The 2010 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 18-22, 2010, Taipei, Taiwan

Optimal Path Planning under Temporal Logic Constraints

Stephen L. Smith Jana Tůmová Calin Belta Daniela Rus

***Abstract*— In this paper we present a method for automatically generating optimal robot trajectories satisfying high level mission specifications. The motion of the robot in the environment is modeled as a weighted transition system. The mission is specified by a general linear temporal logic formula. In addition, we require that an *optimizing* proposition must be repeatedly satisfied. The cost function that we seek to minimize is the maximum time between satisfying instances of the optimizing proposition. For every environment model, and for every formula, our method computes a robot trajectory which minimizes the cost function. The problem is motivated by robotic monitoring and data gathering. In this setting, the optimizing proposition is satisfied at locations where data can be uploaded, and the formula specifies a an infinite horizon data collection mission. Our method utilizes Büchi automata to produce an automaton (which can be thought of as a graph) whose runs satisfy the temporal logic formula. We then present a graph algorithm which computes a path corresponding to the optimal robot trajectory. We also present an implementation for a robot performing a data gathering mission.**

I. Introduction

The goal of this paper is to plan the optimal motion of a robot subject to temporal logic constraints. This is an important problem in many applications where the robot has to perform a sequence of operations subject to external constraints. For example, in a persistent data gathering task the robot is tasked to gather data at several locations and then visit a different set of upload sites to transmit the data. Referring to Fig. 1, we would like to enable tasks such as "Repeatedly gather data at locations P1, P4, and P5. Upload data at either P2 or P3 after each data-gather. Follow the road rules, and avoid the road connecting I4 to I2." We wish to determine robot motion that completes the task, and minimizes a cost function, such as the maximum time between data uploads.

Recently there has been an increased interest in using temporal logic to specify mission plans for robots [1], [2], [3], [4], [5], [6]. Temporal logic is appealing because it provides a formal high level language in which to describe a complex mission. In addition, tools from model checking [7], [8], [9] can be used to verify the existence of a robot trajectory satisfying the specification, and can produce a satisfying trajectory. However, frequently there are multiple robot trajectories that satisfy a given specification. In this case, one would like to choose the "optimal" trajectory according to a cost function. The current tools from model checking do not provide a method for doing this. In this paper we consider linear temporal logic specifications, and a particular form of cost function, and provide a method for computing optimal trajectories.

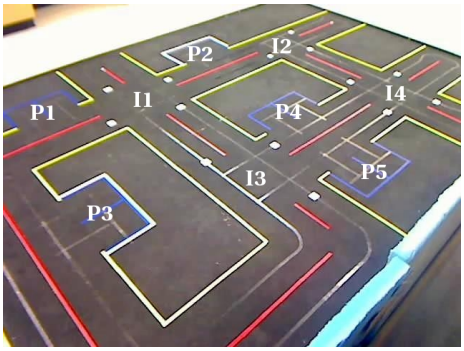

Fig. 1. An environment consisting of roads, intersections and parking lots. An example mission in the environment is "Repeatedly gather data at locations P1, P4, and P5. Upload data at either P2 or P3 after each data-gather. Follow the road rules, and avoid the road connecting I4 to I2."

The problem considered in this paper is related to vehicle routing [10], where the goal is to plan routes for vehicles to service customers. In [11], the authors consider a vehicle routing problem with metric temporal logic constraints. 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 solution by converting the problem to a mixed integer linear program (MILP). The method applies for specifications where the temporal operators are applied only to atomic propositions. Thus, the method does not apply to persistent monitoring and data gathering problems, which have specifications of the form "always eventually." In addition, the approach that we present in this paper leads to an optimization problem on a graph, rather than a MILP.

The contribution of this paper is to present a cost function for which we can determine an optimal robot trajectory that satisfies a general linear temporal logic formula. The cost function is motivated by problems in monitoring and data gathering, and it seeks to minimize the time between satisfying instances of a single *optimizing proposition*. Our solution, summarized in the Optimal-Run algorithm of Section IV, operates as follows. We represent the robot and environment as a weighted transition system. Then, we convert the linear temporal logic specification to a Büchi automaton. We synchronize the transition system with the Büchi automaton creating a product automaton. In this automaton a satisfying run is any run which visits a set of accepting states infinitely often. We show that there exists an optimal run that is in

This research was partially supported by ONR-MURI Award N00014-09-1-1051, ARO Award W911NF-09-1-0088, and grant GA201/09/1389 at Masaryk University.

S. L. Smith and D. Rus are with the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139 (slsmith@mit.edu; rus@csail.mit.edu). J. Tůmová and C. Belta are with the Department of Mechanical Engineering, Boston University, Boston, MA 02215 (tumova@bu.edu;cbelta@bu.edu). J. Tůmová is also affiliated with Faculty of Informatics, Masaryk University, Brno, Czech Republic.

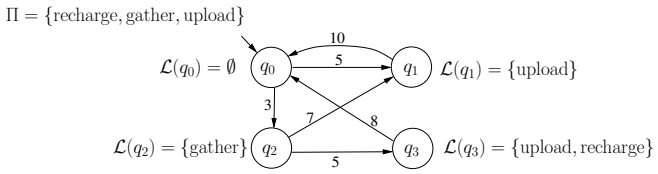

Fig. 2. An example of a weighted transition system. A correct run of the system is for instance $q_0q_2q_1q_0q_2q_3q_0\ldots$, producing the word $\emptyset\{\text{gather}\}\{\text{upload}\}\emptyset\{\text{gather}\}\{\text{upload,recharge}\}\emptyset\ldots$.

