Optimal multi-robot path planning with temporal logic constraints

Citation: Ulusoy, Alphan et al. "Optimal Multi-robot Path Planning with Temporal Logic Constraints." IEEE/RSJ International Conference on Intelligent Robots and Systems 2011 (IROS). 3087–3092.

As Published: http://dx.doi.org/10.1109/IROS.2011.6094884

Publisher: Institute of Electrical and Electronics Engineers (IEEE)

Persistent URL: http://hdl.handle.net/1721.1/72507

Version: Author's final manuscript: final author's manuscript post peer review, without publisher's formatting or copy editing

Optimal Multi-Robot Path Planning with Temporal Logic Constraints

Alphan Ulusoy† Stephen L. Smith⋆ Xu Chu Ding† Calin Belta† Daniela Rus‡

***Abstract*— In this paper we present a method for automatically planning optimal paths for a group of robots that satisfy a common high level mission specification. Each robot's motion in the environment is modeled as a weighted transition system. The mission is given as a Linear Temporal Logic formula. In addition, an optimizing proposition must repeatedly be satisfied. The goal is to minimize the maximum time between satisfying instances of the optimizing proposition. Our method is guaranteed to compute an optimal set of robot paths. We utilize a timed automaton representation in order to capture the relative position of the robots in the environment. We then obtain a bisimulation of this timed automaton as a finite transition system that captures the joint behavior of the robots and apply our earlier algorithm for the single robot case to optimize the group motion. We present a simulation of a persistent monitoring task in a road network environment.**

I. Introduction

Recently there has been an increased interest in using temporal logics to specify mission plans for robots [1], [2], [3], [4], [5]. Temporal logics are appealing because they provide a formal high level language in which to describe a complex mission. In addition, tools from model checking [6], [7] can be used to generate a robot path satisfying the specification, if such a path exists. However, frequently there are multiple robot paths that satisfy a given specification. In this case, one would like to choose the *optimal* path according to a cost function. The current tools from model checking do not provide a method for doing this. In our previous work [8] we considered Linear Temporal Logic (LTL) specifications, and a particular form of cost function, and provided a method for computing optimal robot paths for one robot. In this paper we extend this result to multiple robots.

For simplicity of presentation, we assume that each robot moves among the vertices of an environment modeled as a graph. However, by using feedback controllers for facet reachability and invariance in polytopes [9], [10] the method developed in this paper can be easily applied for motion planning and control of robots with "realistic" continuous dynamics (*e.g.*, unicycle) traversing an environment partitioned using popular partitioning schemes such as triangulations and rectangular partitions.

The main difficulty in moving from a single robot to multiple robots is in synchronizing the motion of the robots, or in allowing the robots to move asynchronously. In [11], the authors propose a method for decentralized motion of multiple robots subject to LTL specifications. In their approach, the robots take transitions (*i.e.*, travel along edges in the graph) synchronously. Once every robot has completed a transition, the robots can synchronously make the next transition. While such an approach is effective for satisfying the LTL formula, it does not lend itself to optimizing the robot motion, since time is spent "waiting" for other robots. In [12], the authors take a different approach, representing the motion of the robots in the environment as a timed automaton. Each robot then has a continuous clock variable that describes its progress along a transition (*i.e.*, a robot's position along an edge between two vertices). The authors then look at satisfying specifications given in computational tree logic (CTL). In this paper, we utilize a similar timed-automaton representation. However, we consider LTL specifications, for which the control synthesis problem is fundamentally different. In addition, rather than just satisfying the formulas, we optimize the motion of the robots.

In terms of optimizing paths, the most closely related work has been on the vehicle routing problem (VRP) [13]. Recent results [14], [15] present extensions of vehicle routing problems to more general classes of temporal constraints. In [15], the authors consider vehicle routing with metric temporal logic specifications. The goal is to minimize a cost function of the vehicle paths (such as total distance traveled). The authors present a method for computing an optimal set of paths by converting the problem to a mixed integer linear program (MILP). While the approach is computationally intensive, it has been used to solve problems of real-world significance. However, their method does not apply to the persistent monitoring and data gathering applications that are of interest in this paper. In particular, it does not allow for specifications of the form "always eventually," which appear when specifying that a robot should repeatedly perform a task. In this paper we take an entirely different approach to optimizing robot motion, resulting in an optimization problem on a graph, rather than a MILP.

The contribution of this paper is to present a method for generating optimal paths for a group of robots satisfying general LTL formulas. We focus on minimizing a cost function that captures the maximum time between satisfying instances of an *optimizing proposition*. The cost is motivated by problems in persistent monitoring and in pickup and delivery problems. Our solution relies on describing the motion of the group of robots in the environment as a timed automaton. This description allows us to represent the relative position between robots. Such information is necessary for optimizing the robot motion. We provide a bisimulation [16] of the

This work was supported in part by ONR-MURI N00014-09-1051, ARO W911NF-09-1-0088, AFOSR YIP FA9550-09-1-020, and NSF CNS-0834260.

† Hybrid and Networked Systems Laboratory, Boston University, Boston, MA 02215 (alphan@bu.edu, xcding@bu.edu, cbelta@bu.edu)

⋆ Dept. of Electrical and Computer Engineering, University of Waterloo, Waterloo ON, N2L 3G1 Canada (stephen.smith@uwaterloo.ca)

‡ Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139 (rus@csail.mit.edu)

infinite-dimensional timed automaton to a finite dimensional transition system. From this point we are able to apply our previous results [8] to compute an optimal run. This run then maps to a path for each robot. We provide simulation results for robots in a road network environment. The main hurdle in our approach is the computational complexity. We discuss ways in which this can be reduced, and show that fairly complex problems can be solved under this framework.

The organization of the paper is as follows. In Section II, we give some preliminaries. In Section III, we formally state the motion planning problem for a team of robots, and in Section IV we present our solution. In Section V we present an experimental case study for a team of robots performing persistent data gathering missions in a road network environment. Finally, in Section VI, we discuss some promising future directions.

II. Preliminaries

A. Transition Systems and LTL

Definition II.1 (**Transition Systems**). *A (weighted) transition system (TS) is a tuple* $\mathbf{T} := (\mathcal{Q}_T, q_T^0, \rightarrow_T, \Pi, \mathcal{L}_T, w_T)$*, where*

(i) $\mathcal{Q}_T$ *is a finite set of states,*
(ii) $q_T^0 \in \mathcal{Q}_T$ *is the initial state,*
(iii) $\rightarrow_T \subseteq \mathcal{Q}_T \times \mathcal{Q}_T$ *is the transition relation,*
(iv) Π *is a finite set of atomic propositions (observations),*
(v) $\mathcal{L}_T : \mathcal{Q}_T \rightarrow 2^\Pi$ *is a map giving the set of all atomic propositions satisfied in a state, and*
(vi) $w_T : \rightarrow_T \rightarrow \mathbb{R}^+$ *is a map that assigns a positive weight to each transition.*

We define a run of $\mathbf{T}$ as an infinite sequence of states $r_T = q^0 q^1 \dots$ such that $q^0 = q_T^0$, $q^k \in \mathcal{Q}_T$ and $(q^k, q^{k+1}) \in \rightarrow_T$ for all $k \geq 1$. A run generates an infinite word $\omega_T = \mathcal{L}(q^0)\mathcal{L}(q^1)\dots$ where $\mathcal{L}(q^k)$ is the set of atomic propositions satisfied at state q^k.

