Language-Guided Sampling-based Planning using Temporal Relaxation

Francisco Penedo[1], Cristian-Ioan Vasile[2], and Calin Belta[1]

[1] Boston University, Boston, MA, U.S.A.
{franp, cbelta}@bu.edu
[2] Massachusetts Institute of Technology, Cambridge, MA, U.S.A.
cvasile@mit.edu

Abstract. In this paper, we focus on robot motion planning from timed temporal logic specifications. We propose a sampling-based algorithm and an associated language-guided biasing scheme. We leverage the notion of temporal relaxation of time-window temporal logic formulae (TWTL) to reformulate the temporal logic synthesis problem into an optimization problem. Our algorithm exhibits an exploration-exploitation structure, but retains probabilistic completeness. Moreover, if the problem does not have a solution due to time constraints, the algorithm returns a candidate path that satisfies a minimally relaxed version of the specification. The path may inform operators about timing problems with the specification or the system. We provide simulations to highlight the performance of the proposed algorithm.

1 Introduction

Motion planning is a fundamental problem in robotics. The objective is to generate control policies for a robot to drive it from an initial state to a goal region in its state space under kino-dynamic constraints [22]. Even without considering dynamics, the problem becomes increasingly difficult in high dimensions, and it has been shown to be PSPACE-complete [7]. Cell decomposition, potential fields and navigation functions [9] are some of the most used techniques to solve the problem. However, they scale poorly with the dimension of the state space and number of obstacles. To overcome these limitations, a class of algorithms was developed relying on randomly sampling the configuration space of the robot and planning local motions between these samples. Probabilistic Roadmaps [18] and Rapidly Exploring Random Trees [22] are among the most widely known examples, along with their asymptotically optimal variants, PRM* and RRT* [16].

Robots are increasingly required to perform complex tasks, where correctness guarantees, such as safety and liveness in human-robot teams and autonomous driving, are critical. One approach is to encode the tasks as temporal logic specifications and leverage formal methods techniques to generate control policies that are correct by construction [4]. As opposed to traditional methods restricted to reach-avoid setups, these frameworks are able to express more complex tasks such as sequencing (e.g., "Reach A, then B"), convergence ("Go to A and stay

there forever"), persistent surveillance ("Visit A, B, and C infinitely often"), and more complex combinations of the above. Temporal logics, such as Linear Temporal Logic (LTL), Computational Tree Logic (CTL), and μ-calculus, and their probabilistic versions (PLTL, PCTL), have been shown to be useful as formal languages for motion planning [20, 38, 6, 15, 10]. Model-checking and automata game techniques were adapted [20, 8] to generate control policies for finite models of robot motion. These models were obtained through an abstraction process that partitions the robot configuration space and captures the ability of the robot to steer between regions in the partition [5]. As a result, these algorithms suffer from the same scalability issues as the cell-based decomposition methods.

Some applications additionally require time constraints [30, 27, 12]. For example, consider the following task: "Visit A, B, and C in this order. Perform action a for 2 time units at A within 10 time units. Then, perform b for 3 time units at B within 6 time units. Finally, in the time window $[3, 9]$ after b is finished, perform c for 1 time unit at C. All three actions must be finished within 15 time units." Tasks with explicit time constraints may be expressed using bounded linear temporal logic (BLTL) [31, 14], metric temporal logic (MTL) [19], signal temporal logic (STL) [24], and time-window temporal logic (TWTL) [35, 1, 36].

A natural approach to generate control strategies from rich task specifications for robots with large state spaces is to combine sampling-based techniques with automata-based synthesis methods. The existing works in this area show that synthesis algorithms from specifications given in μ-calculus [15, 17] and LTL [34] can be adapted to scale incrementally with the graph constructed during the sampling process. In all of these, sampling is performed independently of the specification. Hierarchical planning frameworks were proposed in [28, 25], where a higher level planner performs a discrete search in a partition of the workspace of the robot that guides a lower level sampling-based planner in the configuration space. The idea of language-guided synthesis was also explored in [2], where an iterative partition-refinement algorithm is proposed together with cell-to-cell feedback controllers, and in [37], where candidate discrete plans are enumerated, and then motion plans are generated using an optimization-based procedure.

In this paper, we propose a language-guided sampling-based method to generate robot control policies satisfying tasks expressed as TWTL formulae. We leverage the notion of temporal relaxation [35] to reformulate a temporal logic motion problem as an optimization problem, where the objective is to minimize the temporal relaxation of the specification. This approach has two advantages: (a) the growth of the sampling graph is biased towards satisfaction of a relaxed version of the specification without initially taking into account deadlines, and (b) after an initial policy has been found, sampling can be focused on the parts of the plan that need to be improved. The two stages can be thought of as exploration-exploitation phases, where initially candidate solutions are found, and then focused local sampling is performed on the parts that need to be improved in order to satisfy the specification, i.e. time constraints. As a byproduct, a satisfying policy with respect to a minimally relaxed version of the specification may be returned when the original problem does not have a solution. Such

a policy may inform operators about timing problems in the specification or system (i.e., robot dynamics and/or environment). The algorithm uses annotated finite state automata [35, 36, 1] to represent all possible temporal relaxations of a TWTL formulae. Lastly, we prove that our solution is probabilistically complete.

As opposed to [35], we do not assume a finite model of the system, and propose a sampling-based approach. Although the synthesis algorithm in [35] is more general (w.r.t. the range of specifications it can handle), it is not incremental and thus not suitable for use with sampling-based techniques. Moreover, we define a new temporal relaxation measure over τ-relaxations of TWTL formulae called *linear ramp temporal relaxation* that is better suited for incremental computation. Other studies have investigated minimal violations of LTL fragments [29, 33, 32, 21, 26, 13]. Our approach differs from [29, 33, 32] because we consider explicit time constraints in the specification, and a different semantics for relaxation of a specification. The minimum violation policies strive to minimize the duration that the specification is violated, i.e., satisfaction is preempted. Temporal relaxation on the other hand minimizes the deviation from the deadlines in the specification, and does not allow task interruption. In [13], the specifications were relaxed by minimally revising symbols associated with the transitions of the Büchi automata. Both [21, 26] modify LTL to accomodate partial satisfaction without considering explicit time bounds. Due to the fragment of TWTL we allow, our work is related to the scheduling literature, such as [23, 11]. However, these works do not allow partial satisfaction. In [3], explicit time constraints, as well as first order quantifiers, are allowed in the specification, but no partial completion is considered. A different approach is taken by [39] using the concept of resources to impose soft constraints in the specification.

Notation: Given $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^n$, $n \geq 2$, the relationship $\mathbf{x} \sim \mathbf{x}'$, where $\sim \in \{<, \leq, >, \geq\}$, is true if it holds pairwise for all components. $\mathbf{x} \sim a$ denotes $\mathbf{x} \sim a\mathbf{1}_n$, where $a \in \mathbb{R}$ and $\mathbf{1}_n$ is the n-dimensional vector of all ones. Let S be a finite set. We denote the cardinality and the power set of S by $|S|$ and 2^S, respectively.

