{k+1} in $r_i^\star$. The *notify-set* of s_i^k is the set of robots that robot i must notify as soon as it reaches state q_i^k. As we discussed earlier in Section 4.2, the optimal run $r_{team}^\star$ of the team and the individual optimal runs $r_i^\star, i = 1, \ldots, m$ of the robots are always in prefix–suffix form. Consequently, individual synchronization sequences s_i of the robots are also in prefix–suffix form.

Algorithm 4 is essentially a loop (lines 3–16) that computes the wait-sets for each position of the runs of the robots to guarantee correctness in the field. Initially, synchronization sequences are set so that the robots wait for each other at every position of their runs (line 2). At line 4 of Algorithm 4, if k is the first position of the runs, we do not modify this initial value of $s_{i,wait}^k$. This ensures that all robots start executing their runs in a synchronized way. We also keep this initial value of $s_{i,wait}^k$ if k is the beginning of the suffix cycle, so that all robots synchronize with each other globally at the beginning of each suffix cycle. This lets us define a bound on the deviation from optimality, i.e. the value of the cost function (2) observed in the field, as given in Proposition 6.5. For all other positions of the runs, we try to shrink the wait-set of each s_i^k so that communication

Algorithm 4: SYNC-SEQ

Input: Individual runs $\{r_1^\star, \ldots, r_m^\star\}$, Büchi automaton $\mathbf{B}_{\neg\phi}$ of $\neg\phi$, and models of the robots.
Output: Synchronization sequence for each robot $\{s_1, \ldots, s_m\}$.

1 $\mathcal{I} = \{1, \ldots, m\}$, beg = beginning of suffix cycle, end = end of suffix cycle.
2 $s_{i,wait}^k = \mathcal{I} \setminus i$ for $i \in \mathcal{I}$ and $k = 0, \ldots, end$.
3 **foreach** $k = 0, \ldots, end$ **do**
4 **if** $k \neq 0$ *and* $k \neq beg$ **then**
5 Set $s_{i,wait}^k = \emptyset\ \forall\ i \in \mathcal{I}$.
6 Construct the TS $\mathbf{W}$ that generates every possible $\tilde{\omega}_{team}$ (Algorithm 6).
7 **if** *the language of* $\boldsymbol{B}_{\neg\phi} \times \boldsymbol{W}$ *is empty* **then**
8 Continue to next position k in run.
9 **else**
10 Set $s_{i,wait}^k = \mathcal{I} \setminus i\ \forall\ i \in \mathcal{I}$.
11 **foreach** $i \in \mathcal{I}$ **do**
12 **foreach** $j \in \mathcal{I} \setminus i$ **do**
13 Remove j from $s_{i,wait}^k$.
14 Construct the TS $\mathbf{W}$ that generates every possible $\tilde{\omega}_{team}$ (Algorithm 6).
15 **if** *the language of* $\mathbf{B}_{\neg\phi} \times \mathbf{W}$ *is not empty* **then**
16 Add j back to $s_{i,wait}^k$.
17 Define each $s_{i,notify}^k$ such that $i \in s_{j,wait}^k \Rightarrow j \in s_{i,notify}^k\ \forall\ i \in \mathcal{I}, j \in \mathcal{I}, k = 0, \ldots, end$.
18 The rest of each s_i is an infinite repetition of its suffix cycle, i.e. $s_i^{beg}, \ldots, s_i^{end}, \forall\ i \in \mathcal{I}$.

effort is minimized while we can still guarantee correctness in the field (lines 5–16). To this end, we first consider the case where robots do not wait for each other at this position of the run (lines 5–8). This is actually a heuristic based on the observation that in most missions robots synchronize only occasionally. We set all wait-sets corresponding to this position to empty sets. Then, given the runs, TSs, deviation values, and wait-sets of the robots, we use Algorithm 6 to construct the TS $\mathbf{W}$ that generates all possible words $\tilde{\omega}_{team}$ that can be observed in the field due to the uncertainties in the traveling times. Next, we construct the product $\mathbf{B}_{\neg\phi} \times \mathbf{W}$, where $\mathbf{B}_{\neg\phi}$ is the Büchi automaton corresponding to the negation of the LTL formula ϕ. If the language of this product is empty, then the robots indeed do not need to synchronize at this position. Otherwise, we restore the previous values of the wait-sets of this position (line 10) and consider each one of the robots in robot i's kth wait-set $s_{i,wait}^k$ one by one (lines 11–16). After removing some robot j from $s_{i,wait}^k$, we construct $\mathbf{W}$ and check whether the language of $\mathbf{B}_{\neg\phi} \times \mathbf{W}$ is empty (lines 13–15). If the language

Algorithm 5: SYNC-RUN

Input: The run r_i and synchronization sequence s_i of robot i .

1 $k \leftarrow 0$.
2 **while** *True* **do**
3 **Notify** all robots in $s^k_{i,notify}$.
4 **Wait** until notification messages of all robots in $s^k_{i,wait}$ are received.
5 Make transition to r^{k+1}_i after satisfying the propositions at r^k_i.
6 $k \leftarrow k + 1$.

of the product is empty, then robot i indeed does not need to wait for robot j at the kth position of its run. Thus, we keep the new value of $s^k_{i,wait}$. Otherwise, we restore $s^k_{i,wait}$ to its previous value (line 16) and proceed with the next robot in $s^k_{i,wait}$. Once every robot in $s^k_{i,wait}$ is considered, we proceed with the next robot in the team, and eventually next position of the run. Note that, the synchronization sequences generated by Algorithm 4 are free from any dead-locks as line 17 ensures that if some robot j waits for robot i at position k, then robot i notifies robot j at position k. As the synchronization sequences of the robots are in prefix–suffix form and the robots synchronize with each other globally at the beginning of each suffix cycle (line 4), at line 18, we define the rest of each synchronization sequence as an infinite repetition of its first suffix cycle that we have just generated. Let K denote the total length of the prefix and the first suffix cycle. Then, the worst-case complexity of Algorithm 4 is $O(m^2K(W + E))$ where m is the number of robots, W is the complexity of constructing $\mathbf{W}$, and E is the complexity of checking emptiness of $\mathbf{W} \times \mathbf{B}_{\neg\phi}$ at each iteration. If the robots need to synchronize only occasionally, i.e. if the heuristic at lines 5–8 succeeds most of the time, then the complexity is $O(K(W + E))$. The synchronization protocol that the robots follow in the field is given in Algorithm 5.

We use Algorithm 6 to construct the TS $\mathbf{W}$ that generates all possible words that can be observed in the field for a given set of runs and synchronization sequences of the robots. We must first define some new terms before getting into the details of Algorithm 6. We use the term *position* to refer to the current position of a robot in its run. If some robot i has just reached the state r^k_i in its run and satisfied the corresponding propositions after waiting for all of the robots in its wait-set $s^k_{i,wait}$ as given in Algorithm 5, then the position of the robot is k. If, on the other hand, robot i has left state r^{k-1}_i, but one of the above conditions has not been satisfied yet, then the position of the robot is $(k-1, k)$. A *robot–position* pair is a pair of the form (i, p) meaning that the position of robot i is p which can be either an integer or a pair of integers, as discussed above. For instance, the robot–position pair $(i, (k-1, k))$ means robot i is on its way from state r^{k-1}_i to state r^k_i. An *event* is a set of one or more robot–position pairs that give the new positions of the corresponding robots. In the case of multiple robot–position pairs, all of these changes occur simultaneously. That is, the event $\{(i, k), (j, k)\}$ means that robots i and j have just reached position k in their runs. On the other hand, the event $\{(i, k)\}$ means that robot i has just reached position k and gives no information about the position of robot j. Finally, an *event sequence* is a list of events that occur sequentially. Now we can begin discussing Algorithm 6. The states of $\mathbf{W}$ are tuples of positions such that the ith element $q[i]$ of some state $q \in \mathcal{Q}_W$ gives the current position of robot i. Consequently, at line 1 we set $(0, \ldots, 0)$ to be the initial state of $\mathbf{W}$ as we assume that the robots start their runs synchronously (Algorithm 4). Algorithm 6 is essentially a loop (lines 2–12) that considers all possible sequences of events that may occur in the field. To do this, Algorithm 6 relies on Algorithm 8 to generate pairs of event sequences and corresponding sets of states of $\mathbf{W}$ where those event sequences start. For an event sequence and the corresponding set of start states generated using Algorithm 8, Algorithm 6 adds the necessary states and transitions to $\mathbf{W}$ starting from each possible start state (lines 3–12). Then, at line 5, we consider all events in an event sequence one by one. At lines 6–9, we compute the next state q' after the event e occurs at state q. If the position of some robot i changes due to event e, then $q'[i]$ is set to the new position given in e (line 7). Otherwise we update the position of robot i to capture its progress. If the position of robot i is already a tuple in q, i.e. if robot i is already on road, then we do not change its position in q' (line 8). Otherwise, we update the position of robot i in q' such that it starts traveling towards the next state in its run (line 9). Next, we add the new state q' with the necessary propositions and the new transition (q, q') to $\mathbf{W}$ as required (lines 10–11). Then, we set the current state q of $\mathbf{W}$ to q' and switch to the next event e in the event sequence. Once we process all of the events in this event sequence for all start states, we repeat the same procedure for the next event sequence. Since the runs of the robots are in prefix–suffix form, Algorithm 8 is designed such that it terminates once the positions of the robots reach the end of the first suffix cycle. Since the robots start each suffix cycle in a synchronized way (Algorithm 4), at line 14 of Algorithm 6 we add a transition from all of those states with no outgoing transitions to the state that corresponds to the beginning of the suffix cycle. This final step concludes the construction of $\mathbf{W}$ by capturing the periodic structure of the runs of the robots. In order not to interrupt the flow of the paper, we present and discuss the complexity of Algorithms 8 and 9, which we use to generate the event sequences discussed above, in Appendix A. Next, we characterize the complexity of Algorithm 6.