"prefix-suffix" structure, implying that we can search for runs with a finite transient, followed by a periodic steady-state. Thus, we create a polynomial time graph algorithm based on solutions of bottleneck shortest path problems to find an optimal cycle containing an accepting state. We implement our solution on the physical testbed shown in Fig. 1.

For simplicity of the presentation, we assume that the robot moves among the vertices of an environment modeled as a graph. However, by using feedback controllers for facet reachability and invariance in polytopes [12], [13] the method developed in this paper can be easily applied for motion planning and control of a robot with "realistic" continuous dynamics (*e.g.,* unicycle) traversing an environment partitioned using popular partitioning schemes such as triangulations and rectangular partitions.

Due to page constraints we omit all proofs of all results. An extended version of this paper, which includes all proofs, can be found in [14].

II. Preliminaries

In this section we briefly review some aspects of linear temporal logic (LTL). LTL considers a finite set of variables Π, called *atomic propositions*, each of which can be either true or false. Propositions can capture properties such as "the robot is located in region 1", or "the robot is recharging."

Given a system model, LTL allows us to express the time evolution of the state of the system. We consider a type of finite model called the *weighted transition system*.

Definition II.1 (Weighted Transition System)**.** *A weighted transition system is a tuple $\mathcal{T} := (Q, q_0, R, \Pi, \mathcal{L}, w)$, consisting of (i) a finite set of states Q; (ii) an initial state $q_0 \in Q$; (iii) a transition relation $R \subseteq Q \times Q$; (iv) a set of atomic propositions Π; (v) a labeling function $\mathcal{L} : Q \to 2^{\Pi}$; (vi) a weight function $w : R \to \mathbb{R}_{>0}$.*

We assume that the transition system is non-blocking, implying that there is a transition from each state. The transition relation has the expected definition: given that the system is in state $q_1 \in Q$ at time t_1, the system is in state q_2 at time $t_1 + w\big((q_1, q_2)\big)$ if and only if $(q_1, q_2) \in R$. The labeling function defines for each state $q \in Q$, the set $\mathcal{L}(q)$ of all atomic propositions valid in q.

For our transition system we can define a *run* $r_{\mathcal{T}}$ to be an infinite sequence of states $q_0q_1q_2\ldots$ such that $q_0 \in Q_0$, $q_i \in Q$, for all i, and $(q_i, q_{i+1}) \in R$, for all i. A run $r_{\mathcal{T}}$ defines a *word* $\mathcal{L}(q_0)\mathcal{L}(q_1)\mathcal{L}(q_2)\ldots$ consisting of sets of atomic propositions valid at each state. An example of a weighted transition system is given in Fig. 2.

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

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

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

LTL formulas are interpreted over infinite runs (generated by the transition system $\mathcal{T}$ from Def. II.1). Informally, $\mathbf{X}\,\alpha$ states that at the next state of a run, proposition α is true (i.e., $\alpha \in \mathcal{L}(q_1)$). In contrast, $\alpha_1\,\mathcal{U}\,\alpha_2$ 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 useful 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 $\mathbf{G}\,\alpha$ states that proposition α holds at all states of the run, and $\mathbf{F}\,\alpha$ states that α holds at some future time instance.

An LTL formula can be represented in an automata-theoretic setting as *Büchi automaton*, defined as follows:

Definition II.3 (Büchi Automaton)**.** *A Büchi automaton is a tuple $\mathcal{B} := (S, S_0, \Sigma, \delta, F)$, consisting of (i) a finite set of states S; (ii) a set of initial states $S_0 \subseteq S$; (iii) an input alphabet Σ; (iv) a non-deterministic transition relation $\delta \subseteq S \times \Sigma \times S$; (v) a set of accepting (final) states $F \subseteq S$.*

The semantics of Büchi automata are defined over infinite input words. Setting the input alphabet $\Sigma = 2^{\Pi}$, the semantics are defined over the words consisting of sets of atomic propositions, i.e. those produced by a run of the transition system. Let $\omega = \omega_0\omega_1\omega_2\ldots$ be an infinite input word of automaton $\mathcal{B}$, where $\omega_i \in \Sigma$ for each $i \in \mathbb{N}$ (for example, the input $\omega = \mathcal{L}(q_0)\mathcal{L}(q_1)\mathcal{L}(q_2)\ldots$ could be a word produced by a run $q_0q_1q_2\ldots$ of the transition system $\mathcal{T}$).

A *run* of the Büchi automaton *over* an input word $\omega = \omega_0\omega_1\omega_2\ldots$ is a sequence $r_{\mathcal{B}} = s_0s_1s_2\ldots$, such that $s_0 \in S_0$, and $(s_i, \omega_i, s_{i+1}) \in \delta$, for all $i \in \mathbb{N}$.

Definition II.4 (Büchi Acceptance)**.** *A word ω is accepted by the Büchi automaton $\mathcal{B}$ if and only if there exists $r_{\mathcal{B}}$ over ω so that $\inf(r_{\mathcal{B}}) \cap F \neq \emptyset$, where $\inf(r_{\mathcal{B}})$ denotes the set of states appearing infinitely often in run $r_{\mathcal{B}}$.*

For any LTL formula ϕ over a set of atomic propositions Π, there exists a Büchi automaton $\mathcal{B}_\phi$ with input alphabet 2^{Π} accepting all and only the infinite words satisfying formula ϕ [7]. Efficient translation implementations have been developed in [15], [16]. The size of the obtained Büchi automaton is, in general, exponential with respect to the size of the formula. However, the exponential complexity is in practice not restrictive as the LTL formulas are typically quite small. An example of a Büchi automaton is given in Figure 3.