2 Preliminaries

2.1 Time-Window Temporal Logic (TWTL)

In this paper, we use Time-Window Temporal Logic (TWTL) as a specification language for temporal properties with time constraints. For details see [35] and [36, 1] for applications to robotics. TWTL is a linear-time logic encoding sets of discrete-time sequences with values in a finite alphabet. The syntax of TWTL formulae defined over a set of atomic propositions Π is

$$\phi ::= \mathcal{H}^d s \,|\, \mathcal{H}^d \neg s \,|\, \phi_1 \wedge \phi_2 \,|\, \phi_1 \vee \phi_2 \,|\, \neg \phi_1 \,|\, \phi_1 \cdot \phi_2 \,|\, [\phi_1]^{[a,b]}$$

where s is either the "true" constant $\top$ or an atomic proposition in Π; $\wedge$, $\vee$, and $\neg$ are the conjunction, disjunction, and negation Boolean operators, respectively; $\cdot$ is the concatenation operator; $\mathcal{H}^d$ with $d \in \mathbb{Z}_{\geq 0}$ is the *hold* operator; and $[\,]^{[a,b]}$ with $0 \leq a \leq b$ is the *within* operator. The semantics is defined with respect

to finite (output) words $\mathbf{o} = o_0 o_1 ... o_k$ over the set 2^{Π}. The Boolean operators retain their usual semantics. The *hold* operator $\mathcal{H}^d s$ specifies that an atomic proposition $s \in \Pi$ should be serviced (satisfied) for d time units (i.e., $\mathbf{o} \models \mathcal{H}^d s$ if $o_t = s \; \forall t \in [0, d]$). For convenience, if $d = 0$ we simply write s and $\neg s$ instead of $\mathcal{H}^0 s$ and $\mathcal{H}^0 \neg s$, respectively. The *within* operator $[\phi]^{[a,b]}$ bounds the satisfaction of ϕ within $[a, b]$ time window (i.e., $\mathbf{o} \models [\phi]^{[a,b]}$ if $\exists k \in [0, b-a]$ s.t. $\mathbf{o}' \models \phi$ where $\mathbf{o}' = o_{a+k} \dots o_b$). Lastly, the concatenation of ϕ_i and ϕ_j (i.e., $\phi_i \cdot \phi_j$) specifies that first ϕ_i must be satisfied and then immediately ϕ_j must be satisfied. The satisfaction of a TWTL formula by a word can be decided within bounded time.

The notion of *temporal relaxation* of a TWTL formula was introduced in [35, 1]. To illustrate the main ideas, consider the following TWTL formula:

$$\phi_1 = [\mathcal{H}^1 A]^{[0,2]} \cdot [\mathcal{H}^3 B \wedge [\mathcal{H}^2 C]^{[0,4]}]^{[1,8]}, \tag{1}$$

which reads as "Perform task A of duration 1 within 2 time units. Then, within the time interval $[1, 8]$ perform tasks B and C of durations 3 and 2, respectively. Furthermore, C must be finished within 4 time units from the start of B." If ϕ_1 cannot be satisfied, one way to relax ϕ_1 is to extend the deadlines for the time windows captured by the *within* operators:

$$\phi_1(\boldsymbol{\tau}) = [\mathcal{H}^1 A]^{[0,2+\tau_1]} \cdot [\mathcal{H}^3 B \wedge [\mathcal{H}^2 C]^{[0,4+\tau_2]}]^{[1,8+\tau_3]}, \tag{2}$$

where $\boldsymbol{\tau} = (\tau_1, \tau_2, \tau_3) \in \mathbb{Z}^3$. However, the choice of $\boldsymbol{\tau}$ must preserve the feasibility of the formula, i.e., the following must hold for $\phi_1(\boldsymbol{\tau})$: (i) $2+\tau_1 \geq 1$, (ii) $4+\tau_2 \geq 2$, and (iii) $7+\tau_3 \geq \max\{3, 4+\tau_2\}$. Note that $\boldsymbol{\tau}$ may be non-positive. In such cases, $\phi_1(\boldsymbol{\tau})$ becomes a stronger specification than ϕ_1, which implies that the sub-tasks are performed earlier than their actual deadlines.

Definition 1 (Feasible TWTL formula). *A TWTL formula ϕ is called feasible if the time window corresponding to each* within *operator is greater than the duration of the corresponding enclosed task expressed via the* hold *operators.*

Let ϕ be a TWTL formula. Then, a $\tau-$relaxation of ϕ is defined as follows:

Definition 2 ($\tau-$Relaxation of ϕ). *Let $\boldsymbol{\tau} \in \mathbb{Z}^m$, where m is the number of* within *operators contained in ϕ. A τ-relaxation of ϕ is a* feasible *TWTL formula $\phi(\boldsymbol{\tau})$, where each subformula of the form $[\phi_i]^{[a_i,b_i]}$ is replaced by $[\phi_i]^{[a_i,b_i+\tau_i]}$.*

Clearly, for any ϕ, we have $\phi(\mathbf{0}) = \phi$. Moreover, let $\boldsymbol{\tau'}, \boldsymbol{\tau''} \in \mathbb{Z}^m$ such that $\phi(\boldsymbol{\tau'})$ and $\phi(\boldsymbol{\tau''})$ are feasible relaxations, where m is the number of *within* operators in ϕ. Note that if $\boldsymbol{\tau'} \leq \boldsymbol{\tau''}$, then $\mathbf{o} \models \phi(\boldsymbol{\tau'}) \Rightarrow \mathbf{o} \models \phi(\boldsymbol{\tau''})$.

Let ϕ be a TWTL formula and $\phi(\boldsymbol{\tau})$ its τ-relaxation, where $\boldsymbol{\tau} \in \mathbb{Z}^m$ and m is the number of *within* operators contained in ϕ. We denote by $\mathfrak{I}_{\boldsymbol{\tau}}(\phi) = (\tau_1, \dots, \tau_m)$ the ordered set of deadline deviations.

Definition 3. *Given an output word $\mathbf{o}$, we say that $\mathbf{o}$ satisfies $\phi(\infty)$, i.e., $\mathbf{o} \models \phi(\infty)$, if and only if $\exists \boldsymbol{\tau'} < \infty$ s.t. $\mathbf{o} \models \phi(\boldsymbol{\tau'})$.*

Similarly, if $\boldsymbol{\tau} < \infty$, then $\mathbf{o} \models \phi(\boldsymbol{\tau}) \Rightarrow \mathbf{o} \models \phi(\infty)$, $\forall \; \boldsymbol{\tau}$.

Definition 4 (Concatenation Form). *A TWTL formula ϕ is in concatenation form (CF) if and only if $\phi = \phi_1 \cdot \phi_2 \cdot \ldots \cdot \phi_n$, where ϕ_i are TWTL formulae that do not contain concatenation operators.*

In the following, we use ϕ_i to refer to the ith subformula of a formula in CF in the same way as in the previous definition. We also denote as $\phi^j = \phi_1 \cdot \ldots \cdot \phi_j$ the subformula of ϕ that includes all subformulae $\phi_1, ..., \phi_j$, which is also in CF.