Proposition 6.7. *Let K denote the total length of the prefix and the first suffix cycle. For the case where the intervals of*

Algorithm 6: CONSTRUCT-FIELD-WORDS-TS

Input: $\{r_1, \ldots, r_m\}$, $\{s_{1,wait}, \ldots, s_{m,wait}\}$, $\{\mathbf{T}_1, \ldots, \mathbf{T}_m\}$, and $\overline{\rho_i}, \underline{\rho_i}$, $i = 1, \ldots, m$.

Output: The field words TS $\mathbf{W}$ that generates all possible words that can be observed in the field.

1 Add $q^0_{\mathrm{W}} = (0, \ldots, 0)$ to $\mathcal{Q}_{\mathrm{W}}$.
2 **foreach** (*event_seq*, *start_states*) *generated using* GENERATE-EVENT-SEQ *(Algorithm 8)* **do**
3 **foreach** q_{start} *in start_states* **do**
4 $q = q_{start}$.
5 **foreach** *e in event_seq* **do**
6 **foreach** $i \in \{1, \ldots, m\}$ **do**
7 **if** $(i, k_{new}) \in e$ **then** $q'[i] = k_{new}$.
8 **else if** $q[i]$ *is a tuple* **then** $q'[i] = q[i]$.
9 **else** $q'[i] = (q[i], q[i] + 1)$.
10 **if** q' *is not in* $\mathcal{Q}_{\mathrm{W}}$ **then** add q' to $\mathcal{Q}_{\mathrm{W}}$ with $\mathcal{L}_{\mathrm{W}}(q') = \cup_{(i,k)\in e}\mathcal{L}_i(r^k_i)$.
11 **if** (q, q') *is not in* δ_{W} **then** add (q, q') to δ_{W}.
12 $q = q'$.
13 $q_{suffix} = (beg, \ldots, beg)$ where *beg* corresponds to the beginning of the suffix cycle.
14 **foreach** $q \in \mathcal{Q}_{\mathrm{W}}$ *such that* $\nexists (q, q') \in \delta_{\mathrm{W}}$ *for any* $q' \in \mathcal{Q}_{\mathrm{W}}$ **do** add (q, q_{suffix}) to δ_{W}.

the robots corresponding to different positions do not overlap (discussed in greater detail in Appendix A), the complexity of Algorithm 6 is $O(4^m m^{2m+7} K^2)$ *and the number of states of* $\mathbf{W}$ *is* $O(2^m m^{m+3} K)$.

Proof. From Propositions 8.1 and 8.3, for the given case, we have at most $O(2^m m^{m+2} K)$ event sequences in the prefix and the first suffix cycle with at most m events each. Since Algorithm 6 creates a new state for each new event, the number of states of $\mathbf{W}$ is $O(2^m m^{m+3} K)$. Consequently, each of the event sequences generated by Algorithm 8 can have at most $O(2^m m^{m+3} K)$ different start states. Also, the complexity of the inner loop of Algorithm 6 (lines 5–9) is $O(m^2)$. Thus, the complexity of Algorithm 6 is $O(4^m m^{2m+7} K^2)$. ■

Remark 6.8. *In Proposition 6.7 we assumed that the intervals of the robots corresponding to different positions do not overlap. Let* t_n *denote the planned time until the robots reach the nth position in their runs and K denote the total length of the prefix and the first suffix cycle. The above condition is satisfied when* $\overline{\rho_i} t_{n-1} < \underline{\rho_j} t_n$ *holds for all* $i, j \in \{1, \ldots, m\}$ *and* $n = 1, \ldots, K-1$. *This is typically the case where the deviation values of the robots are small enough (with respect to the length of the suffix cycle and durations between consecutive states in the run) such that the intervals in which the robots can reach different positions in their runs do not overlap. A more general complexity analysis could be performed for the case where robots move to different positions in a single interval, but at the cost of increased difficulty of presentation and interpretation. We employ the same assumption in Propositions 6.10, 8.1, and 8.3 for the same reason.*

Example 6.1 Revisited. *For the example we have shown throughout this section, we obtain the following individual optimal runs and synchronization sequences.*

$\mathbb{T}$	*0*	*2*	*3*	*4*	*5*	*6*	...
$r^\star_1$	*a*	*b*	*ba1*	*a*	*ab1*	*b*	...
s_1	$(\{2\},\{2\})$	$(\emptyset,\emptyset)$	$(\{2\},\{2\})$	$(\emptyset,\emptyset)$	$(\emptyset,\emptyset)$	$(\emptyset,\emptyset)$	...
$\mathcal{L}_1(.)$		$\mathtt{p}_1, \pi$				$\mathtt{p}_1, \pi$	...
$r^\star_2$	*a*	*b*	*c*	*b*	*c*	*b*	...
s_2	$(\{1\},\{1\})$	$(\emptyset,\emptyset)$	$(\{1\},\{1\})$	$(\emptyset,\emptyset)$	$(\emptyset,\emptyset)$	$(\emptyset,\emptyset)$	...
$\mathcal{L}_2(.)$		$\mathtt{p}_2, \pi$	$\mathtt{p}_3$	$\mathtt{p}_2, \pi$	$\mathtt{p}_3$	$\mathtt{p}_2, \pi$	...

In a line corresponding to a synchronization sequence s_i, *first and second elements of the tuple at position k are* $s^k_{i,wait}$ *and* $s^k_{i,notify}$, *respectively. The symbol* $\emptyset$ *denotes an empty wait-set, or notify-set, i.e. the robot does not wait for, or notify, any other robot at that position of its run.*

We finally summarize our robust solution in Algorithm 7, and show that it provides a solution to Problem 3.4. We analyze the overall complexity of Algorithm 7 in Proposition 6.10.

Algorithm 7: ROBUST-MULTI-ROBOT-OPTIMAL-RUN

Input: Transition systems $\{\mathbf{T}_1, \ldots, \mathbf{T}_m\}$, corresponding deviation values and an LTL specification ϕ of the form (1).

Output: A set of runs $\{r^\star_1, \ldots, r^\star_m\}$ that satisfies ϕ and minimizes (2), a set of synchronization sequences $\{s_1, \ldots, s_m\}$ that guarantees correctness in the field (if applicable), and the bound on the performance of the team in the field.

1 Construct the team TS $\mathbf{T}$ using Algorithm 1.
2 Find an optimal run $r^\star_{team}$ on $\mathbf{T}$ using OPTIMAL-RUN (Smith et al., 2011).
3 Insert new traveling states to TSs $\{\mathbf{T}_1, \ldots, \mathbf{T}_m\}$ (see Section 6.1).
4 Obtain individual runs $\{r^\star_1, \ldots, r^\star_m\}$ using Definition 4.4.
5 **if** ϕ is not trace-closed **then**
6 Generate synchronization sequences $\{s_1, \ldots, s_m\}$ using SYNC-SEQ (Algorithm 4).
7 Find the bound on optimality as given in Proposition 6.5.

Proposition 6.9. *Algorithm 7 solves Problem 3.4 when the traveling times of the robots are uncertain during deployment.*

Proof. Note that Algorithm 7 combines all steps outlined in this section. The planned word ω_{team} generated by the entire team satisfies ϕ, and minimizes (2), as shown in Smith et al. (2011). If the mission specification ϕ is trace-closed, correctness during deployment is guaranteed by construction as given in Proposition 6.3. If ϕ is not trace-closed, the synchronization sequences guarantee correctness by ensuring that the $\tilde{\omega}_{team}$ generated in the field never violates ϕ for given deviation values. Therefore, Algorithm 7 solves Problem 3.4. ■

Proposition 6.10. *Suppose that a group of m identical robots are expected to satisfy an LTL specification ϕ in a common environment with Q vertices, Δ edges and a largest edge weight of w_{max}. Further assume that K is the total length of the prefix and the first suffix cycle of the optimal satisfying run, and the intervals of the robots corresponding to different positions do not overlap. Then, for typical cases where $m \ll Q$, $K < Q$, complexity of Algorithm 7 is $O((Q^m + \Delta^m w_{max})^3 \cdot 2^{O(|\phi|)})$.*