Definition II.2 (Formula of LTL). *An LTL formula* ϕ *over the atomic propositions* Π *is defined inductively as follows:*

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

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

LTL formulas are interpreted over infinite words (generated by the transition system $\mathbf{T}$ from Def. II.1). Informally, $\mathbf{X}\,\alpha$ states that at the next state of a word, proposition α is true; and $\alpha_1\,\mathcal{U}\,\alpha_2$ states that there is a future moment when proposition α_2 is true, and proposition α_1 is true at least until α_2 is true. From these temporal operators we can construct two other temporal operators: Eventually (i.e., future), $\mathbf{F}$ defined as $\mathbf{F}\,\phi := \top\,\mathcal{U}\,\phi$, and Always (i.e., globally), $\mathbf{G}$, defined as $\mathbf{G}\,\phi := \neg\,\mathbf{F}\,\neg\,\phi$. The formula $\mathbf{G}\,\phi$ states that ϕ is true at all positions of the word; the 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.

Definition II.3 (Büchi Automaton). *A Büchi automaton is a tuple* $\mathbf{B} := (\mathcal{S}, \mathcal{S}_0, \Sigma, \delta, \mathcal{F})$*, consisting of (i) a finite set of states* $\mathcal{S}$*; (ii) a set of initial states* $\mathcal{S}_0 \subseteq \mathcal{S}$*; (iii) an input alphabet* Σ*; (iv) a non-deterministic transition relation* $\delta \subseteq \mathcal{S} \times \Sigma \times \mathcal{S}$*; (v) a set of accepting (final) states* $\mathcal{F} \subseteq \mathcal{S}$*.*

A *run* of the Büchi automaton over an input word $\omega = \omega^0\omega^1\dots$ is a sequence $r_B = s^0 s^1 \dots$, such that $s^0 \in \mathcal{S}_0$, and $(s^k, \omega^k, s^{k+1}) \in \delta$, for all $k \geq 1$. A Büchi automaton accepts a word over Σ if at least one of the corresponding runs intersects with $\mathcal{F}_B$ infinitely many times. For any LTL formula ϕ over Π, one can construct a Büchi automaton with input alphabet $\Sigma \subseteq 2^\Pi$ accepting all and only words over Π that satisfy ϕ. We refer readers to [17] and references therein for efficient algorithms and freely downloadable implementations to translate a LTL formula over Π to a corresponding Büchi automaton.

B. Timed Automata

A *clock* is a real-valued variable that increases at a rate of one as time progresses. Clocks may be valuated, or reset to zero. Let $\mathcal{C}$ denote a set of clocks. A *clock valuation* of some clock $x \in \mathcal{C}$, denoted as $v(x)$, is a mapping from $\mathcal{C}$ to $\mathbb{R}_{\geq 0}$ that assigns a real value to each clock. A *clock constraint* g over a set of clocks $\mathcal{C}$ is formed according to the grammar

$$g ::= x < c \;\big|\; x \leq c \;\big|\; x > c \;\big|\; x \geq c \;\big|\; g \wedge g,$$

where $c \in \mathbb{N}$ is a constant and $x \in \mathcal{C}$ is a clock. We let $\mathcal{G}$ denote the set of all clock constraints over $\mathcal{C}$. A *clock valuation* $v(x)$ of some clock x satisfies a clock constraint g at some time iff g evaluates to true for $v(x)$.

Definition II.4 (**Timed Automata**). *A timed automaton is a tuple* $\mathbf{A} := (\mathcal{Q}_A, q_A^0, \mathcal{C}_A, \mathcal{G}_A, \rightarrow_A, \Pi, \mathcal{L}_A)$ *where*

(i) $\mathcal{Q}_A$ *is a finite set of states,*
(ii) $q_A^0 \in \mathcal{Q}_A$ *is an initial state,*
(iii) $\mathcal{C}_A$ *is a finite set of clocks,*
(iv) $\mathcal{G}_A$ *is a finite set of clock constraints over* $\mathcal{C}_A$*,*
(v) $\rightarrow_A \subseteq \mathcal{Q}_A \times \mathcal{G}_A \times 2^{\mathcal{C}_A} \times \mathcal{Q}_A$ *is the transition relation. A transition is a tuple* (q, g, c, q') *where* q *is the source state,* q' *is the destination state,* g *is the clock constraint that enables the transition, and* $c \subseteq \mathcal{C}_A$ *is the clock-resets, which is the set of clocks to be reset right after the transition.*
(vi) Π *is a finite set of atomic propositions, and*
(vii) $\mathcal{L}_A$ *is a map assigning a subset of* Π *to each transition of* $\rightarrow_A$*.*

The semantics of the timed automaton can be understood as follows: starting from the initial state q_A^0, the values of all clocks increase at rate one, and the system remains at this state until a clock constraint corresponding to an outgoing transition is satisfied. When this happens, the transition is immediately taken and the clocks in the clock-resets are reset. The timed automaton from Def. II.4 can be seen as a particular case of the timed automaton defined in [18], which also allows for clock invariants associated with states.

A timed automaton, as defined in Def. II.4, has a finite set of clock regions $\mathcal{R}_A$, which is the set of equivalence classes of clock valuations induced by its clock constraints $\mathcal{G}_A$. Intuitively, a clock region $r \in \mathcal{R}_A$ is a subset of the infinite set of all clock valuations of $\mathcal{C}_A$, in which all clock valuations are equivalent in the sense that the future behavior of the system is the same. In [18], it has been shown that a clock region can be either a corner point (*e.g.*, (0,1)), an open line-segment (*e.g.*, $0 \leq x_1 = x_2 \leq 1$), or an open region (*e.g.*, $0 \leq x_1 \leq x_2 \leq 1$). The clock regions $\mathcal{R}_A$ of a timed automaton $\mathbf{A}$ induce an equivalence relation $\sim_A$ over its state space, and a simulation quotient, which we refer to as the region automaton $\mathbf{R} = \mathbf{A}/\sim_A$. The region automaton $\mathbf{R}$ induced by this equivalence relation is a bisimulation quotient. To define $\mathbf{R}$, we define a clock region r'' to be the *time-successor* of a clock region r if and only if there is a $t > 0$ such that all possible clock valuations in r are in clock region r'' after time t.