2.2 Specification Automaton and System Abstraction

In this section we provide a short presentation of the mathematical objects defined in [35] that we will be making use of.

Definition 5 (Deterministic Finite State Automaton). *A deterministic finite state automaton (DFA) is a tuple $\mathcal{A} = (S_\mathcal{A}, s_0, \mathfrak{A}, \delta_\mathcal{A}, F_\mathcal{A})$, where $S_\mathcal{A}$ is a finite set of states, $s_0 \in S_\mathcal{A}$ is the initial state, $\mathfrak{A}$ is the input alphabet, $\delta_\mathcal{A} : S_\mathcal{A} \times \mathfrak{A} \to S_\mathcal{A}$ is the transition function, and $F_\mathcal{A} \subseteq S_\mathcal{A}$ is the accepting set.*

A trajectory of the DFA $\mathbf{s} = s_0 s_1 \ldots s_{n+1}$ is generated by a sequence of symbols $\boldsymbol{\sigma} = \sigma_0 \sigma_1 \ldots \sigma_n$ if $s_0 \in S_\mathcal{A}$ is the initial state of $\mathcal{A}$ and $s_{k+1} = \delta_\mathcal{A}(s_k, \sigma_k)$ for all $0 \leq k \leq n$. The function $\delta^*_\mathcal{A} : \mathfrak{A}^* \to S_\mathcal{A}$ is defined such that $s_{n+1} = \delta^*_\mathcal{A}(\boldsymbol{\sigma})$. An input word $\boldsymbol{\sigma}$ is accepted by a DFA $\mathcal{A}$ if $\delta^*_\mathcal{A}(\boldsymbol{\sigma}) \in F_\mathcal{A}$.

Definition 6 (Transition System). *A transition system (TS) is a tuple $\mathcal{T} = (V, x_0, E, \Pi, h)$, where V is a finite set of states, $x_0 \in V$ is the initial state, $E \subseteq V \times V$ is a set of transitions, Π is a set of properties (atomic propositions), and $h : V \to 2^\Pi$ is a labeling function.*

A trajectory of the system is a finite or infinite sequence of states $\mathbf{x} = x_0 x_1 \ldots$ such that $(x_k, x_{k+1}) \in E$ for all $k \geq 0$. A state trajectory $\mathbf{x}$ generates an *output trajectory* (or word) $\mathbf{o} = o_0 o_1 \ldots$, where $o_k = h(x_k)$ for all $k \geq 0$.

Definition 7 (Product Automaton). *Given $\mathcal{T} = (V, x_0, E, \Pi, h)$ and $\mathcal{A} = (S_\mathcal{A}, s_0, \mathfrak{A}, \delta_\mathcal{A}, F_\mathcal{A})$, their product automaton, denoted by $\mathcal{P} = \mathcal{T} \times \mathcal{A}$, is a tuple $\mathcal{P} = (S_\mathcal{P}, p_0, \Delta_\mathcal{P}, F_\mathcal{P})$, where $S_\mathcal{P} = V \times S_\mathcal{A}$ is the set of states, $p_0 = (x_0, s_0)$ is the initial state, $\Delta_\mathcal{P} = \{((x, s), (x', s')) \mid (x, x') \in E \wedge s' = \delta_\mathcal{A}(s, h(x'))\}$ is the set of transitions, and $F_\mathcal{P} = V \times F_\mathcal{A}$ is the set of accepting states of $\mathcal{P}$.*

A trajectory of $\mathcal{P}$ is a sequence $\mathbf{p} = p_0 \ldots p_{n+1}$ such that $(p_k, p_{k+1}) \in \Delta_\mathcal{P}$ for all $0 \leq k \leq n$. A trajectory $\mathbf{p} = (x_0, s_0)(x_1, s_1) \ldots$ of $\mathcal{P}$ is accepted if and only if $s_0 s_1 \ldots$ is accepted by $\mathcal{A}$. A trajectory of $\mathcal{T}$ obtained from an accepting trajectory of $\mathcal{P}$ satisfies the given specification encoded by $\mathcal{A}$.

3 Problem Formulation

Consider a dynamical system $\Sigma(x_0) : \quad x_{k+1} = f(x_k, u_k)$, where $x_k \in \mathbb{R}^n$ and $u_k \in U \subset \mathbb{R}^m$ are the state and control input at time k, respectively, U is

the control space, and x_0 is the initial state. Let $W \subset \mathbb{R}^n$ be a convex region denoting the workspace, $O \subset W$ the obstacle set, $W_{free} = W \setminus O$ the free space, $\Pi = \{\pi_i | i = 1, ..., p\}$ a set of atomic propositions, and $\mathcal{L} : W \to 2^{\Pi}$ the state labeling function. Let $\mathbf{x} = x_0 x_1 \dots$ be a trajectory of $\Sigma(x_0)$. We say that $\mathbf{x}$ is collision-free if for all $k \geq 0$ and $\lambda \in [0, 1]$, $\lambda x_k + (1 - \lambda) x_{k+1} \in W_{free}$. The output word generated by $\mathbf{x}$ is $\mathbf{o} = o_0 o_1 \dots$, with $o_k = \mathcal{L}(x_k)$. System $\Sigma(x_0)$ under control signal $\mathbf{u} = u_0 u_1 \dots$ is said to satisfy a TWTL formula ϕ if $\mathbf{o} \models \phi$.

Problem 1. Given a TWTL formula ϕ over Π and an initial state $x_0 \in W$, find a control policy $\mathbf{u}^*$ with control inputs in U such that the trajectory of the closed-loop system $\Sigma(x_0)$ under policy $\mathbf{u}^*$ is collision-free and satisfies ϕ.

We can formulate an optimization problem equivalent to Problem 1 by taking advantage of temporal relaxations of TWTL formulae in the following way:

Definition 8. *The linear ramp temporal relaxation (LRTR) of a τ-relaxed formula $\phi(\boldsymbol{\tau})$ is defined as:*

$$|\phi(\boldsymbol{\tau})|_{LRTR} = \sum_{j \in \mathfrak{I}_{\boldsymbol{\tau}}(\phi)} \max\{0, \tau_j\}. \tag{3}$$

Problem 2. Let ϕ be a TWTL formula over Π, and $\phi(\boldsymbol{\tau})$ its τ-relaxation. Consider the optimization problem

$$\min |\phi(\boldsymbol{\tau})|_{LRTR} \quad \text{s.t.} \quad \exists \mathbf{u}^* : \Sigma(x_0) \text{ under } \mathbf{u}^* \text{ satisfies } \phi(\boldsymbol{\tau}). \tag{4}$$

If the minimum obtained from (4) is equal to 0, find a corresponding policy $\mathbf{u}^*$.

Note that this notion of temporal relaxation, LRTR, is different from the one defined in [35], where it is denoted by $|\cdot|_{TR}$. Intuitively, LRTR measures the accumulation of positive deviations from the deadlines, as opposed to the worst deviation in TR. In both, when the temporal relaxation is 0, the unrelaxed formula is satisfied. The almost linear structure and monotonicity of LRTR allows the algorithm proposed in this paper to be incremental.