III. Problem Statement and Approach

Consider a single robot in an environment represented as a transition system (as defined in Section II) $\mathcal{T} = (Q, q_0, R, \Pi, \mathcal{L}, w)$. A run in the transition system starting at q_0 defines a corresponding trajectory of the robot in the

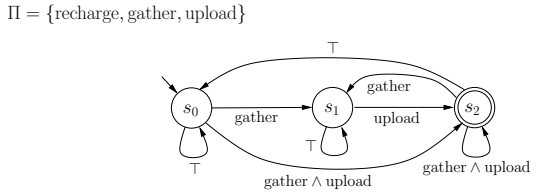

Fig. 3. A Büchi automaton corresponding to LTL formula ($\mathbf{G}\,\mathbf{F}\,\text{gather} \wedge \mathbf{G}\,\mathbf{F}\,\text{upload}$) over the alphabet Π. The illustration of the automaton is simplified. In fact, each transition labeled with $\top$ represents $|2^{\Pi}|$ transitions labeled with all different subsets of atomic propositions. Similarly, a transition labeled with gather represent $|2^{\Pi}|/2$ transitions labeled with all subsets of atomic propositions containing the proposition gather, etc.

environment. The time to take transition $(q_1, q_2) \in R$ (i.e., the time for the robot to travel from q_1 to q_2) is $w(q_1, q_2)$.

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

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

where φ can be any LTL formula over Π, and $\mathbf{G}\,\mathbf{F}\,\pi$ specifies that the proposition π must be satisfied infinitely often, and will simply ensure well-posedness of our optimization.

Let each run of $\mathcal{T}$ start at time $t = 0$, and assume that there is at least one run satisfying LTL formula (1). For each satisfying run $r_{\mathcal{T}} = q_0 q_1 q_2 \ldots$, there is a corresponding word of sets of atomic propositions $\omega = \omega_0\omega_1\omega_2 \ldots$, where $\omega_i = \mathcal{L}(q_i)$. Associated with $r_{\mathcal{T}}$ there is a sequence of time instances $\mathbb{T} := t_0, t_1, t_2, \ldots$, where $t_0 = 0$, and t_i denotes the time at which state q_i is reached ($t_{i+1} = t_i + w(q_i, q_{i+1})$). From this time sequence we can extract all time instances at which the proposition π is satisfied. We let $\mathbb{T}_\pi$ denote the sequence of satisfying instances of the proposition π.

Our goal is to synthesize a run $r_{\mathcal{T}}$ (or robot trajectory) satisfying LTL formula (1), and minimizing the cost function

$$\mathcal{C}(r_{\mathcal{T}}) = \limsup_{i \to +\infty} \left(\mathbb{T}_\pi(i+1) - \mathbb{T}_\pi(i)\right), \tag{2}$$

where $\mathbb{T}_\pi(i)$ is the ith satisfying time instance of proposition π. Note that a finite cost in (2) enforces that $\mathbf{G}\,\mathbf{F}\,\pi$ is satisfied. Thus, the specification appears in ϕ merely to ensure that any satisfying run has finite cost.

Problem III.1. *Determine an algorithm that takes as input a weighted transition system $\mathcal{T}$, an LTL formula ϕ in form* (1)*, and an optimizing proposition π, and outputs a run $r_{\mathcal{T}}$ minimizing the cost $\mathcal{C}(r_{\mathcal{T}})$ in* (2)*.*

Remarks III.2 (Comments on problem statement)**.** Cost function form: *The transition system produces infinite runs. Thus, cost function* (2) *evaluates the steady-state time between satisfying instances of π. In the upcoming sections we design an algorithm that produces runs which reach steady-state in* finite time*. Thus, the runs produced will achieve the cost in* (2) *in finite time.*

Expressivity of LTL formula (1): *The LTL formula φ in (1) allows us to specify various rich robot motion requirements. Some examples are* global absence *($\mathbf{G}\,\neg\psi$, globally keep avoiding ψ),* response *($\mathbf{G}\,(\psi_1 \Rightarrow \mathbf{F}\,\psi_2)$, whenever ψ_1 holds true, ψ_2 will happen in future),* reactivity *($\mathbf{G}\,\mathbf{F}\,\psi_1 \Rightarrow \mathbf{G}\,\mathbf{F}\,\psi_2$, if ψ_1 holds in future for any time point, ψ_2 has to happen in future for any time point as well), and* sequencing *($\psi_1\,\mathcal{U}\,\psi_2\,\mathcal{U}\,\psi_3$, ψ_1 holds until ψ_2 happens, which holds until ψ_3 happens). For concrete examples, see Section V.* □

IV. Problem Solution

In this section we describe our solution to Problem III.1 which applies ideas from automata-theoretic model checking.

A. The Product Automaton

Consider the weighted transition system $\mathcal{T}$, and a proposition $\pi \in \Pi$. In addition, consider an LTL formula $\phi = \varphi \wedge \mathbf{G}\,\mathbf{F}\,\pi$ over Π in form (1), translated into a Büchi automaton $\mathcal{B}_\phi$. We now define a new object, which we call the *product automaton*.