Definition II.5 (Region Automata). *The region automaton $\mathbf{R}$ of a timed automaton $\mathbf{A}$ (Def. II.4) is a tuple $\mathbf{R} := (\mathcal{Q}_R, q_R^0, \rightarrow_R)$, where*

(i) *$\mathcal{Q}_R$ is the set of states of the form $\{q, r\}$ such that $q \in \mathcal{Q}_A$ and $r \in \mathcal{R}_A$,*

(ii) *q_R^0 is the initial state of the form $\{q_A^0, r^0\}$ such that q_A^0 is the initial state of $\mathbf{A}$ and all clock valuations of r^0 are zero, i.e., $x_i = 0 \ \forall \ x_i \in r^0$,*

(iii) *$\rightarrow_R$ is the transition relation such that there is a transition from $\{q, r\}$ to $\{q', r'\}$ if and only if there is a transition from q to q' in $\mathbf{A}$ and a clock constraint g in $\mathcal{G}_A$ and a clock region r'' such that:*

  (a) *r'' is a time-successor of r,*

  (b) *r'' satisfies the clock constraint g, and*

  (c) *r'' goes to r' when corresponding clocks are reset once g is satisfied and the transition is made.*

III. Problem Formulation and Approach

Let

$$\mathcal{E} = (V, \rightarrow_{\mathcal{E}}) \tag{1}$$

be a graph of the environment, where V is the set of vertices and $\rightarrow_{\mathcal{E}} \subseteq V \times V$ is a relation modeling the set of edges. In practice, $\mathcal{E}$ can be 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 roads, intersections, and buildings in an urban-like environment and $\rightarrow_{\mathcal{E}}$ can show how these are connected (see Fig. 5).

Consider a team of m robots moving in an environment modeled by $\mathcal{E}$. The motion capabilities of robot $i = \{1, \ldots, m\}$ can be represented by a transition system (see Def. II.1)

$$\mathbf{T}_i = (\mathcal{Q}_i, q_i^0, \rightarrow_i, \Pi, \mathcal{L}_i, w_i), \tag{2}$$

where $\mathcal{Q}_i \subseteq V$; q_i^0 is the initial vertex of robot i; $\rightarrow_i \subseteq \rightarrow_{\mathcal{E}}$ is a relation modeling the capability of robot i to move among the vertices; Π is a set of propositions assigned to the environment, which are assigned by $\mathcal{L}_i$ to robot i; $w_i(q, q')$ captures the time for robot i to go from vertex q to q', and we assume that $w_i(q, q')$ is always an integer. In this robotic model, robot i travels along the edges of $\mathbf{T}_i$, and spends zero time on the vertices. Note that we allow the assignment of propositions to differ for different robots to capture the possibly different capabilities of the robots to satisfy propositions in the environment. Also, in the definition of transition systems, each transition is deterministic, so any run on $\mathbf{T}_i$ can always be followed by robot i.

We assume that there is an atomic proposition $\pi \in \Pi$, called the *optimizing proposition*. We consider LTL formulas of the form

$$\phi := \varphi \wedge \mathbf{G}\,\mathbf{F}\,\pi, \tag{3}$$

where φ can be any LTL formula over Π, and $\mathbf{G}\,\mathbf{F}\,\pi$ specifies that proposition π must be satisfied infinitely often. In a persistent data gathering task, π can be assigned to regions where new data is gathered, while φ could be used to specify rules (such as traffic rules) that must be obeyed at all times during the task.

We assume that each run $r_i = q_i^0 q_i^1 \ldots$ of a $\mathbf{T}_i$ (robot i) starts at $t = 0$ and generates a word $\omega_i = \omega_i^0 \omega_i^1 \ldots$ and an infinite 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. In order to define the behavior of the team as a whole, we consider the sequences $\mathbb{T}_i$ as sets and take the union $\bigcup_{i=1}^m \mathbb{T}_i$ and order this set in an ascending order to obtain $\mathbb{T} := t^0 t^1, \ldots$. Then, we define $\omega = \omega^0 \omega^1 \ldots$ to be the word generated by the team of robots where ω^k is the union of all propositions satisfied at t^k. Finally, 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 k^{th} time by the team. We can now formulate the problem:

Problem III.1. Given *a team of robots modeled as transitions systems $\mathbf{T}_i$ and an LTL formula ϕ in the form* (3)*;* **Synthesize** *a run r_i for each robot in the team such that the word generated by the team satisfies ϕ and $\mathbb{T}^\pi$ minimizes*

$$J(\mathbb{T}^\pi) = \limsup_{i \to +\infty} \left(\mathbb{T}^\pi(i+1) - \mathbb{T}^\pi(i) \right). \tag{4}$$

Note that a solution to Prob. III.1 minimizes the maximum time between satisfying instances of π. Since we consider LTL formulas containing $\mathbf{GF}\pi$, this optimization problem is always well-posed. For the data gathering task previously mentioned, this translates to minimizing the maximum time in between two data gatherings.

Our solution to Problem III.1 can be outlined as follows:

(i) For each transition system $\mathbf{T}_i, i = 1, \ldots, m$, we obtain the dual transition system $\mathbf{D}_i$ where states and transitions are swapped and propositions are assigned to the transitions (Sec. IV-A);

(ii) For each dual transition system $\mathbf{D}_i$, we obtain a corresponding timed automaton $\mathbf{A}_i$. Each timed automaton consists of a single clock, which keeps track of the amount of time that a robot has been traveling between states in the original transition system $\mathbf{T}_i$ and we create a product timed automaton $\mathbf{P}$ as the parallel composition of $\mathbf{A}_i, i = 1, \ldots, m$ timed automata (Sec. IV-B);

(iii) We obtain the region automaton $\mathbf{R}$ as the bisimulation quotient of $\mathbf{P}$ (Sec. IV-C);
(iv) We find the optimal run on $\mathbf{R}$ using the OPTIMAL-RUN algorithm we previously developed in [8]. We project this run back to the individual $\mathbf{T}_i, i = 1, \ldots, m$ to obtain the solution to Prob. III.1 (Sec. IV-D).

IV. PROBLEM SOLUTION

In this section, we explain each step of the solution to Prob. III.1 in detail. For illustration, we use a simple example throughout this section involving two robots in an environment consisting of three vertices. We present a multi-robot scenario in a more realistic setting in Sec. V.

