Real-Time Heuristic Search with LTLf Goals

Jaime Middleton[1, 4], **Rodrigo Toro Icarte**[1,2], **Jorge A. Baier**[1,2,3]
[1]Department of Computer Science, Pontificia Universidad Católica de Chile, Santiago, Chile
[2]Centro Nacional de Inteligencia Artificial CENIA, Santiago, Chile
[3]Instituto Milenio Fundamentos de los Datos, Santiago, Chile
[4]Tucar SpA, Santiago, Chile
jamiddleton@uc.cl, rodrigo.toro@ing.puc.cl, jabaier@ing.puc.cl

Abstract

In Real-Time Heuristic Search (RTHS) we are given a search graph G, a heuristic, and the objective is to find a path from a given start node to a given goal node in G. As such, one does not impose any trajectory constraints on the path, besides reaching the goal. In this paper we consider a version of RTHS in which temporally extended goals can be defined on the form of the path. Such goals are specified in Linear Temporal Logic over Finite Traces (LTL_f), an expressive language that has been considered in many other frameworks, such as Automated Planning, Synthesis, and Reinforcement Learning, but has not yet been studied in the context of RTHS. We propose a general automata-theoretic approach for RTHS, whereby LTL_f goals are supported as the result of searching over the cross product of the search graph and the automaton for the LTL_f goal; specifically, we describe LTL-LRTA*, a version of LSS-LRTA*. Second, we propose an approach to produce heuristics for LTL_f goals, based on existing goal-dependent heuristics. Finally, we propose a greedy strategy for RTHS with LTL_f goals, which focuses search to make progress over the structure of the automaton; this yields LTL-LRTA*$_A$. In our experimental evaluation over standard benchmarks we show LTL-LRTA*$_A$ may outperform LTL-LRTA* substantially for a variety of LTL_f goals.

1 Introduction

Real-time heuristic search [Korf, 1990] (RHTS) is an approach to solving search problems by interleaving search and execution. It is important for applications in which there is little time to search before an action has to be executed. Some applications are videogames and highly dynamic robotics.

As originally defined, RTHS consists of reaching a goal state on a given search graph. However, many real-time applications may require agents to satisfy temporally extended goals (i.e., *eventually fetch the key and then reach the door*; *eventually board the spaceship while avoiding locations with mud*). Such kinds of goals, which impose constraints over the trajectory of states traversed by the agent, are typically represented in linear temporal logic (LTL) in areas such as automated planning [Cresswell and Coddington, 2004; Baier and McIlraith, 2006; Kabanza and Thiébaux, 2005; Gerevini *et al.*, 2009; Simon and Röger, 2015], synthesis [De Giacomo and Vardi, 2015; Bonet *et al.*, 2020], and reinforcement learning [Toro Icarte *et al.*, 2018; Camacho *et al.*, 2019; Vaezipoor *et al.*, 2021].

In this paper [1] we consider adding goals expressed in linear temporal logic over finite traces (LTL_f) [De Giacomo and Vardi, 2013] to RTHS. Thus, we assume we are given a search graph G, a vertex of G where the agent is initially at, and an LTL_f formula φ specifying legal trajectories for the agent. The problem consists of moving the agent through one of such trajectories.

We take a standard automata-theoretic approach in which we use the fact that an LTL_f formula φ has a corresponding finite-state automaton A_φ which accepts the traces defined by φ. We consider the cross product between the search space and the automata to propose LTL-LRTA*, a version of LSS-LRTA* [Koenig and Sun, 2009] that searches over the cross-product between the search graph and A_φ.