Definition IV.1 (Product Automaton)**.** *The product automaton $\mathcal{P} = \mathcal{T} \times \mathcal{B}_\phi$ between the transition system $\mathcal{T}$ and the Büchi automaton $\mathcal{B}_\phi$ is defined as the tuple $\mathcal{P} := (S_\mathcal{P}, S_{\mathcal{P},0}, \delta_\mathcal{P}, F_\mathcal{P}, w_\mathcal{P}, S_{\mathcal{P},\pi})$, consisting of*

(i) *a finite set of states $S_\mathcal{P} = Q \times S$,*
(ii) *a set of initial states $S_{\mathcal{P},0} = \{q_0\} \times S_0$,*
(iii) *a transition relation $\delta_\mathcal{P} \subseteq S_\mathcal{P} \times S_\mathcal{P}$, where $\big((q,s),(\bar{q},\bar{s})\big) \in \delta_\mathcal{P}$ if and only if $(q,\bar{q}) \in R$ and $(s, \mathcal{L}(q), \bar{s}) \in \delta$.*
(iv) *a set of accepting (final) states $F_\mathcal{P} = Q \times F$.*
(v) *a weight function $w_\mathcal{P} : \delta_\mathcal{P} \to \mathbb{R}_{>0}$, where $w_\mathcal{P}\big((q,s),(\bar{q},\bar{s})\big) = w(q,\bar{q})$, $\forall\ \big((q,s),(\bar{q},\bar{s})\big) \in \delta_\mathcal{P}$.*
(vi) *a set of states $S_{\mathcal{P},\pi} \subseteq S_\mathcal{P}$ in which the proposition π holds true. Thus, $(q,s) \in S_{\mathcal{P},\pi}$ if and only if $\pi \in \mathcal{L}(q)$.*

The product automaton (as defined above) can be seen as a Büchi automaton with a trivial input alphabet. Since the alphabet is trivial, we omit it. Thus, we say that a run $r_\mathcal{P}$ in product automaton $\mathcal{P}$ is accepting if $\inf(r_\mathcal{P}) \cap F_\mathcal{P} \neq \emptyset$.

As in the transition system, we associate with each run $r_\mathcal{P} = p_0 p_1 p_2 \ldots$, a sequence of time instances $\mathbb{T}_\mathcal{P} := t_0 t_1 t_2 \ldots$, where $t_0 = 0$, and t_i denotes the time at which the ith vertex in the run is reached ($t_{i+1} = t_i + w_\mathcal{P}(p_i, p_{i+1})$). From this time sequence we can extract a sequence $\mathbb{T}_{\mathcal{P},\pi}$, containing time instances t_i, where $p_i \in S_{\mathcal{P},\pi}$ (i.e. $\mathbb{T}_{\mathcal{P},\pi}$ is a sequence of satisfying instances of the optimizing proposition π in $\mathcal{T}$). The cost of a run $r_\mathcal{P}$ on the product automaton $\mathcal{P}$ (which corresponds to cost function (2) on transition system $\mathcal{T}$) is

$$\mathcal{C}_\mathcal{P}(r_\mathcal{P}) = \limsup_{i \to +\infty} \left(\mathbb{T}_{\mathcal{P},\pi}(i+1) - \mathbb{T}_{\mathcal{P},\pi}(i)\right). \tag{3}$$

The product automaton can also be viewed as a weighted graph, where the states define vertices of the graph and the transitions define the edges. Thus, we at times refer to runs of the product automaton as *paths*. A *finite path* is then a finite fragment of an infinite path.

Each accepting run of the product automaton can be projected to a run of the transition system satisfying the LTL formula as follows.

Proposition IV.2 (Product Run Projection, [7])**.** *For any accepting run $r_\mathcal{P} = (q_0, s_0)(q_1, s_1)(q_2, s_2) \ldots$ of the product automaton $\mathcal{P}$, the sequence $r_{\mathcal{T}} = q_0 q_1 q_2 \ldots$ is a run of*

$\mathcal{T}$ satisfying ϕ. Furthermore, the values of cost functions $\mathcal{C}_\mathcal{P}$ and $\mathcal{C}$ are equal. Similarly, if $r_\mathcal{T} = q_0 q_1 q_2 \dots$ is a run of $\mathcal{T}$ satisfying ϕ, then there exists an accepting run $r_\mathcal{P} = (q_0, s_0)(q_1, s_1)(q_2, s_2) \dots$ of the product automaton $\mathcal{P}$, such that the values of cost functions $\mathcal{C}$ and $\mathcal{C}_\mathcal{P}$ are equal.

Finally, we need to discuss the structure of an accepting run of a product automaton $\mathcal{P}$.

Definition IV.3 (Prefix-Suffix Structure)**.** *A* prefix *of an accepting run is a finite path from an initial state to an accepting state $f \in F_\mathcal{P}$ containing no other occurrence of f. A* periodic suffix *is an infinite run originating at the accepting state f, and periodically repeating a finite path originating and ending at f, and containing no other occurrence of f (but possibly containing other vertices in $F_\mathcal{P}$). An accepting run is in* prefix-suffix *structure if it consists of a prefix followed by a periodic suffix.*

Intuitively, the prefix can be thought of as the transient, while the suffix is the steady-state periodic behavior.