Proof. For the above mentioned case, the worst-case complexity of lines 1–4 of Algorithm 7 becomes $O((Q^m + \Delta^m w_{max})^3 \cdot 2^{O(|\phi|)})$ from Proposition 5.3. The trace-closedness check (line 5) can be done in time $O(2^{O(|\phi|)} 2^{2^{O(|\phi|)}})$ (Peled et al., 1998). If this check fails, we generate synchronization sequences using Algorithm 4, which runs in time $O(m^2 K(W + E))$. From Proposition 6.7, W is $O(4^m m^{2m+7} K^2)$ and the number of states of $\mathbf{W}$ is $O(2^m m^{m+3} K)$. Thus, E is $O(2^{O(|\neg\phi|)} 2^m m^{m+3} K)$ (Baier and Katoen, 2008) and complexity of Algorithm 4 becomes $O(4^m m^{2m+9} K^3 + 2^{O(|\neg\phi|)} 2^m m^{m+5} K^2)$. Note that the check for trace-closedness at line 5 of Algorithm 7 can be omitted for long formulas by simply assuming that the result is false and proceeding with the generation of the synchronization sequences using Algorithm 4. Then, the complexity of Algorithm 7 is $O((Q^m + \Delta^m w_{max})^3 2^{O(|\phi|)} + 4^m m^{2m+9} K^3 + 2^{O(|\neg\phi|)} 2^m m^{m+5} K^2)$. For typical cases where $m \ll Q$ and $K < Q$, the complexity becomes $O((Q^m + \Delta^m w_{max})^3 \cdot 2^{O(|\phi|)})$. ■

Remark 6.11. *In cases where the conditions given in Propositions 6.7 and 6.10 do not hold, the computational cost of computing synchronization sequences using Algorithm 4 may be undesirably high. In such cases, one can trade communication effort for computational complexity by deploying the robots using the trivially correct synchronization sequence given at line 2 of Algorithm 4 where each robot waits for every other robot at each position of the run. Note that the bound on field performance given in Proposition 6.5 still holds in this case.*

7. Implementation and case studies

We implemented our algorithms in Python as the LTL Optimal Multi-Agent Planner (LOMAP) package, which is publicly available online.[2] LOMAP uses the NetworkX graph package described by Hagberg et al. (2008) to represent various models in our implementation and the LTL2BA software described by Gastin and Oddoux (2001) to convert LTL specifications to Büchi automata. LOMAP also includes an enhanced version of the OPTIMAL-RUN algorithm (Smith et al., 2011) which returns the path with the shortest suffix cycle when there are multiple optimal paths in terms of the cost function (2). Furthermore, this new version can be executed on a computer cluster in a distributed fashion to be able to solve problems with large resource requirements. A typical usage of our package is as follows.

(i) The user defines the TSs $\{\mathbf{T}_1, \ldots, \mathbf{T}_m\}$ that model the robots moving in the environment in a plain text file using LOMAP's format.
(ii) Then, the user writes a short python script that defines the mission specification expressed in LTL in the form of (1) and calls the appropriate LOMAP function.
(iii) Finally, the trajectory of the team and the value of the cost function are returned if the mission specification can be satisfied. Otherwise, our implementation shows an error message and quits.

7.1. Experimental case studies on persistent surveillance

In the following, we present various case studies considering persistent surveillance missions in the environment shown in Figures 4(a) and 4(b). This environment is a road network consisting of roads, intersections, and regions for data gathering and upload. In this network, road segments are connected to each other via intersections, and the surveillance target is located in the middle, surrounded by four data gathering locations. For our case studies, we considered two Pololu m3pi robots with mbed development boards. We realized the environment using lines of black tape that correspond to the roads and intersections of the road network. The robots can navigate in the environment and can sense whether they are at an intersection or not using their infrared reflection sensors. The robots can also communicate with each other and a computer using Xbee wireless modules. In our case studies, inter-robot communication is used for synchronization of the robots, whereas computer–robot communication is used for deploying the robots according to the trajectory generated using our implementation.

The robots that we consider in our experiments have uncertain traveling times. In order to obtain their upper and lower deviation values, we measured the time it takes for both of the robots to complete the cycle "U2, 10, 11, 12, 1, 2, 21, 22, 23, 9, 10, U2" in Figure 4(c) and recorded the maximum and minimum values among 20 trials. We chose this cycle because it tests all the motion primitives of the robots: "*left-turn, right-turn, u-turn,* and *go-straight*". The average time for both robots to complete this cycle was approximately 17 seconds. We used this information to obtain the weights of the model given in Figure 4(c),

which were used as the *nominal* values in our computations. The maximum and minimum times for robot 1 to complete this cycle were 17.67 and 16.68 seconds, respectively. The maximum and minimum times for robot 2 were 17.56 and 16.77 seconds, respectively. Using these measurements we obtained the following deviation values: $\overline{\rho_1} = 1.039, \underline{\rho_1} = 0.981, \overline{\rho_2} = 1.033, \underline{\rho_2} = 0.986$. In the following, we take these deviation values as $\overline{\rho_1} = \overline{\rho_2} = 1.04$ and $\underline{\rho_1} = \underline{\rho_2} = 0.98$ after adding a small margin of safety.

Figure 4(c) illustrates the TSs $\mathbf{T}_1$ and $\mathbf{T}_2$ that model the motion of the robots in this road network. The sets of states $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are the sets of labels assigned to intersections and regions. The transition relations δ_1 and δ_2 give how the intersections and regions are connected and the weight maps w_1 and w_2 capture the time it takes for robots to take a transition. For our experiments, we assume that the TSs $\mathbf{T}_1$ and $\mathbf{T}_2$ are identical except for their initial states and the sets of propositions that can be satisfied at their states. To be able to differentiate between data gatherings and uploads performed at different locations by different robots we define the set of propositions as

Π = {gather, upload, r1gather, r2gather, r1upload, r2upload, gather1, gather2, gather3, gather4, upload1, upload2, r1gather1, r1gather2, r1gather3, r1gather4, r2gather1, r2gather2, r2gather3, r2gather4, r1upload1, r1upload2, r2upload1, r2upload2}.

Propositions `gather` and `upload` mean data has been gathered and uploaded, respectively, whereas propositions of the form `gatherY` and `uploadY`, where `Y` $\in \{1, 2, 3, 4\}$, capture the locations of data gather and upload as well. For instance, `gather3` means data has been gathered at gather location 3. Propositions of the form `rXgather` and `rXupload`, where `X` $\in \{1, 2\}$, mean robot `X` has gathered and uploaded data, respectively. Finally, we use propositions of the form `rXgatherY` and `rXuploadY`, where `X` $\in \{1, 2\}$ and `Y` $\in \{1, 2, 3, 4\}$, to capture both the location and the subject of the data gather and upload, i.e. `r2Upload1` means robot 2 has uploaded data at upload location 1. Consequently, we define the sets Π_1 and Π_2 as

Π_1 = {gather, upload, r1gather, r1upload, gather1, gather2, gather3, gather4, upload1, upload2, r1gather1, r1gather2, r1gather3, r1gather4, r1upload1, r1upload2}, and

Π_2 = {gather, upload, r2gather, r2upload, gather1, gather2, gather3, gather4, upload1, upload2, r2gather1, r2gather2, r2gather3, r2gather4, r2upload1, r2upload2};

and assign the propositions in Π_1 and Π_2 to the states of $\mathbf{T}_1$ and $\mathbf{T}_2$ as given in Table 1. Note that all propositions in Π can be written in terms of the propositions of the last form, and therefore we could have a set Π consisting of 12 propositions of the form `rXgatherY` and `rXuploadY`. However, for the sake of clarity and simplicity, we choose to define Π as given above, because otherwise we would have to use the long boolean expression `r1Gather1` $\vee \ldots \vee$ `r1Gather4`$\vee$`r2Gather1`$\vee \ldots \vee$`r2Gather4` to express a data gather event, instead of using a single proposition, i.e. `gather`.

For the case studies presented next, we ran LOMAP on a computing cluster consisting of five *m2.2xlarge* Amazon Elastic Compute Cloud[3] instances each with 34.2 GB of memory and 2.67 GHz quad-core processing power. As shown in Figure 4(c) TSs $\mathbf{T}_1$ and $\mathbf{T}_2$ of both of the robots have 26 states. Table 2 gives the state count of the team TS, Büchi automaton and the product automaton (product of the Büchi automaton and the team TS constructed by the OPTIMAL-RUN (Smith et al., 2011) algorithm to solve the path planning problem) along with total computation time for each individual case study. Since we consider the same robot model for all case studies presented in this section, the state count of the team TS $\mathbf{T}$ is 2444 for all case studies. We investigate the scalability of our approach in the number of robots and the size of the environment considering a small academic example in Section 7.2.