A. Dual Transition Systems

We proceed by converting the transition system $\mathbf{T}_i$ for each robot to a dual transition system $\mathbf{D}_i$. The dual $\mathbf{D}$ of a transition system $\mathbf{T}$ is obtained by swapping its states with its transitions. More precisely, given $\mathbf{T} = (\mathcal{Q}_T, q_T^0, \rightarrow_T, \Pi, \mathcal{L}_T, w_T)$, we define $\mathbf{D} = (\mathcal{Q}_D, q_D^0 \rightarrow_D, \Pi, \mathcal{L}_D, w_D)$ as follows: if $(a, b) \in \rightarrow_T$, then $ab \in \mathcal{Q}_D$, and $(ab, bc) \in \rightarrow_D$. Intuitively, this means that the robot can "go from a to c through b." As propositions are originally assigned to the states of $\mathbf{T}$, they are satisfied on the transitions of $\mathbf{D}$, *i.e.*, if $(ab, bc) \in \rightarrow_D$, then $\mathcal{L}_D((ab, bc)) = \mathcal{L}_T(b)$. In addition, weights assigned to transitions of $\mathbf{T}$ are now defined on states of $\mathbf{D}$, *i.e.*, $w_D(ab) = w_T(a, b)$. This means that in the dual $\mathbf{D}_i$ of a $\mathbf{T}_i$ time is spent on the vertices and transitions are instantaneous. Since the initial state q_T^0 of $\mathbf{T}$ can have multiple outgoing transitions, the initial state q_D^0 does not correspond to any transitions, therefore it has zero weight, but it connects to all outgoing transitions of q_T^0. The duals of two simple transition systems are shown in Fig. 1.

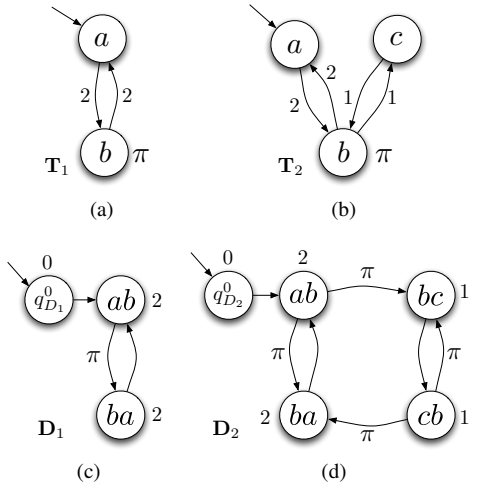


Fig. 1: (a) and (b) show the transition systems $\mathbf{T}_1$ and $\mathbf{T}_2$ for two robots in an environment with three vertices. The states of the transition systems correspond to vertices $\{a, b, c\}$ and the edges represent the motion capabilities of each robot. The weights of the edges represent the time needed to traverse from a state to another; (c) and (d) are the dual transition systems $\mathbf{D}_1$ and $\mathbf{D}_2$ corresponding to $\mathbf{T}_1$ and $\mathbf{T}_2$, respectively. A state labelled ab means that the robot is travelling from vertex a to b.

B. Construction of the timed automata

By constructing the duals of the original transition systems of individual robots, we can now fully capture the evolution of time for each robot taking transitions on $\mathbf{T}_i$ with a timed automaton as defined in Def. II.4. We can then generate a product timed automaton capturing the time evolution of the whole team.

To this end, for each robot, we define a clock x_i, which records how much time has passed in each state of $\mathbf{D}_i$. We interpret the weights on the states of $\mathbf{D}_i$ as clock constraints, *i.e.*, each state ab in $\mathbf{D}_i$ is associated with a clock constraint $v(x_i) \geq w_T(a, b)$. We set the initial value of the clock for each robot to 0, and we let the clock constraint for the initial state of $\mathbf{D}_i$ to be immediately satisfied. At each state, once the clock constraint is satisfied, it triggers an outgoing transition and clock x_i is reset to 0. As mentioned before in Def. II.4, we enforce a transition when a clock constraint is satisfied. We denote the timed automaton corresponding to $\mathbf{D}_i$ as $\mathbf{A}_i$. The timed automata corresponding to the $\mathbf{D}_i$'s in Fig. 1 are illustrated in Fig. 2.

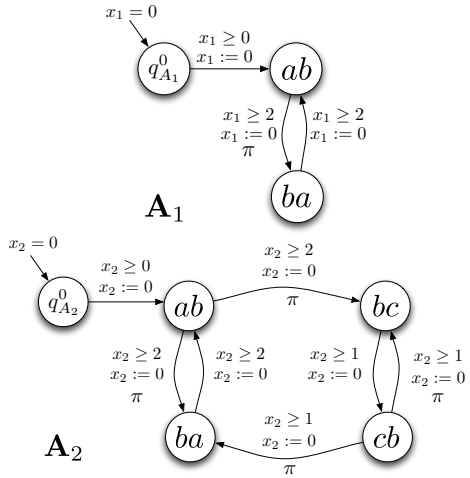


Fig. 2: Timed automata $\mathbf{A}_1$ and $\mathbf{A}_2$ of each robot, corresponding to $\mathbf{D}_1$ and $\mathbf{D}_2$ shown in Fig. 1c and Fig. 1d, respectively. The equations next to each arrow represents the clock constraint and the clock-reset associated with each transition of the timed automaton.

We capture the joint behavior of the robots by taking the parallel composition of the individual timed automata $\mathbf{A}_i$, $i = 1, \ldots, m$, and calling it the product timed automata $\mathbf{P}$. The set of states of $\mathbf{P}$ is the Cartesian product of the set of states of $\mathbf{D}_i$, $i \in \{1, \ldots, m\}$. The initial state of $\mathbf{P}$ is $(q_{D_1}^0, \ldots, q_{D_m}^0)$. We enable a transition from state $(q_1, \ldots, q_m)$ to $(q_1', \ldots, q_m')$ if and only if, for all i, either $(q_i, g_i, c_i, q_i') \in \rightarrow_{A_i}$, or if $(q_i, g_i, c_i, q_i') \notin \rightarrow_{A_i}$ for some i, then $q_i = q_i'$. We label this transition with the union of propositions satisfied by the corresponding transitions in $\rightarrow_{D_i}$, and similarly the clock constraints that enable this transition are the union of all clock constraints g_i associated with the transitions that are taken and inverses of the clock constraints associated with the remaining transitions that are not taken. Moreover, the clocks are reset for all robots i that transitioned to a new state q_i'. We require that at least one robot i makes a transition to a new state for each transition of

$\mathbf{P}$. Since we enforce each transition to be taken immediately when all clock constraints are satisfied, some transitions of $\mathbf{P}$ may never be taken because they are always preceded by some other transitions for all possible clock values. Such transitions will be referred to as invalid transitions. For the example given in Fig. 1 and Fig. 2, we show the resulting product timed automaton $\mathbf{P} = \mathbf{A}_1 \times \mathbf{A}_2$ in Fig. 3 (without invalid transitions).