But simply considering search over the cross-product representation does not address the important problem of guiding real-time search. To that end, we present two orthogonal contributions aimed at guiding search for LTL_f goals. First, we present a method to construct a heuristic function for any LTL_f goal assuming we have a *goal-independent* heuristic $\hat{h}$ in hand, which is such that $\hat{h}(s, g)$ estimates the cost of a path from a state s to a given goal state g. Second, we propose *automata subgoaling*, a novel approach to carry out search within the RTHS algorithm that prioritizes "making progress" in the automaton for the LTL_f formula. This is accomplished by ordering the search frontier considering the distance $\Delta(q)$ between the current automaton state q to an accepting state of A_φ, and using A*'s priority function $f = g+h$ as a tie breaker rather than as the main guiding function. When applying this principle to LTL-LRTA* we obtain LTL-LRTA*$_A$. We prove LTL-LRTA*$_A$ is complete—under standard assumptions—when the goal is such that its automaton's graph structure does not have loops, excluding self-loops.

Since no algorithms for RTHS with LTL_f goals exist, the

[1] Our source code and appendix are publicly available at https://github.com/Jamidd/RTHS-with-LTLf-Goals

objective of our experimental evaluation was to find the strengths and weaknesses of the algorithms we propose. For our experiments we use standard grids, Starcraft maps and mazes, using a number of LTL_f goals. We show that LTL-LRTA*$_A$ may outperform LTL-LRTA* by a significant margin. The relative performance of both algorithms depends on the quality of the heuristic being used. We conclude that using the information captured by the automaton for the goal, either by exploiting the automaton to build a heuristic or by using automata subgoaling, is important.

Although we are the first to propose to use the structure of the goal's automaton to guide search in RTHS, this idea, as a general concept, is not new. Indeed, it has been considered before in planning with LTL goals [Kabanza and Thiébaux, 2005], but in a way that is not compatible with RTHS. Furthermore, the idea of using subgoaling has also been considered before in regular RTHS [Bulitko and Björnsson, 2009; Hernández and Baier, 2011]. The way in which we incorporate subgoaling, by changing the priority of $Open$, is, however, fundamentally different from previous work.

While in this paper we incorporate our techniques in LSS-LRTA*, a generalization of LRTA* [Korf, 1990], our principles are general and can be applied to a number of other RTHS algorithms.

2 Real-Time Heuristic Search

Real-time heuristic search (RTHS) is an approach to solving search problems [Korf, 1990]. A key characteristic of the approach is that the amount of computation for decision making is bounded by a constant B, after which one or more actions may be performed. If the problem has not been solved yet, B more units of computation are allowed for a new decision-making episode. The loop repeats until the problem is solved.

A *final-state search problem* $P = (S, E, c, s_{start}, G)$ is a tuple, where (S, E, c) is a *search graph*, S is a set of *states*, $E \subseteq S \times S$ is a finite set of *directed edges*, $c : S \times S \mapsto (0, \infty]$ is a *cost function*, $s_{start} \in S$ is the *initial state*, $G \subseteq S$ is a set of *goal states*.

We denote the set of neighbors of state s as $N(s)$, formally defined as $N(s) = \{t \mid (s, t) \in E\}$. A *path* π from s_1 to s_n is a sequence of states $(s_1, s_2, \ldots, s_n)$ such that $(s_i, s_{i+1}) \in E$ for all $i \in \{1, \cdots, n-1\}$. Any path from s_{start} to a goal state in G is a *solution* to the search problem $P = (S, E, c, s_{start}, G)$. The *cost* of path π, denoted by $c(\pi)$, is the cumulative cost of all the edges traversed in π; that is, $c(\pi) = \sum_{i=1}^{n-1} c(s_i, s_{i+1})$. State s is a *dead end* if there is no path from s to a state in G.

To help the agent solve a search problem, RTHS algorithms use a *heuristic function* $h : S \to [0, \infty]$, such that $h(s)$ estimates the cost of a path from s to a goal state in G. A heuristic function h is *consistent* iff $h(s) = 0$ for every state $s \in G$ and $h(s) \leq c(s, t) + h(t)$ for every $(s, t) \in E$. The *perfect heuristic* h^* is a heuristic such that $h^*(s)$ denotes the cost of a minimum-cost path between s and a state in G. Heuristic h is *admissible* iff $h(s) \leq h^*(s)$ for every $s \in S$. Consistency implies admissibility.