Case study 1. The first mission specification that we consider is as follows: "Each robot must repeatedly visit data gather locations to gather data and go to an upload location to upload their data before gathering data again. The maximum time between successive data gatherings must be minimized.". This mission specification can be expressed in LTL in the form of (1) as

$$\phi_1 := \mathbf{G}(\texttt{r1gather} \Rightarrow \mathbf{X}(\neg \texttt{r1gather}\, \mathcal{U}\, \texttt{r1upload})) \wedge \mathbf{G}(\texttt{r2gather} \Rightarrow \mathbf{X}(\neg \texttt{r2gather}\, \mathcal{U}\, \texttt{r2upload})) \wedge \mathbf{GF}\pi,$$

where $\pi :=$ `gather` is set as the optimizing proposition. Since the traveling times of our robots are uncertain, we use our robust solution (Section 6). It takes 32.5 minutes for our method to obtain an optimal satisfying team trajectory, and the cost in terms of (2) is 10. For this case, since ϕ_1 is trace-closed, the robots synchronize only at the beginning of their suffix cycles. The upper bound on the value of the cost as given by Proposition 6.5 is 11.6 seconds whereas the maximum value of the cost observed in the field after 10 iterations of this trajectory was 10.66 seconds. For comparison, it also takes approximately 32.5 minutes for our exact solution to return the same trajectory with the same cost. Figure 5(a) illustrates the optimal team trajectory that we obtain for formula ϕ_1. As discussed in Section 4.2, optimal satisfying runs obtained using our approach always consist of a finite prefix followed by infinite repetitions of a finite suffix cycle. In the figures that we present in this section, we omit the prefix for the sake of clarity, and use red and blue lines to illustrate the infinite periodic runs of robots 1 and

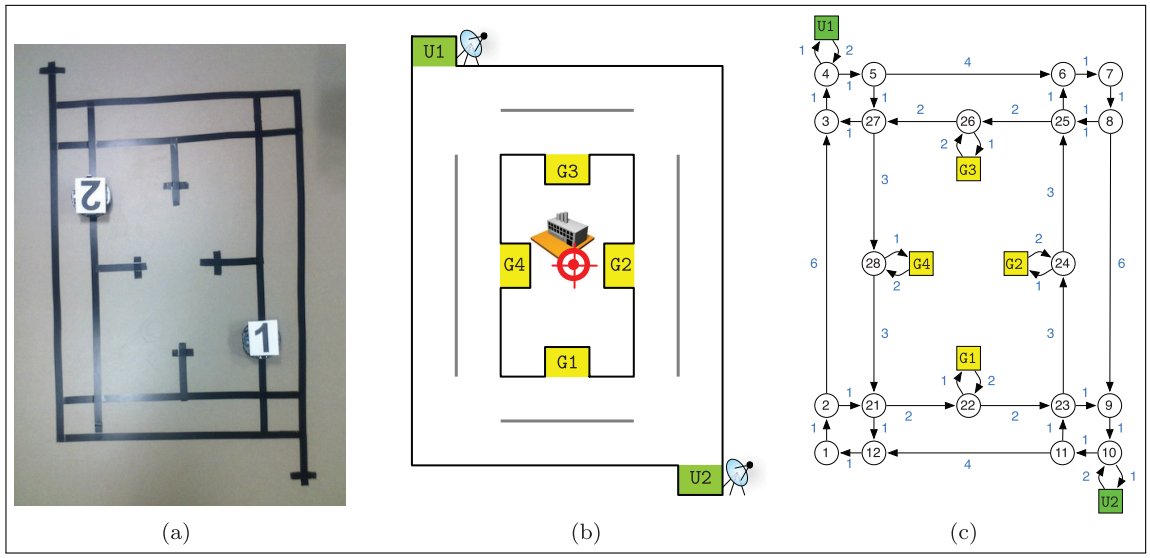

Fig. 4. Figure 4(a) shows our experimental platform where the roads are marked by black tape and the robots are labeled 1 and 2. Figure 4(b) gives a schematic illustration of this road network. The surveillance target is in the middle. Regions highlighted in yellow are data gathering locations and regions highlighted in green are data upload locations. The TS that models the motion of the robots is given in Figure 4(c). The weight of each transition captures the time it takes for the robots to complete that transition.

Table 1. Assignment of the propositions to the regions in the environment.

Region	Propositions of robot 1	Propositions of robot 2
G1	{gather, gather1, r1gather, r1gather1}	{gather, gather1, r2gather, r2gather1}
G2	{gather, gather2, r1gather, r1gather2}	{gather, gather2, r2gather, r2gather2}
G3	{gather, gather3, r1gather, r1gather3}	{gather, gather3, r2gather, r2gather3}
G4	{gather, gather4, r1gather, r1gather4}	{gather, gather4, r2gather, r2gather4}
U1	{upload, upload1, r1upload, r1upload1}	{upload, upload1, r2upload, r2upload1}
U2	{upload, upload2, r1upload, r1upload2}	{upload, upload2, r2upload, r2upload2}

Table 2. Quantitative information on the case studies presented in Section 7.1.

Case study	State count of the team tran. sys.	State count of the Büchi automaton	State count of the product automaton	Total computation time
1	2444	12	17,952	1946 s
2	2444	12	15,080	26 s
3	2444	12	15,072	47 s
4	2444	12	15,050	20 s
5	2444	5	9895	1404 s

2, respectively. We use filled circles to represent the beginning of the suffix cycles of the robots and white triangles to represent the synchronization points.

Case study 2. In some missions, sequential data gatherings at different locations may not be enough to obtain the desired information about the surveillance target. In such cases, synchronous data gatherings by multiple robots may be more desirable. For instance, one can use photographs taken synchronously from different angles to recover depth information which may be used to construct an approximate three-dimensional model of the surveillance target. Also, time-synchronous eavesdropping of radio communications at different locations may substantially increase the chances of recovering useful information from surveillance data. An example mission specification for such a case would be: "Robots must repeatedly gather data in a synchronous fashion, and upload their data before gathering data again.". This mission specification can be written in LTL as

$$\begin{aligned}\phi_2 := \mathbf{G}(\texttt{gather} \Rightarrow (\texttt{r1gather} \wedge \texttt{r2gather})) \\ \wedge \mathbf{G}(\texttt{r1gather} \Rightarrow \mathbf{X}(\neg \texttt{r1gather} \mathcal{U} \texttt{r1upload})) \wedge \\ \mathbf{G}(\texttt{r2gather} \Rightarrow \mathbf{X}(\neg \texttt{r2gather} \mathcal{U} \texttt{r2upload})) \wedge \mathbf{GF}\pi\end{aligned}$$

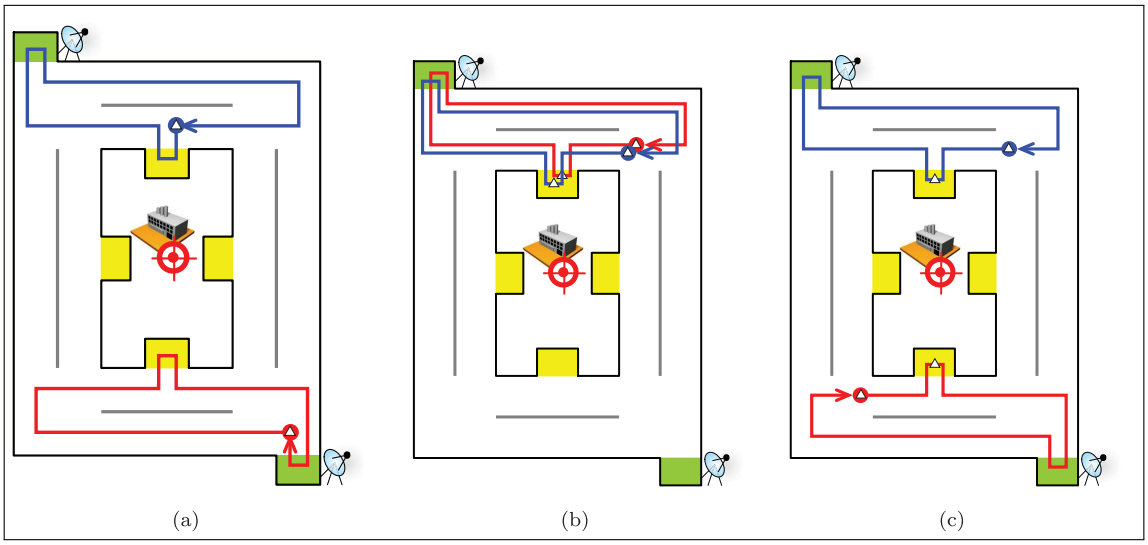

Fig. 5. Team trajectories for case studies 1, 2, and 3. Red and blue lines illustrate trajectories of robot 1 and 2, respectively. Yellow regions are data gathering locations and green regions are data upload locations. Filled circles represent the beginning of the suffix cycles of the robots and the white triangles represent synchronization points.

where $\pi := \texttt{r1gather} \wedge \texttt{r2gather}$. Both of our robust (Section 6) and exact (Section 5) solutions take approximately 26 seconds to compute the trajectory illustrated in Figure 5(b). The cost of this trajectory in terms of (2) is 20. The significant drop in computation time from case study 1 can be explained by the reduction in the size of the solution space in which the OPTIMAL-RUN algorithm has to work. The previous case-study requires 4664 executions of Dijkstra's algorithm, whereas this case study requires only 680 executions of Dijkstra's algorithm on a significantly smaller graph. We were, however, unable to execute this trajectory as our experimental setup does not allow multiple robots to be at the same region at the same time. Next, we discuss how we can address this issue and obtain a more desirable run.