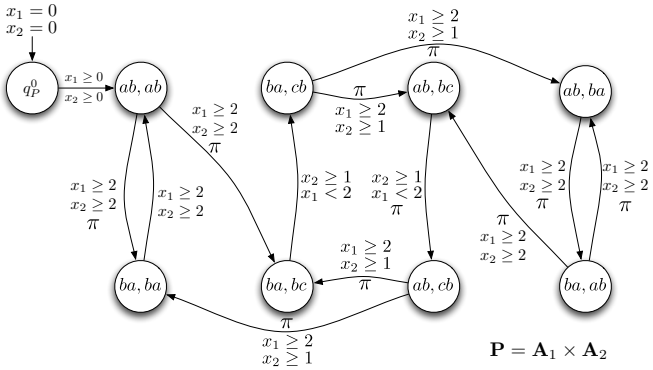


Fig. 3: The product timed automaton $\mathbf{P}$ describing the motion of the two robots. The state ab, ba denotes that $\mathbf{A}_1$ is in state ab and $\mathbf{A}_2$ is in state ba. To avoid notation clustering, we do not show the clock-resets and invalid transitions. For example, in the transition from state (ba, bc) to (ba, cb), robot 2 completes a transition, so its clock is reset, while robot 1 does not complete a transition, the state stays the same and the clock is not reset. The transition from (ba, bc) to (ab, bc) is invalid, because it can never happen before the transition from (ba, bc) to (ba, cb).

C. Construction of the Region Automaton

From the product timed automaton $\mathbf{P}$, we can obtain the *region automaton* $\mathbf{R}$ as a bisimulation quotient of $\mathbf{P}$ (see Sec. II-B). Note that the bisimulation quotient we obtain from $\mathbf{P}$ is a particular case of the bisimulation quotient of a general timed automaton, where the transitions are enforced when clock constraints are satisfied. In the process of obtaining $\mathbf{R}$, all invalid transitions of $\mathbf{P}$ are automatically removed, by the definition of region automata.

We can now assign propositions and weights to $\mathbf{R}$, converting it to a transition system as defined in Def. II.1. We define a function $\mathcal{L}_R : \mathcal{Q}_R \to 2^{\Pi}$ such that, for each transition $(\{q, r\}, \{q', r'\})$, the set of propositions corresponding to the transition (q, g, c, q') on $\mathbf{P}$ are assigned to the state q', *i.e.*, observations defined on the transitions of $\mathbf{P}$ are carried to their destination states in $\mathbf{R}$. In the following, we take m to be the number of clocks, or equivalently the number of robots, in the product timed automaton $\mathbf{P}$, and d_i to be the largest integer constant that some clock $x_i \in \mathcal{C}_P = \{x_1, \ldots, x_m\}$ is compared with.

Proposition IV.1. *For each state $\{q, r\}$ of the region automaton $\mathbf{R}$, clock region r is always a tuple $(v(x_1), \ldots, v(x_m))$, where $v(x_i)$ are integers for all $i = 1, \ldots, m$.*

Proof. Clock constraints are positive integers smaller than or equal to d_i. Since the transitions are enforced when clock constraints are satisfied, and the initial clock is set to 0, every time a transition on P is taken, after the clock-resets, we have $v(x_i) \in \{0, \ldots, d_i - 1\}$, for all $i = 1, \ldots, m$. Therefore, the set of clock regions that can be reached on $\mathbf{R}$ (the bisimulation quotient of $\mathbf{P}$) are always corner points, *i.e.*, a tuple $(v(x_1), \ldots, v(x_m))$, where $v(x_i)$ are integers for all $i = 1, \ldots, m$. ■

Using Prop. IV.1, we now assign a weight to each transition of $\mathbf{R}$. Given a transition $(\{q, r\}, \{q', r'\})$, we define its weight to be the time t it takes to reach from $r = (v(x_1), \ldots, v(x_m))$ to $r'' = (v(x_1) + t, \ldots, v(x_m) + t)$, where r'' is a time-successor of r. The region automaton corresponding to the product automaton from Fig. 3 is shown in Fig. 4.

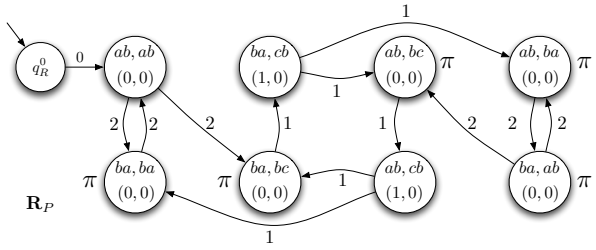


Fig. 4: The finite state region automaton capturing the joint behavior of two robots in 9 states. In the circle representing a state $\{q, r\}$, the first line is q and the second line is r.

The following proposition gives the bound on the size of the region automaton $\mathbf{R}$.

Proposition IV.2. *The number of states $|\mathcal{Q}_R|$ of $\mathbf{R}$ is bounded by*

$$|\mathcal{Q}_P| \left(\prod_{i=1}^{m} d_i - \prod_{i=1}^{m} (d_i - 1) \right) \tag{5}$$

Proof. From Prop. IV.1, all clock regions of $\mathbf{R}$ are corner points, *i.e.*, tuples of integers taking values within the range $\{0, \ldots, d_i - 1\}$. Counting the number of possible reachable clock regions, we have

$$\prod_{i=1}^{m} d_i - \prod_{i=1}^{m} (d_i - 1)$$

where $\prod_{i=1}^{m} d_i$ is the number of all possible corner points and $\prod_{i=1}^{m} (d_i - 1)$ is the number of corner points where all clocks are non-zero (since one clock must be zero after the reset, these corner points cannot be reached). Given a product timed automaton with $|\mathcal{Q}_P|$ number of states, using the above given bound on the number of reachable clock regions we can conclude that $|\mathcal{Q}_R|$ is bounded by (5). ■

Remark IV.3. *In [18] the authors give the upper-bound on the number of clock regions $|\mathcal{R}_P|$ of $\mathbf{P}$ as*

$$m! \cdot 2^m \cdot \prod_{i=1}^{m} (2d_i + 2),$$

which gives the upper bound of $\mathbf{R}$ as $|\mathcal{Q}_P| \cdot m! \cdot 2^m \cdot \prod_{i=1}^{m} (2d_i + 2)$. From Prop. IV.1, using our particular case of timed automata, $|\mathcal{Q}_R|$ is reduced by at least a factor of $m! \cdot 2^{2m}$.