A typical RTHS algorithm runs a main loop in which search is carried out in the vicinity of the current state. Such a search may not exceed a given computation bound, usually given in terms of the maximum number of states which can be expanded during search. The information gathered during search is used to (1) decide which state the agent will move to, and (2) to update the heuristic function. This process repeats until finding a goal state.

3 Search Problems with LTL_f Goals

In this section, we formally define search problems with LTL_f goals. To do so, we first describe the semantics of LTL_f [Bienvenu *et al.*, 2006; De Giacomo and Vardi, 2013]. We then show how to transform LTL_f formulas into goals and heuristics for a search problem. `All the theorems from this section are proven in Appendix A.`

3.1 LTL_f Goals and Automata

LTL_f extends propositional logic with temporal operators such as next($\bigcirc$), weak-next($\bullet$), and until(U). As in propositional logic, LTL_f formulas are defined over a set of propositional symbols $\mathcal{P}$. However, unlike propositional logic, LTL_f formulas are satisfied with respect to *traces*, which are sequences of truth value assignments to the propositions in $\mathcal{P}$.

To use LTL_f in real-time search, we define the semantics of LTL_f with respect to paths on a search graph. To that end, we assume the agent is capable of sensing the truth value of propositions on a given state. We formalize this notion using a labelling function $\mathcal{L} : S \mapsto 2^{\mathcal{P}}$, which maps each state $s \in S$ to a set of propositions in $\mathcal{P}$, which are true in s.

Given a path $\pi = (s_0, s_1, \ldots, s_n)$ in a search graph, we say that π *satisfies* an LTL_f formula φ with respect to $\mathcal{L}$, denoted $\pi \models \varphi$, iff $\pi[0] \models \varphi$, where, for every $i \in \{0, \ldots, n\}$:

1. $\pi[i] \models p$ iff $p \in \mathcal{L}(s_i)$, where $p \in \mathcal{P}$.
2. $\pi[i] \models \neg\varphi$ if $\pi[i] \not\models \varphi$.
3. $\pi[i] \models (\varphi \wedge \psi)$ if $\pi[i] \models \varphi$ and $\pi[i] \models \psi$.
4. $\pi[i] \models \bigcirc\varphi$ if $i < n$ and $\pi[i+1] \models \varphi$.
5. $\pi[i] \models \bullet\varphi$ if $i < n$ implies that $\pi[i+1] \models \varphi$.
6. $\pi[i] \models \varphi \,\mathsf{U}\, \psi$ if $\pi[i] \models \psi$ for some $k \geq i$ and $\pi[j] \models \varphi$ for every $j \in \{i, \ldots, k-1\}$.

The connectives $\vee$ (or), $\Box$ (always), and $\Diamond$ (eventually) can be defined in terms of the above, as $(\varphi \vee \psi) \stackrel{\text{def}}{=} \neg(\neg\varphi \wedge \neg\psi)$, $\Diamond\varphi \stackrel{\text{def}}{=} \mathsf{true}\,\mathsf{U}\,\varphi$, and $\Box\varphi \stackrel{\text{def}}{=} \neg\Diamond\neg\varphi$.