Lemma IV.4 (Prefix-Suffix Structure)**.** *At least one of the accepting runs $r_\mathcal{P}$ of $\mathcal{P}$ that minimizes cost function $\mathcal{C}_\mathcal{P}(r_\mathcal{P})$ is in prefix-suffix structure.*

The proof of the previous result is contained in [14].

Definition IV.5 (Suffix Cost)**.** *The cost of the suffix $p_0 p_1 \dots p_n p_0 p_1 \dots$ of a run $r_\mathcal{P}$ is defined as follows. Let $t_{0,0}, t_{0,1}, \dots, t_{0,n}, t_{1,0}, t_{1,1} \dots$ be the sequence of times at which the vertices of the suffix are reached on run $r_\mathcal{P}$. Extract the sub-sequence $\mathbb{T}^{\text{suf}}_\mathcal{P}$ of times $t_{i,j}$, where $p_j \in S_{\mathcal{P},\pi}$ (i.e. the satisfying instances of proposition π in transition system $\mathcal{T}$). Then, the cost of the suffix is $\mathcal{C}^{\text{suf}}_\mathcal{P}(r_\mathcal{P}) = \max_{i \in \mathbb{N}}(\mathbb{T}^{\text{suf}}_\mathcal{P}(i+1) - \mathbb{T}^{\text{suf}}_\mathcal{P}(i))$.*

From the definition of the product automaton cost $\mathcal{C}_\mathcal{P}$ and the suffix cost $\mathcal{C}^{\text{suf}}_\mathcal{P}$ we obtain the following result.

Lemma IV.6 (Cost of a Run)**.** *Given a run $r_\mathcal{P}$ with prefix-suffix structure and its suffix $p_0 p_1 p_2 \dots p_n p_0 p_1 \dots$, the value of the cost function $\mathcal{C}_\mathcal{P}(r_\mathcal{P})$ is equal to the cost of the suffix $\mathcal{C}^{\text{suf}}_\mathcal{P}(r_\mathcal{P})$.*

Our aim is to synthesize a run $r_\mathcal{T}$ of $\mathcal{T}$ minimizing the cost function $\mathcal{C}(r_\mathcal{T})$ and ensuring that the word produced by this run will be accepted by $\mathcal{B}$. This goal now translates to generating a run $r_\mathcal{P}$ of $\mathcal{P}$, such that the run satisfies the Büchi condition $F_\mathcal{P}$ and minimizes cost function $\mathcal{C}_\mathcal{P}(r_\mathcal{P})$. Furthermore, it is enough to consider runs in prefix-suffix structure (see Lemma IV.4). From Lemma IV.6 it follows that the whole problem reduces to finding a periodic suffix $r^{\text{suf}}_\mathcal{P} = f p_1 p_2 \dots p_n f p_1 \dots$ in $\mathcal{P}$, such that: (i) f is reachable from an initial state in $S_{\mathcal{P},0}$, (ii) $f \in F_\mathcal{P}$ (i.e., f is an accepting state), and (iii) the cost of the suffix $r^{\text{suf}}_\mathcal{P}$ is minimum among all the suffices satisfying (i) and (ii). Finally, we can find a finite prefix in $\mathcal{P}$ leading from an initial state in $S_{\mathcal{P},0}$ to the state f in the suffix $r^{\text{suf}}_\mathcal{P}$. By concatenating the prefix and suffix, we obtain an optimal run in $\mathcal{P}$. By projecting the optimal run to $\mathcal{T}$, via Proposition IV.2, we obtain a solution to our stated problem.

B. Graph Algorithm for Shortest Bottleneck Cycles

We now focus on finding an optimal suffix in the product automaton. We cast this problem as a path optimization on a graph. To do this, let us define some terminology.

A graph $G = (V, E, w)$ consists of a vertex set V, an edge set $E \subseteq V \times V$, and a weight function $w : E \to \mathbb{R}_{>0}$. A *cycle* in G is a vertex sequence $v_1 v_2 \dots v_k v_{k+1}$, such that $(v_i, v_{i+1}) \in E$ for each $i \in \{1, \dots, k\}$, and $v_1 = v_{k+1}$. Given a vertex set $S \subseteq V$, consider a cycle $c = v_1 \dots v_k v_{k+1}$ containing at least one vertex in S. Let $(i_1, i_2, \dots, i_s)$ be the ordered set of vertices in c that are elements of S (i.e., Indices with order $i_1 < i_2 < \dots < i_m$, such that $v_j \in S$ if and only if $j \in \{i_1, i_2, \dots, i_s\}$). Then, the *$S$-bottleneck length* is $\max_{\ell \in \{1,\dots,s\}} \sum_{j=i_\ell}^{i_{\ell+1}-1} w(e_j)$, where $i_{s+1} = i_1$. In words, we S-bottleneck distance is defined as follows.

Definition IV.7 (S-bottleneck length)**.** *Given a graph $G = (V, E, w)$, and a vertex set $S \subseteq V$, the S-bottleneck length of a cycle in G is the maximum distance between appearances of elements of S on the cycle. If the cycle contains no elements of S, then its S-bottleneck length is $+\infty$.*

The *bottleneck length* of a cycle is defined as the maximum length edge on the cycle [17]. In contrast, the S-bottleneck length measures distances between vertices in S.