Example. Consider the workspace in the bottom right of Fig. 1, with areas of interest A, B and C, and several obstacles. The specification is "Perform tasks at A, B, and C, in this order, within time intervals [0, 2], [0, 3], and [0, 4] from the end of the previous task, respectively." The corresponding TWTL formula is $\phi_{simple} = [A]^{[0,2]} \cdot [B]^{[0,3]} \cdot [C]^{[0,4]}$. The dynamics are given by $x_{k+1} = x_k + u_k$, with the state space equal to the workspace, $||u_k|| \leq 0.75$, and $x_0 = (1, 3)$.

4 Solution

We first present an overview of our approach to solve Prob. 1. Consider the algorithm in [35], which requires a discretization of W. First, a finite TS associated with the system Σ and the discretization of W is computed. Then, the formula ϕ is translated into an annotated DFA and the product automaton of the TS and

the DFA is constructed. Finally, a path with minimum temporal relaxation is found in the product automaton using a shortest path algorithm. If ϕ is satisfied by this path, the associated control sequence is a solution to the problem.

In our approach, we assume that it is not possible to obtain a discretization of W. Instead, we incrementally construct a finite transition system, $\mathcal{T} = (V, x_0, E, \Pi, h)$, where $V \subset W_{free}$, $h = \mathcal{L}$ and (V, E) is a tree, using a sampling-based algorithm similar to RRT. Specifically, at each iteration a random sample $x \in W_{free}$ is added to V. Then, a feasible transition (according to Σ) from a state in V to x is included in E. The transition is selected so that a cost function for the path from x_0 to x is minimized. At the same time, the product automaton $\mathcal{P} = \mathcal{T} \times \mathcal{A}_\infty$ is also computed incrementally, where $\mathcal{A}_\infty$ is the annotated DFA obtained from ϕ. The path with minimum temporal relaxation in $\mathcal{P}$ is updated at each iteration by comparing the current best with the (single) path added.

In the following, we assume the following about ϕ: (1) negation operators appear only in front of atomic propositions; (2) all sub-formulae of ϕ correspond to unambiguous languages (no proper subset of the language is a prefix language of the difference). Both are required to translate the formula into a DFA, but are not overly restrictive (see [35] for details). Additionally, we require (3) ϕ to be in concatenation form. This last assumption allows us to divide the specification in tasks and induces a measure of progress towards an objective. It forms the basis of the language-guided sampling that is presented in detail in Sec. 4.3. The main limitation imposed by (3) is, intuitively, to constrain the specification to only describe "tasks" without "subtasks". For example, $\phi = [\mathcal{H}^3 A]^{[0,10]} \cdot [\mathcal{H}^2 B]^{[0,5]}$ (two high level hold tasks) satisfies (3), but $\phi = [[\mathcal{H}^3 A]^{[0,10]} \cdot [\mathcal{H}^2 B]^{[0,5]}]^{[0,20]}$ (a high level task with two hold substasks) does not.

Example (cont). The DFA corresponding to the specification ϕ_{simple} is shown in the upper left corner of Fig. 1. In the center of the figure, we show the grown TS (represented by the dots and arrows) as well as the product with the DFA.

4.1 Algorithm description

In Alg. 1 we present the procedure to incrementally generate a TS from a description of the workspace and a specification. The algorithm takes as inputs the TWTL formula ϕ, and the initial state $x_0 \in W_{free}$. The description of the workspace is implicit in the functions that use it.

The procedure starts by initializing the needed data structures in line 1: the formula is translated to the DFA $\mathcal{A}_\infty$ and the TS $\mathcal{T}$ is constructed with just the initial state. The TS is then grown for a number of iterations (line 2).

A sample pair of a state to expand in the TS and a state in the workspace is generated in line 3. The state that will be considered to be added to the TS, x_{new}, is computed by the steering function *Steer* in line 4. If the path to the new state is feasible (line 5), it is added to the TS (line 8). In order to select the best node to connect x_{new} to, we look at the set of nodes within steering distance in the same state of the DFA as the node to expand (line 6). Then, a node with best cost and a feasible path to x_{new} is chosen from that set (line 7). After adding the new node, other nodes within sterring distance that can be reached from x_{new}

Algorithm 1: Algorithm

Input: ϕ – TWTL formula in concatenation form, x_0 – initial point
Output: $\mathcal{T}$ – transition system

1 $\mathcal{A}_\infty \leftarrow Translate(\phi)$; $\mathcal{T} = (V, x_0, E, \Pi, h) \leftarrow (\{x_0\}, x_0, \emptyset, \Pi, \mathcal{L})$
2 **for** $i = 1, ..., n$ **do**
3 $\quad x_{exp}, x_{ran} \leftarrow Sample()$
4 $\quad x_{new} \leftarrow Steer(x_{exp}, x_{ran})$
5 $\quad$ **if** $ColFree(x_{exp}, x_{new})$ **then**
6 $\quad\quad V_{near} \leftarrow \{x \in V : \delta^*_{\mathcal{A}_\infty}(\vec{\sigma}) = \delta^*_{\mathcal{A}_\infty}(\vec{\sigma}_{exp})), \|x - x_{new}\| \leq d_{steer}\}$
7 $\quad\quad x_{min} \leftarrow \text{argmin}_{\{x \in V_{near} : ColFree(x, x_{new})\}} \{Cost_{\mathcal{A}_\infty}(\vec{x})\}$
8 $\quad\quad V \leftarrow V \cup \{x_{new}\}$; $E \leftarrow E \cup \{(x_{min}, x_{new})\}$
9 $\quad\quad V_{next} \leftarrow \{x \in V : \delta^*_{\mathcal{A}_\infty}(\vec{\sigma}) \in \delta_{\mathcal{A}}(\delta^*_{\mathcal{A}}(\vec{\sigma}_{exp}), o_{new}), \|x - x_{new}\| \leq d_{steer}\}$
10 $\quad\quad$ **foreach** $x_{next} \in V_{next}$ **do**
11 $\quad\quad\quad$ **if** $Cost_{\mathcal{A}_\infty}(\vec{x}_{next}) > Cost_{\mathcal{A}_\infty}(\vec{x}_{new} \oplus x_{next}) \wedge ColFree(x_{new}, x_{next})$ **then**
12 $\quad\quad\quad\quad E \leftarrow (E \setminus \{(Parent(x_{next}), x_{next})\}) \cup \{(x_{new}, x_{next})\}$
13 **return** $\mathcal{T}$

(regarding both state consistency in the DFA and path feasibility) are considered for rewiring. If the path through x_{new} has better cost than their current one, the tree is rewired (lines 9-12, see more details on rewiring in Sec. 4.2).