A useful feature of LTL_f is that any formula can be translated into an equivalent *deterministic finite-state automaton (DFA)* using standard libraries, such as Spot [Duret-Lutz *et al.*, 2016] or LTL_f2DFA [Fuggitti, 2019]. In the context of real-time search, translating an LTL_f formula φ into a DFA results in a DFA $A_\varphi = (Q, 2^{\mathcal{P}}, \delta, q_{start}, F)$, where Q is a finite set of automaton states, $2^{\mathcal{P}}$ is the power set of the propositions in $\mathcal{P}$ (i.e., the range of the labelling function), $\delta : Q \times 2^{\mathcal{P}} \to Q$ is the transition function, $q_{start} \in Q$ is the initial state, and $F \subseteq Q$ is the set of accepting states. Then, a path $\pi = (s_1, \ldots, s_n)$ in the search graph is accepted by A_φ iff $q_n \in F$, where $q_{i+1} = \delta(q_i, \mathcal{L}(s_i))$ for all $i \in \{1, \ldots, n-1\}$ and $q_1 = q_{start}$. In other words, A_φ accepts π iff, starting from q_{start} and updating the DFA state according to δ and π, the DFA ends in an accepting state. Finally, we note that $\pi \models \varphi$ iff A_φ accepts π by construction. We could say that the $i\text{-}th$ automata state, represents the $i\text{-}th$ sub-problem to be solved in order to complete the goal. Below we use notation $\Delta(q)$ to represent the minimum number of transitions required to reach an accepting state in A_φ from q. As such, $\Delta(q) = 0$ if $q \in F$, and $\Delta(q) = \infty$ if there is no path in A_φ from q to an accepting state.

3.2 Running Example

As a simple example, consider the grid world shown in Figure 1a, which includes an agent, walls, and four marked locations: a, b, c, and d. We might consider that each marked location represents a subtask that the agent can solve by reaching that location. In this domain, we can define the set of propositional symbols $\mathcal{P} = \{a, b, c, d\}$ and its corresponding labelling function $\mathcal{L}$ such that $p \in \mathcal{L}(s)$ iff the proposition p is true in state s (i.e., the agent is at location p). Now we can consider different LTL$_f$ goals for the agent in this domain. For instance, $\Diamond a$ (i.e., *eventually* a) requires that the agent reaches a cell marked with a. The previous task might be considered a standard goal in RTHS. However, LTL$_f$ goals allow us to define more complex, temporally extended goals. Some examples include to solve task a and then task b (i.e., $\Diamond(a \wedge \bigcirc \Diamond b)$), to avoid states in which certain property d holds (i.e., $\Box \neg d$), or combinations of LTL$_f$ goals (e.g., $\Diamond(a \wedge \bigcirc \Diamond b) \wedge \Box \neg d$).

Figures 1b and 1c show examples of two more complex LTL$_f$ goals with their corresponding DFAs. The first LTL$_f$ goal asks the agent to solve all the subtasks in alphabetical order. The second LTL$_f$ goal is similar to the first LTL$_f$ goal but it also forces the agent to solve the subtasks in strict order. That means that, if the agent solves task b before solving task a, then the agent fails at solving overall task.

3.3 LTL$_f$ Search Problems (LSP)

Now we formally define a search problem with an LTL$_f$ goal. Intuitively, a search problem with an LTL$_f$ goal is like any other search problem but where the objective for the agent is to traverse a sequence of states that satisfies an LTL$_f$ formula instead of reaching a particular goal state.

Definition 1 (LTL$_f$ Search Problem (LSP)).
An LTL$_f$ search problem $P = (S, E, c, s_{start}, \mathcal{P}, \varphi, \mathcal{L})$ is a tuple, where (S, E, c) is a search graph, $s_{start} \in S$ is the start state, φ is an LTL$_f$ formula over a set of propositions $\mathcal{P}$, and $\mathcal{L} : S \mapsto 2^{\mathcal{P}}$ is a labelling function.

Now we formally define what is a solution to an LSP.

Definition 2 (Solution to an LSP).
Let P be an LSP. A path $\pi = (s_1, \ldots, s_n)$ is a solution *to P iff π is a path over the search graph (S, E, c) from the initial state (i.e., $s_1 = s_{start}$) and $\pi \models \varphi$ with respect to $\mathcal{L}$. Moreover, the* solution cost *of π is $c(\pi) = \sum_{i=1}^{n-1} c(s_i, s_{i+1})$.*

A standard technique to solve search problems with temporally extended goals is to exploit the cross-product between the automaton and the search graph [Bacchus *et al.*, 1997; Baier and McIlraith, 2006]. By doing so we can *compile away* the temporal goal, transforming the LSP into a regular search problem. An important advantage of this transformation is that it allows solving an LSP with any off-the-shelf RTHS algorithm. The formal definition of a cross-product LSP and the equivalence between solving the LSP and the corresponding cLSP follows.