With the terminology in place, our goal is to solve the *constrained S-bottleneck problem*:

Problem IV.8. *Given a graph $G = (V, E, w)$, and two vertex sets $F, S \subseteq V$, find a cycle in G containing at least one vertex in F, with minimum S-bottleneck length.*

Our solution, shown in the MIN-BOTTLENECK-CYCLE algorithm, utilizes Dijkstra's algorithm [17] for computing shortest paths between pairs of vertices (called SHORTEST-PATH), and a slight variation of Dijkstra's algorithm for computing shortest bottleneck paths between pairs of vertices (called SHORTEST-BOT-PATH).

SHORTEST-PATH takes as inputs a graph $G = (V, E, w)$, a set of source vertices $A \subseteq V$, and a set of destination vertices $B \subseteq V$. It outputs a distance matrix $D \in \mathbb{R}^{|A| \times |B|}$, where the entry $D(i, j)$ gives the shortest-path distance from A_i to B_j. It also outputs a predecessor matrix $P \in V^{|A| \times |V|}$, where $P(i, j)$ is the predecessor of j on a shortest path from A_i to V_j. For a vertex $v \in V$, the shortest path from v to v is defined as the shortest cycle containing v. If there does not exist a path between vertices, then the distance is $+\infty$.

SHORTEST-BOT-PATH has the same inputs as SHORTEST-PATH, but it outputs paths which minimize the maximum edge length, rather than the sum of edge lengths.

Fig. 4 (left) shows an example input to the algorithm and Fig. 4 (right) shows the optimal S-bottleneck cycle. In the algorithm, one has to be careful that cycle lengths are computed properly when $f = s_1$, $s_1 = s_2$, or $f = s_2$. This is done by setting some entries of $D_{F \to S}$ and $D_{S \to F}$ to zero in step 4, and by defining the cost differently when $f \neq s_1 = s_2$ in step 5. In the following theorem we show the correctness of the algorithm. The proof is contained in [14].

MIN-BOTTLENECK-CYCLE(G, S, F)

Input: A directed graph G, and vertex subsets F and S
Output: A cycle in G which contains at least one vertex in F and minimizes the S-bottleneck distance.

1: Shortest paths between vertices in S: $(D, P) \leftarrow$ SHORTEST-PATH(G, S, S).
2: Define a graph G_S with vertices S and adjacency matrix D.
3: Shortest S-bottleneck paths between vertices in S: $(D_{\text{bot}}, P_{\text{bot}}) \leftarrow$ SHORTEST-BOT-PATH(G_S, S, S).
4: Shortest paths from each vertex in F to each vertex in S, and from each vertex in S to each vertex in F: $(D_{F\to S}, P_{F\to S}) \leftarrow$ SHORTEST-PATH(G, F, S) and $(D_{S\to F}, P_{S\to F}) \leftarrow$ SHORTEST-PATH(G, S, F). Set $D_{F\to S}(i,j) = 0$ and $D_{S\to F}(j,i) = 0$ for all i, j such that $F_i = S_j$.
5: For each triple $(f, s_1, s_2) \in F \times S \times S$, let $C(f, s_1, s_2)$ be $D_{F\to S}(f, s_1) + D_{S\to F}(s_2, f)$, if $f \neq s_1 = s_2$, and $\max\{D_{F\to S}(f, s_1) + D_{S\to F}(s_2, f), D_{\text{bot}}(s_1, s_2)\}$, otherwise.
6: Find the triple (f^*, s_1^*, s_2^*) that minimizes $C(f, s_1, s_2)$.
7: If minimum cost is $+\infty$, then output "no cycle exists." Else, output cycle by extracting the path from f^* to s_1^* using $P_{F\to S}$, the path from s_1^* to s_2^* using P_{bot} and P, and the path from s_2^* to f^* using $P_{S\to F}$.

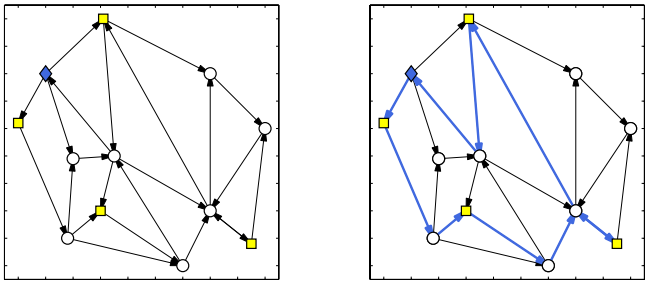

Fig. 4. A directed graph for illustrating the algorithm. The edge weights are given by the Euclidean distance. The set F is a singleton given by the blue diamond. The vertices in S are drawn as yellow squares. The thick blue edges in the right figure form a cycle with minimum S-bottleneck length.

Theorem IV.9 (MIN-BOTTLENECK-CYCLE Optimality)**.** *The* MIN-BOTTLENECK-CYCLE *algorithm solves the constrained S-bottleneck problem (Problem IV.8).*

Computational Complexity: Let n, m, n_S, and n_F, be the number of vertices (edges) in the sets V, E, S, and F, respectively. Then, the run time of the MIN-BOTTLENECK-CYCLE algorithm is $O\big((n_S+n_F)(n\log n+m+n_S^2)\big)$. Thus, in the worst-case, the run time is $O(n^3)$. For sparse graphs with $n_S, n_F \ll n$, the run time is $O\big((n_S+n_F)n\log n\big)$ [14].

C. The OPTIMAL-RUN algorithm

The solution to Problem III.1 is given by the OPTIMAL-RUN algorithm. Combining Lemma IV.4, Theorem IV.9, and Proposition IV.2, we obtain the following result.