Some primitive functions are assumed to be available in Alg. 1. *Sample* will be discussed in Sec. 4.3. *Steer* computes a state close to x_{ran} within steering distance of x_{exp} and $ColFree$ checks if the path between two points is free of obstacles; a more in depth discussion of both can be found in [22]. Let $x \in V$ be a state in $\mathcal{T}$. We denote by $\vec{x}$ and $\vec{o}$ the (unique) path in $\mathcal{T}$ from the initial state x_0 to x and its corresponding output word, respectively. The function $Parent : V \to V \cup \{\bowtie\}$ returns the parent of x in the TS, with $\bowtie = Parent(x_0)$. We denote by $\vec{x} \oplus x_{new}$ the path resulting from appending the node x_{new} to $\vec{x}$. The cost associated with $\vec{x}$ is given by $Cost_{\mathcal{A}_\infty}(\vec{x})$. See Sec. 4.2 for details.

Note that even though we describe the algorithm as building the incremental TS $\mathcal{T}$, it is easy to see that the product automaton $\mathcal{P}$ is also being incrementally built: it is only necessary to consider each state x of the TS as augmented with the corresponding DFA state, $\delta^*_{\mathcal{A}_\infty}(\vec{\sigma})$. From an implementation point of view, computing the product automaton does not incur in a performance penalty, since the number of states is equal to that of the TS. In fact, the function $\delta^*_{\mathcal{A}_\infty}$ is now immediately available for all states of $\mathcal{T}$ and only the transition function $\delta_{\mathcal{A}_\infty}$ needs to be computed for each new sample. In the following, we assume the algorithm incrementally builds the product automaton $\mathcal{P}$ explicitly.

4.2 Cost function

Each path $\vec{x}_{L-1} = \{x_i\}_{i=0}^{L-1}$ in the TS has a cost represented by the function $Cost_{\mathcal{A}_\infty}$. We first look for the largest j such that $\vec{x}_{L-1} \models \phi^j(\boldsymbol{\tau})$, for some $\boldsymbol{\tau}$.

Let $\boldsymbol{\tau}^*$ be the tightest τ-relaxation of ϕ^j that $\vec{x}_{L-1}$ satisfies, i.e., the one that minimizes $\left|\phi^j(\boldsymbol{\tau})\right|_{LRTR}$. Then, we obtain the shortest subpath of $\vec{x}_{L-1}$ that satisfies $\phi^j(\boldsymbol{\tau}^*)$, $\vec{x}_{S-1} = \{x_i\}_{i=0}^{S-1}$. Finally, we can define the cost as follows:

$$Cost_{\mathcal{A}_\infty}(\vec{x}_{L-1}) = L - S + \left|\phi^j(\boldsymbol{\tau}^*)\right|_{LRTR}. \tag{5}$$

It is immediate to see that finding a path that satisfies $\phi(\boldsymbol{\tau})$ with minimum cost provides a solution for Prob. 2. Note that the cost increases by 1 with each node added to the path that does not render a longer subformula ϕ^{j+1} true. Moreover, when a node is added such that ϕ^{j+1} is satisfied, we only need to consider the subpath associated with ϕ_{j+1} in order to obtain the new $\boldsymbol{\tau}^*$. This leads to the following definition of $Cost_{\mathcal{A}_\infty}$, equivalent to the previous one:

$$\begin{aligned} &Cost_{\mathcal{A}_\infty}(\vec{x}_{L-1}) = \\ &\quad \begin{cases} Cost_{\mathcal{A}_\infty}(\vec{x}_{S-1}) + \max\{Cost_{\mathcal{A}_\infty}(\vec{x}_{L-2}) - Cost_{\mathcal{A}_\infty}(\vec{x}_{S-1}) + 1 - b_j, 0\}, \text{ if } \mathfrak{P}(\vec{x}) \\ Cost_{\mathcal{A}_\infty}(\vec{x}_{L-2}) + 1, \text{ otherwise} \end{cases} \\ &\mathfrak{P}(\vec{x}) = \vec{x}_{L-1} \models \phi^j(\boldsymbol{\tau}) \ \wedge \ \vec{x}_{L-2} \not\models \phi^j(\boldsymbol{\tau}) \ \wedge \ \vec{x}_{S-1} \models \phi^{j-1}(\boldsymbol{\tau}) \ \wedge \ \vec{x}_{S-2} \not\models \phi^{j-1}(\boldsymbol{\tau}) \end{aligned} \tag{6}$$

In the equation above, b_j is the upper bound of the time window for subformula ϕ_j. This cost function is easy to implement by storing the cost of $\vec{x}$ in the node x and obtaining the cost of new nodes recursively using Eq. (6).

Alg. 1 tries to keep minimum cost paths between the nodes of the tree. In lines 9-12, a rewiring process is executed after the tree has been extended. This rewiring is structurally very similar to the one performed by the RRT* algorithm [16]. However, while RRT* considers nodes nearby the last added node for rewiring, we add a consistency condition so that only nodes corresponding to a successor state in the product automaton will be considered for rewiring (see line 9). Moreover, the cost function deviates from the usual continuous additive cost functions considered in RRT*. In Sec. 4.4, we discuss the implications of rewiring from a (probabilistic) completeness and optimality point of view.

4.3 Sampling

The *Sample* function generates a random state to add to the TS and selects the candidate node to connect it to. The usual sampling function for RRT-type of algorithms requires the samples to be from the free space, W_{free}. However, as was pointed out when describing the algorithm, not only the TS $\mathcal{T}$, which has states in W_{free}, is constructed, but also the product automaton $\mathcal{P} = \mathcal{T} \times \mathcal{A}_\infty$. Therefore, we also need to sample a state from the DFA's set of states $S_{\mathcal{A}_\infty}$. In Fig. 1 we explicitly show the product automaton in layers corresponding to the different states of the DFA to illustrate this idea.

The straightforward approach to sampling would proceed as follows: obtain a random sample $p_{ran} = (x_{ran}, s_{ran})$ from $W_{free} \times S_{\mathcal{A}_\infty}$. Then, find the nearest state p_{exp} in $\mathcal{P}$ that can be connected to the sample p_{ran} (see below). However, in order to simplify the computation, first we sample from $S_{\mathcal{A}_\infty}$ a DFA state

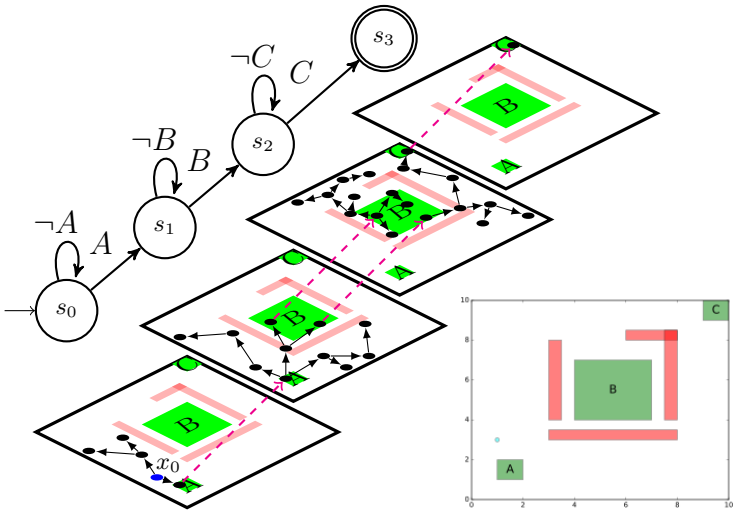


Fig. 1. A schematic representation of Alg. 1 applied to the example problem. Regions A, B and C are green, obstacles are red and the initial point x_0 is blue. The DFA corresponding to ϕ_{simple} is plotted in the upper left part. In each layer we plot in black the nodes of the product automaton that correspond to the accompanying DFA state. Black arrows represent transitions that do not change the DFA state, while magenta arrows indicate that the next node is in a new DFA state.

s_{exp} that restricts the product states we need to consider for extension. Then, a sample x_{ran} from W_{free} is obtained. Finally, p_{exp} is selected as before (with the DFA part now fixed) and s_{ran} is determined by $\delta_{\mathcal{A}_\infty}(\delta^*_{\mathcal{A}_\infty}(\vec{o}_{exp}), o_{ran})$. This procedure is sound in the sense that it does not generate "naive" random trees (i.e., biased towards the initial state, see [22]), since the DFA states are finite.