Theorem 1 (Cross-product LSP (cLSP)). *Given an LSP $P = (S, E, c, s_{start}, \mathcal{P}, \varphi, \mathcal{L})$, let the DFA for φ be $A_\varphi = (Q, 2^{\mathcal{P}}, \delta, q_{start}, F)$. Finally, let P_φ be the cross-product LSP (cLSP) defined as $(S_\varphi, E_\varphi, c_\varphi, s_{start\varphi}, G_\varphi)$ such that:*

1. *$S_\varphi = S \times Q$,*
2. *$((s, q), (t, r)) \in E_\varphi$ iff $(s, t) \in E$ and $r = \delta(q, \mathcal{L}(t))$,*
3. *$c_\varphi((s, q), (t, r)) = c(s, t)$,*
4. *$s_{start\,\varphi} = (s_{start}, q_{start})$, and*
5. *$(s, q) \in G_\varphi$ iff $q \in F$.*

Then path $\pi = (s_1, \ldots, s_n)$ is a solution to P iff $\pi_\varphi = ((s_1, q_1), \ldots, (s_n, q_n))$ is a solution to P_φ, where $q_1 = q_{start}$, $s_1 = s_{start}$, and $q_{i+1} = \delta(q_i, \mathcal{L}(s_{i+1}))$, for every $i \in \{1, \ldots, n-1\}$. In addition, $c(\pi) = c_\varphi(\pi_\varphi)$.

3.4 Heuristics for LSPs

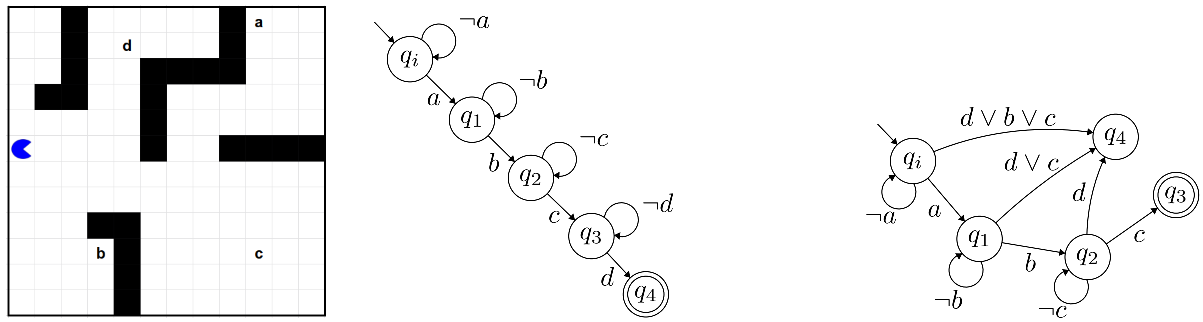
The cross-product transformation allows us to solve any LSP with an off-the-shelf RTHS algorithm. However, heuristics are key for the performance of any heuristic search algorithm, including RTHS algorithms. While users of heuristic search algorithms are aware of methods to construct heuristics for a given standard search problem P with final-state goals, their techniques may not be readily applicable for building heuristics for LSPs. In this section we present a simple approach to generate a heuristic for any given LTL$_f$ formula from an existing final-state heuristic for the same underlying problem.

More specifically, assume that given a search graph (S, E) and a cost function c, we have a heuristic $\hat{h}(s, t)$ that estimates the cost of a path from s to t. Henceforth we call these heuristics goal-independent, since they receive the goal as a parameter. For many search problems with fixed goal states (i.e., 15-puzzle, Pancacke problem) goal-dependent heuristics can be easily generated after a simple state transformation. These state transformations may be even applicable to Pattern-database heuristics, which are goal-dependent. In addition, domain-independent heuristics typically used in planning [Bonet and Geffner, 2000; Hoffmann and Nebel, 2001; Helmert, 2006] are goal-independent.