We use Alg. 1 to obtain the region automaton $\mathbf{R}$, by applying a (recursive) depth-first search (DFS) on $\mathbf{P}$. We note that line 8 in Alg. 1 removes all invalid transitions in $\mathbf{P}$. Moreover, Alg. 1 generates $\mathbf{R}$ by finding all reachable clock regions of $\mathbf{P}$.

Algorithm 1: OBTAIN-REGION-AUTOMATON

Input: Product timed automaton $\mathbf{P}$.
Output: Corresponding region automaton $\mathbf{R}$.

```
1  begin
2      Obtain R by running a DFS on P starting from the
       initial state and clock region r^0 = (0,...,0):
       dfsP(q_P^0, r^0).
3      Function dfsP(state q, clock region r)
4      begin
5          Find the next clock region r'' when we have a
           transition out of q.
6          w ← Time between r and r''.
7          foreach transition t taken at r'' do
8              Find the next clock region r' once t is taken by
               resetting the appropriate clock.
9              q' ← Target state of t.
10             if {q', r'} ∉ Q_R then
11                 Add state {q', r'} to Q_R with proposition
                   L_P(t) of t.
12                 Add {q, r} → {q', r'} to →_R with w.
13                 Continue search from {q', r'}: dfsP(q', r')
14             else if {q, r} → {q', r'} ∉→_R then
15                 Add {q, r} → {q', r'} to →_R with w.
```

We now show that the region automaton indeed captures the behavior of the team. Given a run r_R on $\mathbf{R}$, we denote the corresponding word (see Sec. II-A) as ω_R and the corresponding time sequence of satisfying instances of propositions (see Sec. III) as $\mathbb{T}_R$. We have

Proposition IV.4. *Given individual runs of the team, $r_i = q_i^0 q_i^1 \ldots, i = 1, \ldots, m$, there is a corresponding run r_R on $\mathbf{R}$ such that, the word ω generated by the team is ω_R and the time sequence $\mathbb{T}$ of satisfying instances of propositions for the team is $\mathbb{T}_R$.*

Proof. Each run $r_i = q_i^0 q_i^1 \ldots$ uniquely corresponds to a run on $\mathbf{D}_i$, $r_{D_i} = q_{D_i}^0 (q_i^0 q_i^1)(q_i^1 q_i^2), \ldots$. Since the weight $w_{T_i}(q_i^k, q_i^{k+1})$ is defined to be the clock constraint associated with state $q_i^k q_i^{k+1}$ on $\mathbf{A}_i$, there is a sequence of transitions r_P on the product timed automaton $\mathbf{P}$ such that a transition occurs if some set of states are visited on $\mathbf{T}_i$'s. Since $\mathbf{R}$ is a bisimulation quotient of $\mathbf{P}$, this sequence of transitions corresponds to a run on $r_R = q_R^0 \{q^0, r^0\}\{q^1, r^1\} \ldots$, such that each transition $(\{q^k, r^k\}, \{q^{k+1}, r^{k+1}\})$ in r_R corresponds to some set of states being visited on $\mathbf{T}_i$'s, which we denote as $I(\{q^k, r^k\}, \{q^{k+1}, r^{k+1}\})$. Similarly, if some set of states are visited on $\mathbf{T}_i$'s, there is a corresponding transition $(\{q^k, r^k\}, \{q^{k+1}, r^{k+1}\})$ for some k. The set of propositions satisfied at the set of states $I(\{q^k, r^k\}, \{q^{k+1}, r^{k+1}\})$ is satisfied when the transition $(\{q^k, r^k\}, \{q^{k+1}, r^{k+1}\})$ is taken and the state $\{q^{k+1}, r^{k+1}\}$ is reached on $\mathbf{R}$. Therefore the word ω generated by $\mathbf{R}$ is exactly the word generated by the team. Also note that the state $\{q^{k+1}, r^{k+1}\}$ corresponds to robots leaving vertices $I(\{q^k, r^k\}, \{q^{k+1}, r^{k+1}\})$. Because robots spend zero time at vertices, $\{q^{k+1}, r^{k+1}\}$ is reached at the same time as when robots reach $I(\{q^k, r^k\}, \{q^{k+1}, r^{k+1}\})$. Therefore, the time sequence $\mathbb{T}$ of satisfying instances of propositions for the team is exactly $\mathbb{T}_R$ for run r_R. ■

D. Generating the optimal runs for individual robots

Once the region automaton capturing the behavior of the team is constructed, we can use Alg. OPTIMAL-RUN [8] to obtain an optimal run $r_R^\star$ on $\mathbf{R}$ that minimizes the $\limsup$ as defined in (4). The optimal run $r_R^\star$ always consists of a finite sequence of states of $\mathbf{R}$ (prefix), followed by infinite repetitions of another finite sequence of states of $\mathbf{R}$ (suffix). Such a run is said to be in a prefix-suffix form.

For the example we have shown throughout this section, running Alg. OPTIMAL-RUN [8] on $\mathbf{R}$ given in Fig. 4 for the formula $\phi := \mathbf{GF}\pi$ results in the optimal run

$\mathbb{T}$	0	2	3	4	6	8	10	…
$r_R^\star$	ab,ab (0,0)	ba,bc (0,0)	ba,cb (1,0)	ab,ba (0,0)	ba,ab (0,0)	ab,ba (0,0)	ba,ab (0,0)	…
$\mathcal{L}_{\mathbf{R}}(\cdot)$		π		π	π	π	π	…

where the first row corresponds to the times when transitions occur, the second row comprises the run $r_R^\star$, and the last row shows the satisfying atomic propositions. For this run, we see that $\{(ab, ab), (0,0)\}\{(ba, bc), (0,0)\}\{(ba, cb), (1,0)\}$ is the prefix and $\{(ab, ba), (0,0)\}\{(ba, ab), (0,0)\}$ is the suffix and will be repeated infinite number of times. Moreover, for this example, the time sequence of satisfaction of π is $\mathbb{T}^\pi = 2, 4, 6, 8, 10, \ldots$ and the cost as defined in (4) is $J(\mathbb{T}^\pi) = 2$.

Given a run r_R of $\mathbf{R}$, we can finally project it down to individual robots to obtain individual runs r_i of $\mathbf{T}_i$.

Definition IV.5 (Projection of a run on $\mathbf{R}$ to $\mathbf{T}_i$'s). *Given a run r_R on $\mathbf{R}$ where*

$$r_R = \big\{(q_1^0 q_1^1, \ldots, q_m^0 q_m^1), (v^0(x_1), \ldots, v^0(x_m))\big\} \\ \big\{(q_1^1 q_1^2, \ldots, q_m^1 q_m^2), (v^1(x_1), \ldots, v^1(x_m))\big\} \ldots,$$

we define its projection on $\mathbf{T}_i$ as run $r_i = q_i^0 q_i^1 \ldots$ for all $i = 1, \ldots, m$, where q_i^k only appears in r_i if $v^k(x_i) = 0$.