Case study 3. Figure 5(b) shows that lock-step motion of the robots is an optimal team trajectory for ϕ_2. However, as our motivation for synchronous surveillance is to gather data synchronously from different locations, we can include this requirement in our specification to eliminate such undesired behaviors. Then, the mission specification can be written as

$$\begin{aligned}\phi_3 :=& \phi_2 \wedge \mathbf{G}(\neg(\texttt{r1gather1} \wedge \texttt{r2gather1}) \\ & \wedge \neg(\texttt{r1gather2} \wedge \texttt{r2gather2}) \wedge \\ & \neg(\texttt{r1gather3} \wedge \texttt{r2gather3}) \\ & \wedge \neg(\texttt{r1gather4} \wedge \texttt{r2gather4}))\end{aligned}$$

where ϕ_2 is the specification of the previous case study with $\pi := \texttt{r1gather} \wedge \texttt{r2gather}$ and the rest of ϕ_3 forbids robots to gather data at the same place at the same time. Figure 5(c) illustrates the optimal team trajectory we obtain for ϕ_3 using our robust approach. Note that in addition to synchronizing at the beginning of their suffix cycles,

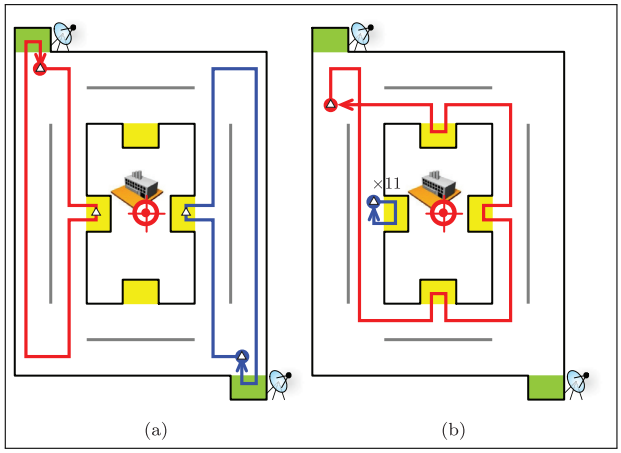

Fig. 6. Team trajectories for case studies 4 and 5. Red and blue lines illustrate trajectories of robots 1 and 2, respectively. Yellow regions are data gathering locations and green regions are data upload locations. Filled circles represent the beginning of the suffix cycles of the robots and the white triangles represent synchronization points.

the robots also synchronize with each other before gathering data in order not to violate the mission specification. It takes 47 seconds for our robust solution to compute this trajectory and the cost is 20. During 10 iterations of this trajectory, the maximum value of the cost observed in the field never exceeded the upper bound of 22 seconds given by our approach. Extension 1 shows the execution of this trajectory by the robots.

Case study 4. Now we consider the case where we need to assign each robot a specific region for data gathering while still requiring them to gather data synchronously. This is typical in scenarios where data gathering capabilities of the

robots are not identical and the robots need to visit specific regions to gather useful surveillance. An example specification where robot 1 is assigned to G4 and robot 2 is assigned to G2 would be

$$\phi_4 := \mathbf{G}(\texttt{gather} \Rightarrow (\texttt{r1gather4} \wedge \texttt{r2gather2})) \wedge \mathbf{G}(\texttt{r1gather} \Rightarrow \mathbf{X}(\neg \texttt{r1gather} \mathcal{U} \texttt{r1upload})) \wedge \mathbf{G}(\texttt{r2gather} \Rightarrow \mathbf{X}(\neg \texttt{r2gather} \mathcal{U} \texttt{r2upload})) \wedge \mathbf{GF}\pi$$

where $\pi := \texttt{r1gather4} \wedge \texttt{r2gather2}$. Note that it is the sub-formula $\mathbf{G}(\texttt{gather} \Rightarrow (\texttt{r1gather4} \wedge \texttt{r2gather2}))$ in ϕ_4 that enforces the first robot to gather data at G4 and the second robot to gather data at G2. Figure 6(a) illustrates the optimal team trajectory we obtain for ϕ_4 using our robust approach. For this case, total computation time is 20 seconds and the cost is 24 with an upper bound of 26.4 seconds. After 10 iterations of this trajectory, maximum value of the cost observed in the field never exceeded 25.3 seconds.

Case study 5. In all of the case studies that we have considered so far, some of the data gathering locations have not been visited in order to optimize the team trajectory. Also, we have had the requirement that the robots must go to a dedicated upload region to upload their data before their next data gathering. However, in many cases, robots have uninterrupted links to their bases by means of some sort of wireless communication channel, and are not required to visit an upload location to upload their data. Now, we consider the case where the robots are required to visit all of the data gathering locations and are not required to visit an upload region before each data gathering. This can be expressed in LTL as

$$\phi_5 := \mathbf{GF}\texttt{gather1} \wedge \mathbf{GF}\texttt{gather2} \wedge \mathbf{GF}\texttt{gather3} \wedge \mathbf{GF}\texttt{gather4} \wedge \mathbf{GF}\pi$$

where the optimizing proposition is set as $\pi := \texttt{gather}$. Figure 6(b) illustrates the optimal team trajectory we obtain for ϕ_5. For this case, it takes 23.5 minutes for our robust approach to obtain this trajectory. The cost of the trajectory is 3, with an upper bound of 5.1 seconds. Since ϕ_5 is trace-closed, the robots synchronize only at the beginning of their suffix cycles. It is interesting to note that the optimal solution for this case is to have robot 2 repeatedly gather data at G4 while using robot 1 to visit the remaining data gathering locations. Here, the trajectory of robot 2 minimizes the cost by gathering data as frequently as possible whereas the trajectory of robot 1 satisfies the rest of mission specification by visiting the remaining data gathering locations.

7.2. Numerical case studies on scalability

In this section we investigate the scalability of our approach both in the number of robots and in the size of the environment considering a small patrolling example in an environment with nine regions. Figure 7 illustrates the TS that models the motion of the robots in a 3×3 grid environment,

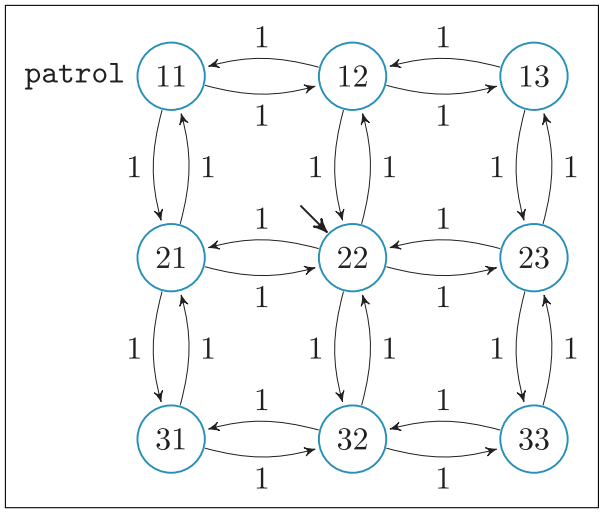

Fig. 7. The TS that models the motion of the robots in the 3×3 environment that we consider in our scalability experiments. The patrol proposition is defined at state 11 and the initial state is 22.

where the center region (state 22) is the initial state of the robots and the proposition patrol is assigned to the upper left region (state 11). We assume that the robots are identical to each other and can follow a given trajectory exactly, i.e. we use our exact solution given in Section 5. We consider the mission specification $\phi := \mathbf{GF}\pi$ where the optimizing task is $\pi := \texttt{patrol}$. For the case studies presented next, our implementation is run on an iMac i5 computer with 32 GB of RAM.

In order to evaluate the scalability of our approach in the number of robots, we run our implementation for increasing number of robots starting from two robots going up to five robots. A summary of these four case studies is presented in Table 3. Note that as we consider the same mission, the size of the Büchi automaton remains the same for all cases. The last column of Table 3 gives the ratio of total computation times between the cases with m and $m-1$ robots for $m = 3, 4, 5$, as 117, 186, and 197. On the other hand, the worst-case bounds on these values as given by Proposition 5.3 are 10,868 are 12,565, and 13,327. The state count of the team TS (second column in Table 3) also remains well below the worst-case bound of 9^m, $m = 2, 3, 4, 5$, given by Proposition 4.2. Thus, we see that for this example our approach scales better in the number of robots than the worst-case bounds.

Next, we evaluate the scalability of our approach in the size of the environment by considering two robots moving over grids of increasing size: 3×3, 5×5, 7×7, 9×9, 11×11, and 13×13. Each environment that we consider here is basically a bigger version of the 3×3 environment given in Figure 7, where the patrol proposition is defined at the upper left region and the initial state of each robot is the center of the grid. Table 4 gives a summary of these six case studies. The last column of Table 4 gives the ratio

Table 3. Quantitative information on the scalability of our approach in the number of robots. We assume that robots are identical and each one of them is modeled as given in Figure 7.

Number of robots	State count of the team tran. sys.	State count of the Büchi automaton	State count of the product automaton	Total computation time	Ratio to the previous case
2	41	2	50	0.07 s	—
3	189	2	250	8.2 s	117
4	881	2	1250	1530 s	186
5	4149	2	6250	301,734 s	197

Table 4. Quantitative information on the scalability of our approach in the size of the environment for two identical robots. Each 5×5 and larger environment is a bigger version of 3×3 grid given in Figure 7.