Algorithm 2: Sample

Input: $\mathcal{T} = (V, x_0, E, \Pi, h)$ – Current TS, ϕ – TWTL formula in concatenation form, $\boldsymbol{\tau}$ – Best temporal relaxation found so far, $\mathcal{A}_\infty$ – The DFA associated with ϕ

Output: A state in $\mathcal{T}$ to extend from, and a random state to extend to

if $\text{Unif}([0,1]) < p_{bias}$ **then**
  $k \leftarrow \text{argmax}_{i \in \mathfrak{I}_{\mathcal{A}_\infty}(\phi(\boldsymbol{\tau}))}\{\tau_i\}$
  $s_{ran} \leftarrow \text{Unif}(States_{\mathcal{A}_\infty}(\phi_k))$
else
  $s_{ran} \leftarrow \text{Unif}(States_{\mathcal{A}_\infty}(\phi))$
$x_{ran} \leftarrow \text{Unif}(W_{free})$
$x_{exp} \leftarrow Nearest(\{x \in V : \delta^*_{\mathcal{A}_\infty}(\vec{o}) = s_{ran}\}, x_{ran})$
return (x_{exp}, x_{ran})

Intuitively, sampling the DFA state has the effect of choosing which layer, as shown in Fig. 1, we are exploring next. This observation allows us to develop heuristics that bias the sampling towards states that need to be better explored. In particular, we propose a sampling method in which the states associated with the subformula ϕ_i with largest τ_i are sampled with greater probability.

Our sampling procedure is presented in Alg. 2. The biased sampling described above is performed over a uniform sampling with probability p_{bias}. The $States_{\mathcal{A}_\infty}$ function returns the states in the DFA associated with a subformula; the Unif function chooses uniformly from a given set; and the $Nearest$ function returns the nearest point from a set to another point. The input parameter $\boldsymbol{\tau}$ represents the best temporal relaxation (LRTR) for a path satisfying $\phi(\boldsymbol{\tau})$ so far, and it is important to keep it stored and updated throughout the execution of the algorithm in order to avoid unnecessary computation.

4.4 Completeness and optimality

We assume for our analysis an additional constraint for Prob. 1: the set U is discrete. Furthermore, the steering function is modified such that $Steer(x, y)$ returns the result of applying a random control input from U for one timestep from x. We start with an observation about the rewiring process:

Theorem 1. *Consider the graph $G = (V_G, E_G)$ obtained by running Alg. 1 for some number of iterations with line 12 deleted, as well as the tree $T = (V_T, E_T)$ obtained by running the unmodified algorithm. Then, $V_G = V_T$ and $\forall v \in V_G, Cost_{\mathcal{A}_\infty}(\vec{v}^G) \geq Cost_{\mathcal{A}_\infty}(\vec{v}^T)$, where $\vec{v}^T$ is the path from x_0 to v in T and $\vec{v}^G$ is any path from x_0 to v in G.*

Proof. Since the rewiring process only modifies the set of edges, the set of nodes remains the same. For the second property, note that line 11 ensures that the edges resulting in paths with a worse cost are removed.

Theorem 2 (Probabilistic Completeness). *If Prob. 1 with the discrete control input assumption has a solution, then Alg. 1 finds it with probability 1 as the number of iterations go to infinity.*

Proof. We sketch a proof closely following that in [22]. First, consider the algorithm without line 12. Suppose a solution path is $\mathbf{x} = \{x_k\}_{k=0}^{L}$. By induction on k, assume the graph contains a path until x_k for some k. Since the states of $\mathcal{A}_\infty$ are finite, with non-zero probability the state associated with the path from x_0 to x_k, $\delta^*_{\mathcal{A}_\infty}(\vec{o}_k)$, will be selected for extension. Consider now the Voronoi diagram associated with the nodes in the selected state. With non-zero probability, a sample in the Voronoi cell associated with x_k will be obtained. Finally, with non-zero probability the correct control input that steers the system towards x_{k+1} will be selected. Therefore, the next point in the solution path will be constructed with probability tending to one. This process continues until the solution path is constructed in G. Since the path has cost 0, the path to its last node in the tree resulting from enabling rewiring has cost less than or equal to 0 by Theorem 1. Therefore, it is a solution path.

We now turn our attention to the behavior of the algorithm when faced with a scenario in which only a relaxation of the specification can be satisfied:

Theorem 3. *Suppose Prob. 1 has a solution for a τ-relaxation of the specification ϕ, $\phi(\boldsymbol{\tau})$, with $|\phi(\boldsymbol{\tau})|_{LRTR} = c$, and no τ-relaxation $\phi(\boldsymbol{\tau'})$, with $|\phi(\boldsymbol{\tau'})|_{LRTR} = c' < c$, yields a satisfiable specification. Then, if Alg. 1 is run for the original specification ϕ, it will find a path satisfying $\phi(\boldsymbol{\tau''})$, with $|\phi(\boldsymbol{\tau''})|_{LRTR} = c$, for some τ-relaxation $\phi(\boldsymbol{\tau''})$, with probability 1.*

Proof. Suppose the algorithm is run for the specification $\psi = \phi(\boldsymbol{\tau})$ and the rewiring process is disabled. The resulting graph has the same nodes as it would have if run for the specification ϕ. Moreover, a path with cost 0 with respect to ψ, or, equivalently, c with respect to ϕ, exists in the graph with probability 1, by Theorem 2. When rewiring is applied, the path to the final node will have less or equal cost by Theorem 1. However, by hypothesis, no path satisfying a relaxed formula exists with cost less than c. Therefore, a path with cost c is found with probability 1.