Theorem IV.10 (Correctness of OPTIMAL-RUN)**.** *The* OPTIMAL-RUN *algorithm solves Problem III.1.*

V. EXPERIMENTS

We have implemented the OPTIMAL-RUN algorithm in simulation and on a physical road network testbed. The platform shown in Fig. 1 is a collection of roads, intersections,

OPTIMAL-RUN$(\mathcal{T}, \phi)$

Input: A weighted transition system $\mathcal{T}$, and temporal logic specification ϕ in form (1).
Output: A run in $\mathcal{T}$ which satisfies ϕ and minimizes (2).

1: Convert ϕ to a Büchi automaton $\mathcal{B}_\phi$.
2: Compute the product automaton $\mathcal{P} = \mathcal{T} \times \mathcal{B}_\phi$.
3: Compute the cycle MIN-BOTTLENECK-CYCLE$(G, S_{\mathcal{P},\pi}, F_{\mathcal{P}})$, where $G = (S_{\mathcal{P}}, \delta_{\mathcal{P}}, w_{\mathcal{P}})$.
4: Compute a shortest path from $S_{\mathcal{P},0}$ to the cycle.
5: Project the complete run (path and cycle) to a run on $\mathcal{T}$ using Proposition IV.2.

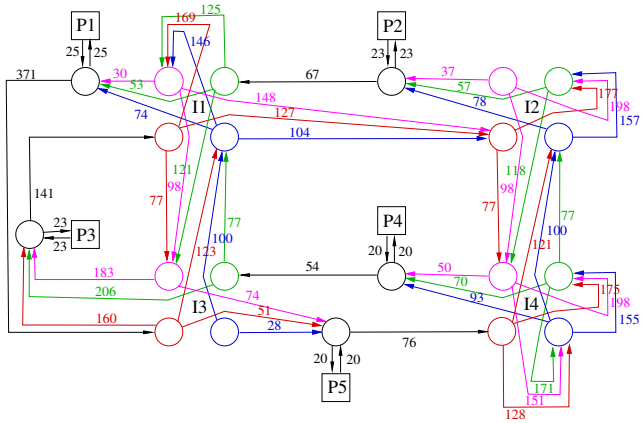

Fig. 5. The weighted transition system for the road network in Fig. 1.

and parking lots, connected by a simple set of rules (*e.g.,* a road connects two intersections, the parking lots can only be located on the side of a road). The robot is a Khepera III miniature car. The car can sense when intersections, roads, parking lots, and obstacles. The car is programmed with motion primitives allowing it to safely drive on a road, turn in an intersection, and park. The car communicates through Wi-Fi with a desktop computer, which is used as an interface to the user (*i.e.,* to enter the specification) and to perform all the computation necessary to generate the control strategy. Once computed, this is sent to the car, which executes the task autonomously by interacting with the environment.

The motion of the car in the road network is modeled as a weighted transition system (Def. II.1), and is shown in Fig. 5. The set of states Q are intersections, parking lots, and the branching points between the roads and parking lots. The transition relation R shows how the regions are connected and the transitions' labels give distances between them (measured in cm). In our testbed the robot moves at constant speed ν, and thus the distances and travel times are equivalent. For these experiments, the robot must drive on the right side of road, and it cannot make U-turns. To capture this, we model each intersection as four different states. Note that, in reality, each state in Q has associated a set of motion primitives, and the selection of a motion primitive (*e.g., go_straight*, *turn_right*) determines the transition to one unique next states. This motivates our assumption that the weighted transition system from Def. II.1 is deterministic, and therefore its inputs can be removed.

In our experiments, we considered the following task. Parking spots P2 and P3 in Fig. 5 are data upload locations (light shaded in Fig. 6) and parking spots P1, P4, and P5 are data gather locations (dark shaded in Fig. 6). The optimizing proposition is $\pi := \mathrm{P2} \vee \mathrm{P3}$, i.e. we want to minimize the time between data uploads. Both upload locations provide the same service. In contrast, each data gather location provides the robot with a different kind of data. The motion requirements can be specified as LTL formulas, where atomic propositions are simply names of the parking spots. In the formula φ of the LTL formula (1), we demand the conjunction of the following: (i) The robot keeps visiting each data gather location: $\mathbf{G}\,\mathbf{F}\,\mathrm{P1} \wedge \mathbf{G}\,\mathbf{F}\,\mathrm{P4} \wedge \mathbf{G}\,\mathbf{F}\,\mathrm{P5}$. (ii) Whenever the robot gathers data, it uploads it before doing another data gather: $\mathbf{G}\,((\mathrm{P1} \vee \mathrm{P4} \vee \mathrm{P5}) \Rightarrow \mathbf{X}\,(\neg(\mathrm{P1} \vee \mathrm{P4} \vee \mathrm{P5})\,\mathcal{U}\,(\mathrm{P2} \vee \mathrm{P3})))$. (iii) Whenever the robot uploads data, it does not visit an upload location again before gathering new data: $\mathbf{G}\,((\mathrm{P2} \vee \mathrm{P3}) \Rightarrow \mathbf{X}\,(\neg(\mathrm{P2} \vee \mathrm{P3})\,\mathcal{U}\,(\mathrm{P1} \vee \mathrm{P4} \vee \mathrm{P5})))$.