Environment size	State count of the team tran. sys.	State count of the Büchi automaton	State count of the product automaton	Total computation time	Ratio to the previous case
3×3	41	2	50	0.07 s	—
5×5	313	2	338	1 s	14
7×7	1201	2	1250	7.8 s	7.8
9×9	3281	2	3362	35.5 s	4.55
11×11	7321	2	7442	122.5 s	3.45
13×13	14,281	2	14,450	344.7 s	2.81

of total computation times between environments of size $n \times n$ and $(n-2) \times (n-2)$ for $n = 5, 7, 9, 11, 13$, as 14, 7.8, 4.55, 3.45, and 2.81. The worst-case bounds of these values as given by Proposition 5.3 are approximately 1222, 83, 25, 12.6, and 8. Thus, for this example, our algorithm scales better also in the size of the environment than the worst-case bounds.

These results suggest that, in practice, the computational complexity of our approach depends very much on the problem at hand and one can potentially observe much better running times and scalability (both in the number of robots and the size of the environment) than the worst-case analysis given in Proposition 5.3. Such differences in running times can be attributed to the mission specification, locations of the propositions, and connectivity between the states of the robot models under consideration.

8. Conclusions and future work

In this paper we have presented a method for automatic planning of optimal paths for a team of robots subject to temporal logic constraints. We have considered mission specifications expressed in LTL where an optimizing proposition must repeatedly be satisfied. We have provided an algorithm to model the asynchronous behavior of the team as a whole, which let us extend our previous work on single robot optimal path planning to multiple robots. The motion plan that our method provides is optimal in the sense that it minimizes the maximum time in between successive satisfying instances of the optimizing proposition. Our approach is general and robust enough to handle cases where the robots cannot follow planned trajectories exactly. If the traveling times observed in the field deviate from those given by the models of the robots, our method leverages the communication capabilities of the robots to guarantee that the mission specification is never violated while the overall communication effort is minimized. Our method also provides an upper bound on the difference between the performance in the field and the optimal performance in case of uncertain traveling times. We experimentally evaluate our approach and demonstrate its relevance in persistent surveillance missions in a road network environment

In order to be able to obtain a globally optimal team trajectory, our method constructs a relatively large model that captures all members of the team and the mission specification. Thus, the main drawback of this approach is its complexity. While the method presented in this paper can be extended to Markov decision processes (MDPs) and different cost functions, the most rewarding direction for future research seems likely to be in the area of distributed synthesis of optimal multi-robot path plans for general mission specifications.

Notes

1. Throughout the paper, we will denote TSs and automata with boldface letters, e.g. **T** and **B**. We use the double-barred letter $\mathbb{T}$ exclusively for referring to various time sequences that we define in this section, e.g. $\mathbb{T}_i$, $\mathbb{T}$, and $\mathbb{T}^\pi$.
2. LTL Optimal Multi-Agent Planner (LOMAP) Python Package is available at http://hyness.bu.edu/lomap/.
3. Amazon EC2 is a commercial cluster computing service available at http://aws.amazon.com/ec2/.

Funding

This work was supported in part by the Office of Naval Research (grant number MURI N00014-09-1051), Army Research Office

(grant number W911NF-09-1-0088), Air Force Office of Scientific Research (grant number YIP FA9550-09-1-020), National Science Foundation (grant number CNS-0834260), Singapore-MIT Alliance for Research and Technology (SMART) Future of Urban Mobility Project and by Natural Sciences and Engineering Research Council of Canada.

References

Baier C and Katoen JP (2008) *Principles of Model Checking*. Cambridge, MA: MIT Press.

Bianco A and de Alfaro L (1995) Model checking of probabilistic and nondeterministic systems. In: *Foundations of Software Technology and Theoretical Computer Science* (*Lecture Notes in Computer Science*, vol. 1026). Berlin: Springer, pp. 499–513.

Chen Y, Ding XC and Belta C (2011) Synthesis of distributed control and communication schemes from global LTL specifications. In: *2011 IEEE Conference on Decision and Control (CDC 2011)*, Orlando, FL, pp. 2718–2723.

Chen Y, Ding XC, Stefanescu A and Belta C (2012) A formal approach to the deployment of distributed robotic teams. *IEEE Transactions on Robotics* 28(1): 158–171.

Choset H, Lynch KM, Hutchinson S, et al. (2005) *Principles of Robot Motion - Theory, Algorithms, and Implementations*. Cambridge, MA: MIT Press.

Clarke EM, Peled D and Grumberg O (1999) *Model checking*. Cambridge, MA: MIT Press.

Ding XC, Smith SL, Belta C and Rus D (2011) MDP optimal control under temporal logic constraints. In: *2011 IEEE Conference on Decision and Control (CDC 2011)*, Orlando, FL, pp. 532–538.

Gastin P and Oddoux D (2001) Fast LTL to Büchi automata translation. In: *CAV '01 Proceedings of the 13th International Conference on Computer Aided Verification* (*Lecture Notes in Computer Science*, vol. 2102). Berlin: Springer, pp. 53–65.

Hagberg AA, Schult DA and Swart PJ (2008) Exploring network structure, dynamics, and function using NetworkX. In: *Proceedings of the 7th Python in Science Conference (SciPy2008)*, Pasadena, CA, pp. 11–15.

Hopcroft JE, Motwani R and Ullman JD (2007) *Introduction to Automata Theory, Languages, and Computation*. Reading, MA: Addison-Wesley.

Karaman S and Frazzoli E (2008a) Complex mission optimization for multiple-UAVs using linear temporal logic. In: *American Control Conference*, Seattle, WA, pp. 2003–2009.

Karaman S and Frazzoli E (2008b) Vehicle routing problem with metric temporal logic specifications. In: *IEEE Conference on Decision and Control*, Cancún, México, pp. 3953–3958.

Kavraki L, Svestka P, Latombe J and Overmars M (1996) Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 12(4): 566–580.

Kloetzer M and Belta C (2008) Dealing with non-determinism in symbolic control. In: Egerstedt M and Mishra B (eds.), *Hybrid Systems: Computation and Control: 11th International Workshop* (*Lecture Notes in Computer Science*, vol. 4981). Berlin: Springer, pp. 287–300.

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

Kress-Gazit H, Fainekos G and Pappas GJ (2007) Where's Waldo? Sensor-based temporal logic motion planning. In: *IEEE International Conference Robotics and Automation*, pp. 3116–3121.

Kress-Gazit H, Wongpiromsarn T and Topcu U (2011) Correct, reactive robot control from abstraction and temporal logic specifications. *IEEE Robotics & Automation Magazine* 18: 65–74.

Kuffner J and LaValle S (2000) Rrt-connect: An efficient approach to single-query path planning. In: *IEEE International Conference Robotics and Automation*, pp. 995–1001.

Kwiatkowska M, Norman G and Parker D (2002) Probabilistic symbolic model checking with PRISM: A hybrid approach. In: *International Journal on Software Tools for Technology Transfer* 6(2): 52–66.

LaValle SM (2006) *Planning Algorithms*. Cambridge: Cambridge University Press.

Lozano-Perez T (1983) Spatial planning: A configuration space approach. *IEEE Transactions on Computers* 32(2): 108–120.

Peled D, Wilke T and Wolper P (1998) An algorithmic approach for checking closure properties of temporal logic specifications and omega-regular languages. *Theoretical Computer Science* 195(2): 183–203.

Quottrup MM, Bak T and Izadi-Zamanabadi R (2004) Multi-robot motion planning: A timed automata approach. In: *IEEE International Conference Robotics and Automation*, New Orleans, LA, pp. 4417–4422.

Rimon E and Koditschek DE (1992) Exact robot navigation using artificial potential functions. *IEEE Transactions on Robotics and Automation* 8(5): 501–518.

Sistla AP and Clarke EM (1985) The complexity of propositional linear temporal logics. *Journal of the Association for Computing Machinery* 32(3): 733–749.

Smith SL, Tůmová J, Belta C and Rus D (2011) Optimal path planning for surveillance with temporal logic constraints. *The International Journal of Robotics Research* 30(14): 1695–1708.

Tabuada P and Pappas GJ (2006) Linear time logic control of discrete-time linear systems. *IEEE Transactions on Automatic Control* 51(12): 1862–1877.

Thomas W (2002) Infinite games and verification. In: *Proceedings of CAV*, pp. 58–64.

Toth P and Vigo D (eds.) (2001) *The Vehicle Routing Problem* (*Monographs on Discrete Mathematics and Applications*). Philadelphia, PA: SIAM.

Ulusoy A, Smith SL and Belta C (2012a) Optimal multi-robot path planning with LTL constraints: Guaranteeing correctness through synchronization. In: *International Symposium on Distributed and Autonomous Robotic Systems*, Baltimore, MD, USA.

Ulusoy A, Smith SL, Ding XC and Belta C (2012b) Robust multi-robot optimal path planning with temporal logic constraints. In: *IEEE International Conference Robotics and Automation*, St. Paul, MN, USA, pp. 4693–4698.

Ulusoy A, Smith SL, Ding XC, Belta C and Rus D (2011) Optimal multi-robot path planning with temporal logic constraints. In: *IEEE/RSJ International Conference Intelligent Robots and Systems*, San Francisco, CA, pp. 3087–3092.

Yordanov B, Tumova J, Cerna I, Barnat J and Belta C (2012) Temporal logic control of discrete-time piecewise affine systems. *IEEE Transactions on Automatic Control* 57(6): 1491–1504.