4.5 Complexity

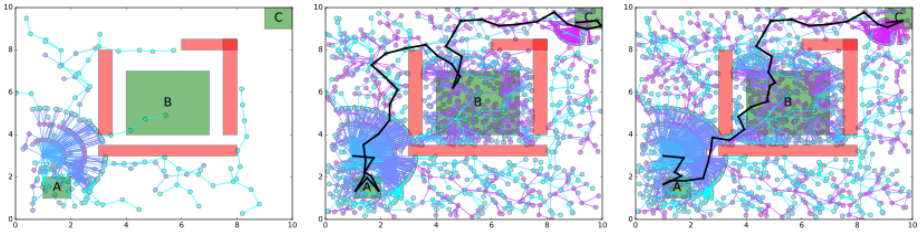
Given the similarity of Alg. 1 with RRT*, it is not surprising that the computational complexity is similar. We analyze the number of calls to $ColFree$ per iteration as well as the cost of some of the operations described in Sec. 4.1.

Consider an arbitrary ϕ, which gets translated into a DFA with set of states $\{s_i\}_{i=1}^{S}$. Note that $ColFree$ is restricted to nodes in particular DFA states, given by the construction of the V_{near} and V_{next} sets, which is related to the sampled DFA state, s_{ran}. The number of considered states depends on certain features of the formula (such as the number of disjunction operands) that are not necessarily related to S, so we assume it fixed. Let p_i be the probability of a node being in one of those DFA states when s_i is sampled (which in general decreases when S increases, although its distribution depends on the structure of the formula and the workspace) and let P_S be its expected value. Then, the number of calls to $ColFree$ in iteration n is $\mathcal{O}(P_S n)$.

We can proceed in a similar way in order to analyze the primitive procedures. Regarding the $ColFree$ function itself, it can be executed in $\mathcal{O}(\log^d m)$, where d is the dimension of the space and m is the number of obstacles . The sets V_{near} and V_{next} computed in line 6 and line 9 respectively are also referred to as *range search problems* and can be approximately solved in $\mathcal{O}(\log P_S n + (1/\epsilon)^{d-1})$, where ϵ is a parameter controlling the precision. Finally, the optimization problem solved in line 7 to find x_{min} is an instance of the nearest neighbor search problem, which can be approximately solved in $\mathcal{O}(c_{d,\epsilon} \log P_S n)$, where $c_{d,\epsilon} \leq d\lceil 1+6d/\epsilon \rceil^d$. See [16] for a discussion on these results.

5 Case Studies

We implemented the algorithm in Python2.7 and we ran all examples on an Intel(R) Core(TM) i5-4690K CPU @ 3.50GHz with 8GB RAM. We compared



(a) TS after a few iterations (b) Path with $\boldsymbol{\tau}^* = (0, 3, 0)$ (c) Satisfying path

Fig. 2. Evolution of Alg. 1 for case study 1. The regions of interest are colored in green and obstacles in red. The color of each node in the TS represents its associated DFA state. The current best path satisfying the relaxed formula is highlighted in bold black.

execution times with [35] using a grid with 20 divisions per dimension for the state space partition and skipping the computation of transitions between cells.

5.1 Case Study 1: Example revisited

We consider the following more complicated specification for the example: "Perform tasks at A, B, and C, in this order, of duration 2, 3, and 2 time units, within time intervals [3, 10], [0, 15], and [0, 15] from the end of the previous task, respectively." The corresponding TWTL formula is $\phi = [\mathcal{H}^2 A]^{[3,10]} \cdot [\mathcal{H}^3 B]^{[0,15]} \cdot [\mathcal{H}^2 C]^{[0,15]}$. The parameters of the algorithm are $d_{steer} = 0.75$ and $p_{bias} = 0.5$.

We show in Fig. 2 several snapshots of the state of the algorithm after some iterations. Note that the layers associated with each state in the DFA shown schematically in Fig. 1 can be identified in the figure by the different colors in nodes and edges. In Fig. 2a, we show the state of the algorithm when it has yet to find a path that satisfies any τ-relaxation of ϕ. At this point, the algorithm is exploring the state space in search of a candidate path. The cluster of nodes near x_0 is due to the sampling of states associated with delaying until the lower bound of the first time interval. The next snapshot, Fig. 2b, highlights a candidate path. After this iteration, the exploitation phase starts and the algorithm biases the sampling towards the subpath with worst deadline deviation (in this case, the subpath from A to B). If the algorithm is stopped at this iteration, it would return the highlighted path, which is a partial solution that violates the specification. We quantify the violation with the temporal relaxation. In Fig. 2c the candidate path was refined enough to finally satisfy ϕ. The two predominant colors in the figure, cyan and light magenta, correspond to the initial states of the second and third subformula respectively. The maximum, minimum and average times it took to solve the problem over 20 executions were 250, 7 and 61 seconds, respectively. We repeated the simulation with $p_{bias} = 0$ obtaining maximum, minimum and average times of 1766, 17 and 289 seconds respectively, which shows a performance decrease when language-guided sampling is

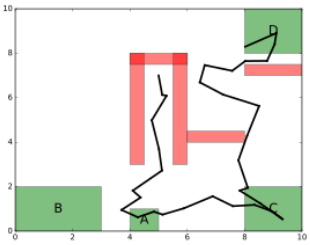


Fig. 3. Final path returned by the algorithm for the second case study, as seen when projected onto its first two components.

disabled. In comparison, the algorithm proposed in [35] is able to obtain a path in an average of 1 second.

5.2 Case Study 2

We consider a workspace in $\mathbb{R}^{10}$ with five obstacles and three regions of interest, A, B and C. The system dynamics are the same as in the previous example, with $||u_k|| \leq 2$ and initial point $x_0 = (5, 7, 3, \ldots, 3)$. The specification in this case is $\phi = [\mathcal{H}^1 A]^{[0,25]} \cdot \left([\mathcal{H}^1 B]^{[0,15]} \vee [\mathcal{H}^1 C]^{[0,15]}\right) \cdot [\mathcal{H}^1 D]^{[0,35]}$, and we set $d_{steer} = 2$ and $p_{bias} = 0.5$. We show in Fig. 3 the satisfying path found by the algorithm. The execution time needed to solve an instance of the problem was 3834 seconds. In this case, the algorithm in [35] required too much memory to run.

6 Conclusion

In this paper, we introduced a sampling-based algorithm for solving motion planning problems under temporal logic specifications given as TWTL formulae. The algorithm initially finds a path that satisfies a temporally relaxed version of the specification. Then, sampling is biased towards the subpath that needs more improvement in order to satisfy the time bounds of the specification.

Our algorithm relies on the translation of TWTL formulae to annotated Deterministic Finite State Automata, a process recently developed in [35]. The design of the algorithm is inspired from RRT*, but differs in two major aspects. First, we incrementally construct the product between a Transition System, with states in the workspace, and the DFA. This allows us to not only grow a random tree in a similar way as RRT*, but to also keep track of our progress towards satisfying the specification. Second, we make use of a cost function related to the satisfaction of the formula that deviates from the usual metrics used by RRT*, like path length. We showed that for this cost function, not only our algorithm is probabilistically complete, but it can also obtain a "minimally violating" path in those cases were only a temporally relaxed version of the specification can be satisfied, again in a probabilistically complete way.

We implemented the algorithm in Python and we tested it for high dimensional problems. We obtained correct results at a moderately high computational cost, partially due to our naive implementation lacking the best known algorithms for solving nearest neighbors and range search problems.