Note that the above specifications implicitly enforce $\mathbf{G}\,\mathbf{F}\,\pi$. Running the OPTIMAL-RUN algorithm, we obtain the solution as illustrated in the top three environment shots in Fig. 6. The transition system has 26 states, and the Büchi automaton has 16 states, giving a product automaton with 416 states. In the product automaton, $F_{\mathcal{P}}$ contained 52 states, and $S_{\mathcal{P},\pi}$ contained 32 states. The OPTIMAL-RUN algorithm ran in approximately 6 seconds on a standard laptop. The value of the cost function was 9.13 meters, which corresponds to a robot travel time of 3.6 minutes (i.e., the maximum travel time between uploads was 3.6 minutes). Our video submission displays the robot trajectory for this run.

The bottom three shots in Fig. 6 illustrate the situation with the same motion requirements and a further restriction saying that the robot cannot upload data in P2 after data is gathered in location P5: $\mathbf{G}\,(\mathrm{P5} \Rightarrow (\neg \mathrm{P2}\,\mathcal{U}\,\mathrm{P3}))$. In this case the Büchi automaton contained 29 states, the algorithm ran in 22 seconds, and the value of the cost function was 9.50 meters with a travel time of 3.77 minutes.

VI. CONCLUSIONS AND FUTURE DIRECTIONS

In this paper we presented a method for planning the optimal motion of a robot subject to temporal logic constraints. We considered temporal logic specifications which contain a single *optimizing proposition* that must be repeatedly satisfied. We provided a method for computing a robot trajectory that minimizes the maximum time between satisfying instances of the optimizing proposition. For future work we are looking at ways to extend the cost functions that can be optimized. In particular, we are looking at extensions to more general types of patrolling problems. Another interesting direction is the extension to multiple robots and to non-deterministic transition systems.

Acknowledgements: We thank Y. Chen and S. Birch at Boston University for their work on the road network.

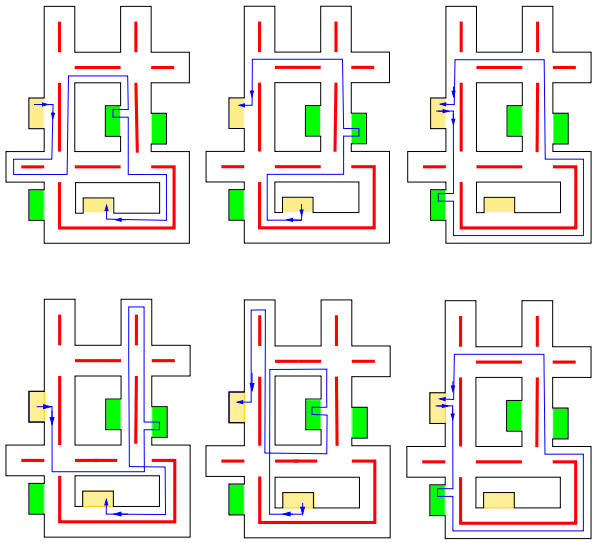

Fig. 6. The robot trajectories (blue arrows) for the data gathering mission. Green (dark shaded) areas are data-gathering locations, and yellow (light shaded) areas are upload locations. The bottom three figures show the new robot trajectory when we restrict data upload to location P3 (the bottom yellow location) after each data-gather at P5 (the rightmost green location).

REFERENCES

[1] M. Antoniotti and B. Mishra, "Discrete event models + temporal logic = supervisory controller: Automatic synthesis of locomotion controllers," in *Proc ICRA*, 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 *Proc CDC*, Paradise Island, Bahamas, 2004, pp. 153–158.

[3] M. M. Quottrup, T. Bak, and R. Izadi-Zamanabadi, "Multi-robot motion planning: A timed automata approach," in *Proc ICRA*, New Orleans, LA, 2004, pp. 4417–4422.

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

[5] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for mobile robots," in *Proc ICRA*, Barcelona, Spain, Apr. 2005, pp. 2032–2037.

[6] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning for dynamical systems," in *Proc CDC*, Shanghai, China, 2009, pp. 5997–6004.

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

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

[9] J. Barnat, L. Brim, and P. Ročkai, "DiVinE 2.0: High-performance model checking," in *High Performance Computational Systems Biology*. IEEE Computer Society Press, 2009, pp. 31–32.

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

[11] S. Karaman and E. Frazzoli, "Vehicle routing problem with metric temporal logic specifications," in *Proc CDC*, Cancún, México, 2008, pp. 3953–3958.

[12] 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.

[13] C. Belta and L. Habets, "Control of a class of nonlinear systems on rectangles," *IEEE Trans Automatic Ctrl*, vol. 51, no. 11, pp. 1749–1759, 2006.

[14] S. L. Smith, J. Tůmová, C. Belta, and D. Rus, "Optimal path planning under temporal constraints," July 2010, available at http://arxiv.org/abs/1007.2212.

[15] R. Gerth, D. Peled, M. Vardi, and P. Wolper, "Simple on-the-fly automatic verification of linear temporal logic," in *Protocol Specification, Testing and Verification*. Chapman & Hall, 1995, pp. 3–18.

[16] P. Gastin and D. Oddoux, "Fast LTL to Büchi automata translation," in *Conf. on Computer Aided Verification*, ser. Lect. Notes Comp. Science, no. 2102. Springer Verlag, 2001, pp. 53–65.

[17] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, 4th ed., ser. Algorithmics and Combinatorics. Springer Verlag, 2007, vol. 21.