Now we show how given a heuristic $\hat{h}(s, t)$ for search graph (S, E, c) and an LTL$_f$ formula φ we can construct a heuristic h_φ.

Definition 3 (The cross-product heuristic).
Given an LSP $P = (S, E, c, s_{start}, \mathcal{P}, \varphi, \mathcal{L})$ and a goal-independent heuristic $\hat{h} : S \times S \mapsto [0, \infty)$ for $\mathcal{G} = (S, E, c)$, we define the cross-product heuristic $h_\varphi : S \times Q \mapsto [0, \infty)$ as follows. Let $A_\varphi = (Q, 2^{\mathcal{P}}, \delta, q_0, F)$ be the DFA representation of the LTL$_f$ formula φ. And let $S_q \subseteq S$ be the set of all states that cause a transition from $q \in Q$ to $q' \in Q$ such that $q \neq q'$ while an accepting state can be reached from q' in A_φ. That is, $S_q = \{s \in S \mid q' = \delta(q, \mathcal{L}(s)), q \neq q', \Delta(q') < \infty\}$. Then,

- *$h_\varphi(s, q) = 0$ if $q \in F$.*
- *$h_\varphi(s, q) = \infty$ if $\Delta(q) = \infty$.*



(a) A grid world in which propositions a, b, c, d are true in certain cells.

(b) DFA for $\Diamond(a \wedge \bigcirc\Diamond(b \wedge \bigcirc\Diamond(c \wedge \bigcirc\Diamond d)))$, which requires the agent to solve all subtasks in alphabetical order.

(c) DFA for $\Box\neg d \wedge (\neg(b \vee c) \,\mathsf{U}\, a) \wedge (\neg c \,\mathsf{U}\, b) \wedge \Diamond(a \wedge \bigcirc\Diamond(b \wedge \bigcirc\Diamond c))$, which is similar to the task of (b) but forces the agent to solve the subtasks in strict order.

Figure 1: Running example with two conceivable LTL_f goals. Transitions in the DFAs have been simplified using the fact that there is no state in which two propositions are true. In addition, states containing no outgoing transitions are assumed to have a self-loop with 'true'.

- $h_\varphi(s,q) = \min_{t \in S_q} \tilde{c}(s,t) + h_\varphi(t, \delta(q, \mathcal{L}(t)))$, *otherwise.*

where $\tilde{c}(s,t) = \max(\hat{h}(s,t), \epsilon)$ *and* ϵ *is the minimum cost of any transitions in* $\mathcal{G}$. *That is,* $\epsilon = \min\{c(s,t) \mid (s,t) \in E\}$.

Intuitively, $h_\varphi(s,q)$ estimates the cost to go from (s,q) to a goal state $(s^*, q^*) \in G_\varphi$ by solving a simplification of the LSP P. This simplification consists of only considering the subset of states $S_q \subseteq S$ that can change the current DFA state q. Then, the cost to go from $s \in S$ to $t \in S_q$, which might not be connected in the original LSP, is given by the heuristic $\hat{h}(s,t)$. The system of equations from Definition 3 simply states the standard constraints that define a solution to the shortest path problem that results from solving the simplified version of P [Wolsey, 1998]. Note that we introduced ϵ to ensure that all the cost are greater than zero in the simplified problem. This ensures that any solution to the system of equations will indeed reach an accepting state (if that state is reachable) in the simplified problem.