Appendix A: Generation of event sequences

In this appendix, we discuss how we generate the event sequences and corresponding sets of start states that we process in Algorithm 6 (Section 6.3). We start by recalling the definitions of the terms *position*, *robot–position pair*, *event*, and *event sequence* as defined in Section 6.3. We use the term *position* to refer to the current position of a robot in its run. If some robot i has just reached the state r_i^k in its run and satisfied the corresponding propositions after waiting for all of the robots in its wait-set $s_{i,wait}^k$ as given in Algorithm 5, then the position of the robot is k. If, on the other hand, robot i has left state r_i^{k-1}, but one of the above conditions has not yet been satisfied, then the position of the robot is $(k-1,k)$. A *robot–position* pair is a pair of the form (i,p) meaning that the position of robot i is p which can be either an integer or a pair of integers, as discussed above. For instance, the robot–position pair $(i,(k-1,k))$ means robot i is on its way from state r_i^{k-1} to state r_i^k. An *event* is a set of one or more robot–position pairs that give the new positions of the corresponding robots. In case of multiple robot–position pairs, all of these changes occur simultaneously. That is, the event $\{(i,k),(j,k)\}$ means that robots i and j have just reached position k in their runs. On the other hand, the event $\{(i,k)\}$ means that robot i has just reached position k and gives no information about the position of robot j. Finally, an *event sequence* is a list of events that occur sequentially.

Algorithm 6 relies on Algorithm 8 to construct the TS $\mathbf{W}$ that generates all possible words that can be observed in the field. Algorithm 8 is a loop (lines 3–25) that processes a dictionary called *tl*, short for timeline, which we construct using Algorithm 9 (line 1) presented later in this section. A dictionary is a data structure that comprises a set of keys, a set of values, and a function that maps each key to a value. In the case of *tl*, the keys are time intervals and the values are sets of robot–position pairs. Owing to non-deterministic traveling times, the time at which the robots reach their new positions in the field, in general, is not a single point but an interval. The dictionary *tl* captures this information by dividing the time from the beginning of the run until the end of the first suffix cycle to disjoint intervals and by associating a set of robot–position pairs with each interval. The set of robot–position pairs that corresponds to some interval in *tl* gives the new positions of the robots that can be achieved in that interval. In *tl*, the sets of robot–position pairs that correspond to different intervals are not guaranteed to be disjoint. Thus, new positions of the robots can span multiple intervals and can be reached in either one of the intervals that they span. Suppose that the sets of robot–position pairs $\{(1,1)\}$, $\{(1,1),(2,1)\}$, $\{(1,1)\}$ correspond to the intervals $[0.8,0.9)$, $[0.9,1.1]$, $(1.1,1.2]$, respectively. Then, robot 1 can reach position 1 in either one of the three intervals, whereas robot 2 can reach position 1 only in the interval $[0.9,1.1]$.

The first part of Algorithm 8 (lines 6–12) takes this fact into account while computing all possible position sequences that can be achieved by each robot at each interval. At lines 7–9, we first construct three sets of positions for each robot i: the set pos_{this} of positions that the robot can reach at this interval, the set pos_{prev} of positions that the robot can reach at either this interval or the previous interval, and the set pos_{next} of positions that the robot can reach

Algorithm 8: GENERATE-EVENT-SEQ

Input: $\mathbf{W}$, $\{r_1,\ldots,r_m\}$, $\{s_{1,wait},\ldots,s_{m,wait}\}$, $\{\mathbf{T}_1,\ldots,\mathbf{T}_m\}$, and $\underline{\rho_i}$, $\overline{\rho_i}$, $i=1,\ldots,m$.
Output: Yields a valid event sequence and the corresponding set of starting states.

1 Obtain dictionary *tl* using COMPUTE-TIMELINE (Algorithm 9).
2 ivs = Sorted list of intervals of *tl*, len_{ivs} = length of ivs.
3 **foreach** $l = 1 \ldots len_{ivs}$ **do**
4 $\quad all\_pos_{this} = tl[ivs[l]]$, $all\_pos_{prev} = \emptyset$, $all\_pos_{next} = \emptyset$, $robot\_seq$ = array of m empty sets.
5 $\quad$ **if** $l > 1$ **then** $all\_pos_{prev} = tl[ivs[l-1]]$, **if** $l < len_{ivs}$ **then** $all\_pos_{next} = tl[ivs[l+1]]$.
6 $\quad$ **foreach** $i \in \{1,\ldots,m\}$ **do**
7 $\quad\quad pos_{this} = \{p | (i,p) \in all\_pos_{this}\}$.
8 $\quad\quad pos_{prev} = \{p | (i,p) \in all\_pos_{prev} \cap all\_pos_{this}\} \cup \{\sim\}$.
9 $\quad\quad pos_{next} = \{p | (i,p) \in all\_pos_{next} \cap all\_pos_{this}\} \cup \{\sim\}$.
10 $\quad\quad$ **foreach** *tuple* $(prev, next)$ *in* $pos_{prev} \times pos_{next}$ **do**
11 $\quad\quad\quad pos'_{this} = \{p | p \in pos_{this}, (p > prev \vee prev = \sim), (p < next \vee next = \sim)\}$.
12 $\quad\quad\quad$ Sort pos'_{this} in ascending order and add to $robot\_seq[i]$.
13 $\quad$ Set $robot\_seq[i] = \{[]\}$ if $robot\_seq[i] = \emptyset \; \forall \; i \in \{1,\ldots,m\}$.
14 $\quad$ **foreach** *seq_tuple in* $robot\_seq[1] \times \ldots \times robot\_seq[m]$ **do**
15 $\quad\quad len_{seq}[i]$ = length of $seq\_tuple[i] \; \forall \; i \in \{1,\ldots,m\}$.
16 $\quad\quad max\_event\_cnt = \sum_{i=1}^{m} len_{seq}[i]$, all_perms = array of m empty sets.
17 $\quad\quad all\_perms[i]$ = all $len_{seq}[i]$ ordered combinations of $\{1,\ldots,max\_event\_cnt\} \; \forall \; i \in \{1,\ldots,m\}$.
18 $\quad\quad$ **foreach** *perm_tuple in* $all\_perms[1] \times \ldots \times all\_perms[m]$ **do**
19 $\quad\quad\quad event\_seq$ = array of max_event_cnt empty sets.
20 $\quad\quad\quad$ **foreach** $i \in \{1,\ldots,m\}$ **do**
21 $\quad\quad\quad\quad$ **foreach** $n \in \{1,\ldots,len_{seq}[i]\}$ **do**
22 $\quad\quad\quad\quad\quad$ Add event $(i, seq\_tuple[i][n])$ to $event\_seq[perm\_tuple[i][n]]$.
23 $\quad\quad\quad$ Remove those entries with $event\_seq[i] = \emptyset$ for $i \in \{1,\ldots,max\_event\_cnt\}$.
24 $\quad\quad\quad$ Define $start\_states$ as the set of states of $\mathbf{W}$ at which $event\_seq$ can start occurring.
25 $\quad\quad\quad$ Yield $(event\_seq, start\_states)$ after performing wait-set checks.

at either this interval or the next interval. Then, at line 10, we iterate over the elements of the product $pos_{prev} \times pos_{next}$. For each element $(prev, next)$ of this product set, we interpret *prev* as the last position that is reached in the previous interval and *next* as the first position that is reached in the next interval, and we obtain the remaining set of positions pos'_{this} to be reached at this interval as given in line 11. Then, we sort pos'_{this} in ascending order and add it to $robot\_seq[i]$, which gives the set of all possible position sequences that can be achieved by robot i at this interval. In a given interval, different robots can reach their new positions in any order with respect to each other, including simultaneously. The second part of Algorithm 8 (lines 14–25) addresses this by generating all possible event sequences that can be achieved by the robotic team. At line 14 we consider all combinations of position sequences that can be achieved by the robots by iterating over the elements of the product $robot\_seq[1] \times \ldots \times robot\_seq[m]$. An element *seq_tuple* of this set is an m-tuple of position sequences whose ith element is a position sequence that can be realized by robot i and $len_{seq}[i]$ (line 15) gives the length of this position sequence. Next, we define *max_event_cnt* as the maximum number of events that can occur in this interval, given by the case where the robots reach the positions in *seq_tuple* sequentially (line 16). In order to generate all possible event sequences, we use the variable *event_seq* to interpret the current interval as a box with *max_event_cnt* bins labeled $\{1, \ldots, max\_event\_cnt\}$ (line 19). For each robot i, we compute all $len_{seq}[i]$ ordered combinations of the sequence $\{1, \ldots, max\_event\_cnt\}$ (line 17) and iterate over the elements of the product $all\_perms[1] \times \ldots \times all\_perms[m]$ (line 18). Each element of this product set is a tuple that gives how the events of individual robots are ordered with respect to the events of the other robots. Next, we obtain the event sequence corresponding to each *perm_tuple* by placing the events of the robots into *event_seq* according to the positions given by the *perm_tuple* (lines 20–22). Note that, as events of different robots can occur simultaneously, we may end up with some empty *bins* in *event_seq*. We remove such empty entries of *event_seq* at line 23. Next, at line 24, we compute the set of start states of **W** at which *event_seq* can start occurring. Finally, at line 25 we yield the *event_seq* along with the corresponding set of start states after making sure that they do not violate the given wait-sets. At the next call, Algorithm 8 continues execution from line 18 with the next *perm_tuple*, then from line 14 with the next *seq_tuple*, and eventually from line 3 with the next interval. Once all of the intervals of *tl* are considered, Algorithm 8 terminates causing the loop that it is called in Algorithm 6 to terminate as well.