As future work, we plan to extend the fragment of TWTL that we accept in order to allow specifications with "subtasks", which could be solved by recursive calls to the proposed algorithm. We also want to assess the performance when better algorithms for primitive operations are used.

Acknowledgments. This work was partially supported by the NSF under grant NRI-1426907 at Boston University.

References

[1] Aksaray, D., Vasile, C.I., Belta, C.: Dynamic Routing of Energy-Aware Vehicles with Temporal Logic Constraints. In: IEEE International Conference on Robotics and Automation. pp. 3141–3146. Stockholm, Sweden (May 2016)
[2] Aydin Gol, E., Lazar, M., Belta, C.: Language-Guided Controller Synthesis for Linear Systems. IEEE Trans. on Automatic Control 59(5), 1163–1176 (2014)
[3] Bacchus, F., Kabanza, F.: Planning for temporally extended goals. Annals of Mathematics and Artificial Intelligence 22(1-2), 5–27
[4] Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press (2008)
[5] Belta, C., Isler, V., Pappas, G.J.: Discrete abstractions for robot planning and control in polygonal environments. IEEE Trans. on Robotics 21(5), 864–874 (2005)
[6] Bhatia, A., Kavraki, L., Vardi, M.: Sampling-based motion planning with temporal goals. In: IEEE International Conference on Robotics and Automation (2010)
[7] Canny, J.F.: The Complexity of Robot Motion Planning. MIT Press, USA (1988)
[8] Chen, Y., Tumova, J., Belta, C.: LTL Robot Motion Control based on Automata Learning of Environmental Dynamics. In: IEEE International Conference on Robotics and Automation. Saint Paul, MN, USA (2012)
[9] Choset, H., Lynch, K., et al.: Principles of Robot Motion: Theory, Algorithms, and Implementations. MIT Press, Boston, MA (2005)
[10] Ding, X.C., Kloetzer, M., et al.: Formal Methods for Automatic Deployment of Robotic Teams. IEEE Robotics and Automation Magazine 18, 75–86 (2011)
[11] Doherty, P., Kvarnström, J., Heintz, F.: A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. Autonomous Agents and Multi-Agent Systems 19(3), 332–377 (Feb 2009)
[12] Gol, E.A., Belta, C.: Time-Constrained Temporal Logic Control of Multi-Affine Systems. Nonlinear Analysis: Hybrid Systems 10, 21–23 (2013)
[13] Guo, M., Dimarogonas, D.V.: Multi-agent plan reconfiguration under local LTL specifications. International Journal of Robotics Research 34(2), 218–235 (2015)
[14] Jha, S.K., Clarke, E.M., et al.: A bayesian approach to model checking biological systems. In: Computational Methods in Systems Biology. Springer-Verlag (2009)
[15] Karaman, S., Frazzoli, E.: Sampling-based Motion Planning with Deterministic μ-Calculus Specifications. In: IEEE Conference on Decision and Control (2009)
[16] Karaman, S., Frazzoli, E.: Sampling-based Algorithms for Optimal Motion Planning. International Journal of Robotics Research 30(7), 846–894 (June 2011)
[17] Karaman, S., Frazzoli, E.: Sampling-based Optimal Motion Planning with Deterministic μ-Calculus Specifications. In: American Control Conference (2012)
[18] Kavraki, L., Svestka, P., et al.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation 12(4), 566–580 (1996)
[19] Koymans, R.: Specifying real-time properties with metric temporal logic. Real-time systems 2(4), 255–299 (1990)

[20] Kress-Gazit, H., Fainekos, G.E., Pappas, G.J.: Where's Waldo? Sensor-based temporal logic motion planning. In: IEEE International Conference on Robotics and Automation. pp. 3116–3121 (2007)
[21] Lahijanian, M., Almagor, S., et al.: This Time the Robot Settles for a Cost: A Quantitative Approach to Temporal Logic Planning with Partial Satisfaction. In: AAAI Conference on Artificial Intelligence. pp. 3664–3671. Austin, Texas (2015)
[22] LaValle, S.M., Kuffner, J.J.: Randomized Kinodynamic Planning. International Journal of Robotics Research 20(5), 378–400
[23] Luo, R., Valenzano, R.A., et al.: Using Metric Temporal Logic to Specify Scheduling Problems. In: Principles of Knowledge Representation and Reasoning (2016)
[24] Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, pp. 152–166. Springer (2004)
[25] Maly, M., Lahijanian, M., et al.: Iterative Temporal Motion Planning for Hybrid Systems in Partially Unknown Environments. In: Int. Conference on Hybrid Systems: Computation and Control (2013)
[26] Nakagawa, S., Hasuo, I.: Near-Optimal Scheduling for LTL with Future Discounting. In: Ganty, P., Loreti, M. (eds.) Trustworthy Global Computing, pp. 112–130. No. 9533 in Lecture Notes in Computer Science, Springer (2015)
[27] Pavone, M., Bisnik, N., et al.: A stochastic and dynamic vehicle routing problem with time windows and customer impatience. Mobile Networks and Applications 14(3), 350–364 (2009)
[28] Plaku, E., Kavraki, L.E., Vardi, M.Y.: Motion Planning with Dynamics by a Synergistic Combination of Layers of Planning. IEEE Trans. on Robotics 26(3), 469–482 (2010)
[29] Reyes Castro, L., Chaudhari, P., et al.: Incremental sampling-based algorithm for minimum-violation motion planning. In: IEEE Conference on Decision and Control. pp. 3217–3224 (2013)
[30] Solomon, M.M.: Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations research 35(2), 254–265 (1987)
[31] Tkachev, I., Abate, A.: Formula-free Finite Abstractions for Linear Temporal Verification of Stochastic Hybrid Systems. In: Int. Conference on Hybrid Systems: Computation and Control. Philadelphia, PA (2013)
[32] Tumova, J., Marzinotto, A., et al.: Maximally satisfying LTL action planning. In: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems. pp. 1503–1510 (2014)
[33] Tumova, J., Hall, G.C., et al.: Least-violating Control Strategy Synthesis with Safety Rules. In: Hybrid Systems: Computation and Control. pp. 1–10 (2013)
[34] Vasile, C., Belta, C.: Sampling-Based Temporal Logic Path Planning. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (2013)
[35] Vasile, C.I., Aksaray, D., Belta, C.: Time Window Temporal Logic. Theoretical Computer Science p. (submitted), http://arxiv.org/abs/1602.04294
[36] Vasile, C.I., Belta, C.: An Automata-Theoretic Approach to the Vehicle Routing Problem. In: Robotics: Science and Systems Conference. USA (2014)
[37] Wolff, E.M., Topcu, U., Murray, R.M.: Automaton-guided controller synthesis for nonlinear systems with temporal logic. In: IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 4332–4339 (2013)
[38] Wongpiromsarn, T., Topcu, U., Murray, R.M.: Receding Horizon Temporal Logic Planning for Dynamical Systems. In: Conference on Decision and Control (2009)
[39] Yoo, C., Fitch, R., Sukkarieh, S.: Probabilistic temporal logic for motion planning with resource threshold constraints (2012)