It can be easily seen that, given r_R, its set of projected runs r_i correspond to r_R as defined in Prop. IV.4, *i.e.*, the behavior of the team where robot i follows run r_i is captured exactly by r_R. Moreover, if run r_R is in prefix-suffix form, all individual runs r_i projected from r_R are in prefix-suffix form. Therefore, the individual runs projected from the optimal run $r_R^\star$ are always in prefix-suffix form. For the optimal run we obtained for the previous example, using Def. IV.5, we have runs of individual robots as follows:

$\mathbb{T}$	0	2	3	4	6	8	10	…
$r_1^\star$	a	b		a	b	a	b	…
$r_2^\star$	a	b	c	b	a	b	a	…

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 the

clock of the first robot is not zero at this time, *i.e.*, $v^3(x_1) \neq 0$, and b does not appear in $r_1^\star$ at time $t = 3$.

We finally summarize our approach in Alg. 2 and show that this algorithm indeed gives a solution to Prob. III.1.

Proposition IV.6. *Alg. 2 solves Prob. III.1.*

Proof. Note that Alg. 2 combines all steps outlined in this section. Run $r_R^\star$ obtained from Alg. OPTIMAL-RUN both satisfies ϕ and minimizes (4) among all runs of $\mathbf{R}$, which was shown in [8]. As shown in Prop. IV.4 and as mentioned above, there is a one-to-one correspondence between a set of runs $\{r_1, \ldots, r_m\}$ and a run r_R. Therefore, $\{r_1^\star, \ldots, r_m^\star\}$ as a projection of $r_R^\star$ onto $\mathbf{T}_i$'s is the solution to Prob. III.1. ■

Algorithm 2: MULTI-ROBOT-OPTIMAL-RUN

Input: m $\mathbf{T}_i$'s and a LTL specification ϕ of form (3).
Output: A set of runs $\{r_1^\star, \ldots, r_m^\star\}$ that both satisfies ϕ and minimizes (4).

begin
  forall the $\mathbf{T}_i$ **do**
    Construct the timed automaton A_i by first constructing the dual TS D_i and then defining clocks and clock constraints.
  Find the product timed automaton $\mathbf{P} = \Pi_{i=1}^m A_i$.
  Construct the region automaton $\mathbf{R}$ using OBTAIN-REGION-AUTOMATON .
  Find the optimal run $r_R^\star$ using OPTIMAL-RUN [8].
  Project $r_R^\star$ to T_i's to obtain runs $\{r_1^\star, \ldots, r_m^\star\}$.

V. IMPLEMENTATION AND CASE STUDIES

We implemented Alg. 2 in objective-C as the software package LTL OPTIMAL MULTI-ROBOT PLANNER (LOMP) and used it in conjunction with our earlier OPTIMAL-RUN [8] algorithm to obtain simulations of robots performing persistent data gathering missions in a road network environment. Our user-friendly software package is available at `http://hyness.bu.edu/Software.html`. It utilizes the dot tool [19] to visualize transition systems and the OPTIMAL-RUN algorithm uses the LTL2BA software [20] to convert LTL specifications to Büchi automata. A typical usage of our software comprises three steps:

(i) The user defines $\mathbf{T}_i$'s in text and imports them to the application. Then, the application creates the region automaton $\mathbf{R}$ following the steps detailed in Sec.IV and exports an M-file which defines $\mathbf{R}$ in Matlab.
(ii) OPTIMAL-RUN algorithm is executed in Matlab to find the optimal run $r_R^\star$ on $\mathbf{R}$, which is projected onto $\mathbf{T}_i, i = 1, \ldots, m$ to obtain the solution to Prob. III.1.
(iii) Finally, the resulting motion of the team is shown in a simulator.

The road network that we consider for our case studies is a collection of roads, intersections, and task locations. In this road network, a road connects two intersections and the task locations are always located on the side of a road. The transition system that we used to model the motion of the robots in this environment is illustrated in Fig. 5. We assume that the transition systems $\mathbf{T}_i$ of robots are identical except at the initial state. In $\mathbf{T}_i$'s, the weights of transitions are quantized so that the resulting region transition system has a manageable size while still preserving the relative distances of the road segments. In the following, we consider two cases where the weights fall in the range $\{1, \ldots, 5\}$ and $\{1, \ldots, 20\}$, respectively.

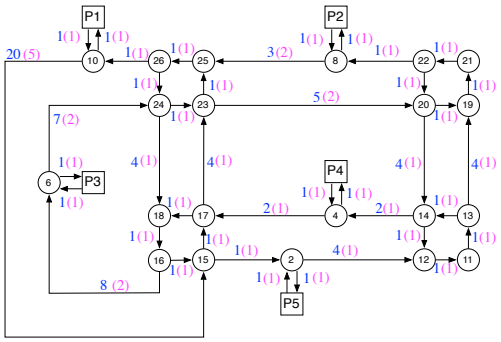


Fig. 5: The road network showing the labels of task locations and the quantized weights of the road segments for the two case studies. Values in blue are weights for the case where the weights are in $\{1 \ldots 20\}$ and values in magenta are for the case where the weights are in $\{1 \ldots 5\}$.

We consider a persistent monitoring task where robots are deployed to repeatedly gather and upload data. We require robot 1 to gather data at P1 and upload the gathered data at P5; and robot 2 to gather data at P2 and upload the gathered data at P4. To specify this task, we let the set of atomic propositions to be

$$\Pi = \{\texttt{Gather}, \texttt{R1Gather}, \texttt{R1Upload}, \texttt{R2Gather}, \texttt{R2Upload}\}$$

and assign the atomic propositions as follows:

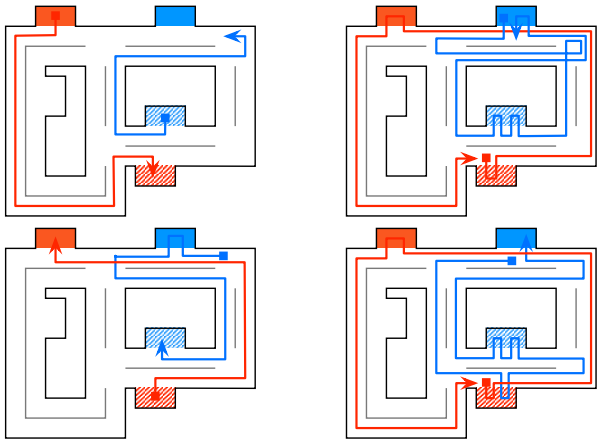
$$\mathcal{L}_1(P_1) = \{\texttt{R1Gather}, \texttt{Gather}\}, \mathcal{L}_1(P5) = \{\texttt{R1Upload}\}$$
$$\mathcal{L}_2(P_2) = \{\texttt{R2Gather}, \texttt{Gather}\}, \mathcal{L}_2(P4) = \{\texttt{R2Upload}\}.$$