Proposition A.1. *Let $O(T)$ denote the time complexity of constructing the timeline tl and let I denote the number of intervals in tl. For the case where the intervals of the robots corresponding to different positions do not overlap, complexity of Algorithm 8 is $O(I\, 2^m\, m^{m+1} + T)$.*

Proof. It follows from our assumption that there is at most one robot–position pair per robot per interval. Then, complexity of the first part of the algorithm (lines 6–12) is $O(m)$, and the maximum values of $len_{seq}[i]$ and *max_event_cnt* are 1 and m. As $|all\_perms[i]|$ is at most m, the complexity of the inner loop at lines 18–25 becomes $O(m^{m+1})$. Since each $|robot\_seq[i]|$ is at most 2, the complexity of the second part of the algorithm (lines 14–25) is $O(2^m\, m^{m+1})$. As $O(m) < O(2^m\, m^{m+1})$, the complexity of Algorithm 8 for each interval considered at line 3 is also $O(2^m\, m^{m+1})$. After substituting I for the number of intervals and $O(T)$ for the time complexity of constructing *tl*, the overall complexity of Algorithm 8 becomes $O(I\, 2^m\, m^{m+1} + T)$.

Remark A.2. *In Proposition A.1 we assumed that the intervals of the robots corresponding to different positions do not overlap. Let t_n denote the planned time until the robots reach the nth position in their runs and K denote the total length of the prefix and the first suffix cycle. The above condition is satisfied when $\overline{\rho_i} t_{n-1} < \underline{\rho_j} t_n$ holds for all $i, j \in \{1, \ldots, m\}$ and $n = 1, \ldots, K-1$. This is typically the case where the deviation values of the robots are small enough (with respect to the length of the suffix cycle and durations between consecutive states in the run) such that the intervals in which the robots can reach different positions in their runs do not overlap. A more general complexity analysis could be performed for the case where robots move to different positions in a single interval, but at the cost of increased difficulty of presentation and interpretation. We employ the same assumption in Proposition A.3 for the same reason.*

We use Algorithm 9 to construct the dictionary *tl*, short for timeline, that we use in Algorithm 8. As discussed earlier, since the runs of the robots are periodic and the robots synchronize at the beginning of each suffix cycle, we consider only the prefix and the first suffix cycle of the runs of the robots during the construction of *tl*. The first part of Algorithm 9 (lines 1–7) computes the intervals in which the robots can reach the next positions in their runs. The interval in which robot i can reach position k is determined by the deviation values $\overline{\rho_i}$ and $\underline{\rho_i}$, the nominal time $w_i(r_i^{k-1}, r_i^k)$ it takes for the robot to reach r_i^k from its previous state r_i^{k-1}, wait-set $s_{i,wait}^k$ of the robot for position k, and the interval in which the robot can depart from its previous position. In Algorithm 9, we use $pos\_ivs[i][k].start$ and $pos\_ivs[i][k].end$ to denote the start and end points of the interval in which robot i can reach position k. As the robots start their runs in a synchronized way, we set the interval of the first positions of all robots to $[0, 0]$ at line 3. For all other positions, we first construct the set *waits_for* that includes both robot i itself and the robots that robot i has to wait for at that position (line 4). Next, at lines 5–6 we calculate the earliest and latest time that robot i can reach position k by using the models of the robots in the set *waits_for* and the intervals of their previous positions. Then, at line 7, we save the interval of robot–position pair (i, k) in the *pos_ivs* array.

Algorithm 9: COMPUTE-TIMELINE

Input: Individual runs $\{r_1, \ldots, r_m\}$, wait-sets $\{s_{1,wait}, \ldots, s_{m,wait}\}$, TSs $\{\mathbf{T}_1, \ldots, \mathbf{T}_m\}$, and deviation values $\overline{\rho_i}, \underline{\rho_i}$, $i = 1, \ldots, m$ of the robots.

Output: The dictionary *tl* of sets of robot–position pairs keyed by disjoint intervals.

1 **for** $k = 0, \ldots, end$ **do**
2 **for** $i = 1, \ldots, m$ **do**
3 **if** k *is* 0 **then** $pos\_ivs[i][k] = [0, 0]$ **else**
4 $waits\_for = \{i\} \cup s^k_{i,wait}$.
5 $earliest = \max_{j \in waits\_for}(pos\_ivs[j][k - 1].start + \underline{\rho_j} * w_j(r_j^{k-1}, r_j^k))$.
6 $latest = \max_{j \in waits\_for}(pos\_ivs[j][k - 1].end + \overline{\rho_j} * w_j(r_j^{k-1}, r_j^k))$.
7 $pos\_ivs[i][k].start = earliest$, $pos\_ivs[i][k].end = latest$.
8 **for** $k = 0, \ldots, end$ **do**
9 **for** $i = 1, \ldots, m$ **do**
10 $projection\_queue = \{pos\_ivs[i][k]\}$.
11 **foreach** $new\_iv \in projection\_queue$ **do**
12 $intersected = False$.
13 **foreach** $old\_iv \in tl$ **do**
14 int_iv is the intersection of new_iv and old_iv.
15 **if** *new_iv intersects with old_iv* **then**
16 $intersected = True$.
17 $tl[int\_iv] = tl[old\_iv] \cup \{(i, k)\}$.
18 **if** $old\_iv.start < new\_iv.start$ **then**
19 $tl[[old\_iv.start, new\_iv.start)] = tl[old\_iv]$.
20 Remove old_iv from tl.
21 **if** $old\_iv.end > new\_iv.end$ **then**
22 $tl[(new\_iv.end, old\_iv.end]] = tl[old\_iv]$.
23 Remove old_iv from tl.
24 **if** $new\_iv.start < old\_iv.start$ **then**
25 Add $[new\_iv.start, old\_iv.start)$ to *projection_queue*.
26 **if** $new\_iv.end > old\_iv.end$ **then**
27 Add $(old\_iv.end, new\_iv.end]$ to *projection_queue*.
28 **if** *intersected is False* **then**
 $tl[new\_iv] = \{(i, k)\}$
29 Return tl.

The second part of Algorithm 9 (lines 8–28), projects the intervals in *pos_ivs* to a common *timeline* by considering each position k of each robot i. The variable *tl* is a dictionary of sets of robot–position pairs keyed by intervals. To be able to use this dictionary by iterating over its keys as discussed earlier, we need to make sure that its keys, which are intervals, do not intersect with each other. To this end, we maintain the queue *projection_queue* to hold the remaining parts of the intersecting intervals that we may need to break up during the projection. We start the projection by adding the interval of the robot–position pair (i, k) to the projection queue. Then, for each interval *new_iv* in the projection queue, we check all of the intervals in *tl* to see whether any of them intersects with *new_iv*. If not, we add this interval *new_iv* to the timeline along with its set of robot–position pairs (line 28). If, on the other hand, the interval *new_iv* intersects with some interval *old_iv* in *tl*, we set the interval *int_iv* to be the intersection of *new_iv* and *old_iv* and add it to the timeline with the appropriate set of robot–position pairs (line 17). Next, at lines 18–27 we check to see whether we need to break the *old_iv* or *new_iv*. If *old_iv* extends beyond *new_iv* from the beginning or the end, we break it appropriately by defining a new entry for the extending parts and removing the old entry that corresponds to *old_iv* from *tl*. If, on the other hand, *new_iv* extends beyond *old_iv*, we do not add the extending parts to *tl* directly as they may intersect with other intervals already in *tl*. Instead, we add the extending parts of *new_iv* to the projection queue so that they are processed in the coming iterations. Algorithm 9 terminates once it processes all positions of all robots up to the end of the first suffix cycle of their runs.

Proposition A.3. *Let K denote the total length of the prefix and the first suffix cycle. For the case where the intervals of the robots corresponding to different positions do not overlap, complexity of Algorithm 9 is $O(m^2K^2)$.*

Proof. In the worst case, each robot waits for every other robot, thus computation of each $pos\_ivs[i][k]$ at lines 4–7 takes time $O(m)$. Then, the complexity of the first part of the algorithm (lines 1–7) is $O(m^2K)$. In the second part of the algorithm (lines 8–28), each projected interval may intersect with previously defined intervals resulting in up to two additional intervals per projection. Thus, we have $O(m)$ intervals for each position and $O(mK)$ intervals in total. Consequently, the loop at lines 11–28 executes $O(mK)$ times for each projection, and complexity of the second part of the algorithm (lines 8–28) becomes $O(m^2K^2)$. Thus, the overall complexity of Algorithm 9 is $O(m^2K^2)$.

Appendix B: Index to Multimedia Extensions

The multimedia extension page is found at http://www.ijrr.org

Table of Multimedia Extensions

Extension	Type	Description
1	Video	Execution of the trajectory in case study 3.