We aim to minimize the maximum time in between data-gatherings performed by either robot 1 or 2. Therefore we set the proposition `Gather` to be satisfied when either robots visit their gathering locations, and we set it as the optimizing proposition (π as in formula (3)). We set the propositions $\{\texttt{R1Gather}, \texttt{R1Upload}\}$ and $\{\texttt{R2Gather}, \texttt{R2Upload}\}$ to be robot specific since robots gather and upload at different locations. For both robots, we enforce the rule that, after each data gathering, the data must be uploaded at the upload location before another data gathering. This rule can be specified in LTL as follows:

$$\varphi = \mathbf{G}(\texttt{R1Gather} \Rightarrow \mathbf{X}(\neg \texttt{R1Gather}\ \mathcal{U}\ \texttt{R1Upload})) \wedge \mathbf{G}(\texttt{R2Gather} \Rightarrow \mathbf{X}(\neg \texttt{R2Gather}\ \mathcal{U}\ \texttt{R2Upload})).$$

Our overall LTL formula in the form of (3) is $\phi = \varphi \wedge \mathbf{G}\,\mathbf{F}\,\texttt{Gather}$.

Running our algorithms on an iMac i5 quad-core computer, we obtain the solutions as illustrated in Fig. 6. For the case where the weights are in the range $\{1 \ldots 5\}$ the algorithm ran for 90 seconds, the region transition system



(a) (b)

Fig. 6: Simulated team trajectories for the two case studies. (a) and (b) correspond to the cases where the weights are within the ranges $\{1 \dots 5\}$ and $\{1 \dots 20\}$, respectively. Robot 1 and robot 2 travel between red and blue task locations respectively. Regions filled with a solid color are data gathering locations and regions with a diagonal pattern are upload locations.

$\mathbf{R}$ that the OPTIMAL-RUN algorithm worked on had 2337 states and the value of the cost function was 11 time units, meaning that the maximum time in between data gatherings was 11 time units. For the case where the weights are in the range $\{1 \dots 20\}$ our algorithm ran for 10 minutes, $\mathbf{R}$ had 6191 states and the value of the cost function was 22 time units. Our video submission accompanying the paper displays the robot trajectories for both cases.

It is interesting to note that, for the case where the weights are in $\{1 \dots 20\}$, the optimal team trajectories have robots spending extra time entering and exiting some vertices. This behavior is actually time-wise optimal since it decreases the maximum time between satisfying instances of the optimizing proposition, minimizing the cost function.

VI. CONCLUSIONS

In this paper we presented a method for planning the optimal motion for a team of robots in a common environment subject to temporal logic constraints. The problem is important in applications where multiple robots have to perform a sequence of operations collectively subject to various external constraints. We considered temporal logic specifications which contain an optimizing proposition that must be repeatedly satisfied. The motion plan that our method provides is optimal in the sense that it minimizes the maximum time between satisfying instances of the optimizing proposition.

There are many promising directions for future work. In particular, we are looking at the case where one allows delays when robots take transitions. We are also investigating more realistic robot models such as Markov Decision Processes (MDPs) and Partially Observable MDPs.

REFERENCES

[1] M. Antoniotti and B. Mishra, "Discrete event models + temporal logic = supervisory controller: Automatic synthesis of locomotion controllers," in *IEEE Int. Conf. on Robotics and Automation*, Nagoya, Japan, 1995, pp. 1441–1446.

[2] S. G. Loizou and K. J. Kyriakopoulos, "Automatic synthesis of multiagent motion tasks based on LTL specifications," in *IEEE Conf. on Decision and Control*, Paradise Island, Bahamas, 2004, pp. 153–158.

[3] C. Belta, V. Isler, and G. J. Pappas, "Discrete abstractions for robot motion planning and control in polygonal environment," *IEEE Transactions on Robotics*, vol. 21, no. 5, pp. 864–875, 2005.

[4] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal logic-based reactive mission and motion planning," *IEEE Transactions on Robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[5] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon control for temporal logic specifications," in *Hybrid systems: Computation and Control*, Stockholm, Sweden, 2010, pp. 101–110.

[6] M. Y. Vardi and P. Wolper, "An automata-theoretic approach to automatic program verification," in *Logic in Computer Science*, 1986, pp. 322–331.

[7] G. Holzmann, "The model checker SPIN," *IEEE Transactions on Software Engineering*, vol. 25, no. 5, pp. 279–295, 1997.

[8] S. L. Smith, J. Tůmová, C. Belta, and D. Rus, "Optimal path planning under temporal logic constraints," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, Taipei, Taiwan, Oct. 2010, pp. 3288–3293.

[9] L. C. G. J. M. Habets and J. H. van Schuppen, "A control problem for affine dynamical systems on a full-dimensional polytope," *Automatica*, vol. 40, pp. 21–35, 2004.

[10] C. Belta and L. C. G. J. M. Habets, "Control of a class of nonlinear systems on rectangles," *IEEE Transactions on Automatic Control*, vol. 51, no. 11, pp. 1749–1759, 2006.

[11] M. Kloetzer and C. Belta, "Automatic deployment of distributed teams of robots from temporal logic specifications," *IEEE Transactions on Robotics*, vol. 26, no. 1, pp. 48–61, 2010.

[12] M. M. Quottrup, T. Bak, and R. Izadi-Zamanabadi, "Multi-robot motion planning: A timed automata approach," in *IEEE Int. Conf. on Robotics and Automation*, New Orleans, LA, 2004, pp. 4417–4422.

[13] P. Toth and D. Vigo, Eds., *The Vehicle Routing Problem*, ser. Monographs on Discrete Mathematics and Applications. SIAM, 2001.

[14] S. Karaman and E. Frazzoli, "Complex mission optimization for multiple-uavs using linear temporal logic," in *American Control Conference*, Seattle, WA, 2008, pp. 2003–2009.

[15] ——, "Vehicle routing problem with metric temporal logic specifications," in *IEEE Conf. on Decision and Control*, Cancún, México, 2008, pp. 3953–3958.

[16] R. Milner, *Communication and concurrency*. Prentice-Hall, 1989.

[17] P. Gastin and D. Oddoux, "Fast LTL to Buchi automata translation," *Lecture Notes in Computer Science*, pp. 53–65, 2001.

[18] R. Alur and D. Dill, "A theory of timed automata," *Theoretical computer science*, vol. 126, no. 2, pp. 183–235, 1994.

[19] "Graphviz - graph visualization software," http://www.graphviz.org/.

[20] "LTL2BA," http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/index.php.