As an example, let us consider two possible cross-product heuristics for our running example (Figure 1a). In this domain, the cost of performing any action is equal to 1. Therefore, a simple heuristic for this problem is $\hat{h}^1(s,t) = 1$ if $s \neq t$ and zero otherwise. Let s be the current state of the agent in Figure 1a and φ be the LTL_f formula from Figure 1b. Here, the cross-product heuristic will simply sum $\hat{h}^1(s, s_a)$, $\hat{h}^1(s_a, s_b)$, $\hat{h}^1(s_b, s_c)$, and $\hat{h}^1(s_c, s_d)$. Then, $h^1_\varphi(s, q_{start}) = 4$ because, to reach an accepting state in the simplified problem, the agent has to hit locations `a`, `b`, `c`, and `d`.

We note that $\hat{h}^1$ is a particularly naive heuristic for this problem. A better heuristic would be to use the Manhattan distance $\hat{h}^M$ to construct the cross-product heuristic h^M_φ. The Manhattan distance $\hat{h}^M(s,t)$ estimates the cost to go from s to t by counting the number of moves needed to go from s to t if we ignore all obstacles. In the running example, $h^M_\varphi(s, q_{start}) = 48$ because that is the sum of the Manhattan distances to go through the states s, s_a, s_b, s_c, and finally s_d.

On the theoretical side, we now show that the cross-product heuristic $h_\varphi(s_\varphi)$ inherits properties from its base heuristic $\hat{h}(s,t)$. In particular, if $\hat{h}$ is admissible (resp. consistent) for $\mathcal{G}$, then h_φ is admissible (resp. consistent) for P_φ. As a result, the two previously discussed heuristics, namely h^1_φ and h^M_φ, are consistent for the running example.

Theorem 2 (Properties of the cross-product heuristic)**.** *Let* $P = (S, E, c, s_{start}, \mathcal{P}, \varphi, \mathcal{L})$ *be an LSP,* P_φ *be the cLSP for* P*, and* $\hat{h} : S \times S \mapsto [0, \infty)$ *be a heuristic for* $\mathcal{G} = (S, E, c)$. *Let* h_φ *be the cross-product heuristic constructed from* P *and* $\hat{h}$. *Then, the following two properties hold. First, if* $\hat{h}$ *is admissible for* $\mathcal{G}$ *then* h_φ *is admissible for* P_φ. *And second, if* $\hat{h}$ *is consistent for* $\mathcal{G}$ *then* h_φ *is consistent for* P_φ.

On the practical side, we note that the effectiveness of computing the cross-product heuristic $h_\varphi(s,q)$ depends on the cardinality of S_q. In particular, the complexity of computing $h_\varphi(s,q)$ by solving the resulting shortest path-problem using Dijkstra is $O(v^2)$, where $v = 1 + \sum_{q \in Q} |S_q|$. In some problems, such as in grid worlds, $|S_q|$ is much smaller than $|S|$ and, thus, h_φ can be precomputed for all cross-product states $(s,q) \in S \times Q$ in just a few seconds. However, computing h_φ might be a challenge in highly combinatorial problems, such as the 15-puzzle. For those cases, we devise two possibilities. One option is to compute h_φ using methods other than Dijkstra's. For instance, h^1_φ can be computed by only looking at the DFA, meaning that the complexity of computing h^1_φ can go down to $O(|Q|^2)$ regardless of the size of the S_q sets.

Another option is to approximate h_φ by removing the recursion from Definition 3. That is, to define $\tilde{h}_\varphi(s,q)$ exactly as $h_\varphi(s,q)$, but when $q \notin F$ and $\Delta(q) < \infty$, then $\tilde{h}_\varphi(s,q) = \min\{\tilde{c}(s,t) \mid t \in S_q\}$. We will refer to $\tilde{h}_\varphi$ as a *myopic heuristic*. $\tilde{h}_\varphi$ has two nice properties. First, Theorem 2 also applies to $\tilde{h}_\varphi$. Second, the complexity of computing $\tilde{h}_\varphi(s,q)$ is just $O(|S_q|)$. However, $\tilde{h}_\varphi$ is weaker than h_φ since $\tilde{h}_\varphi(s,q) \leq h_\varphi(s,q)$ for all $(s,q) \in S \times Q$. We formally discuss all these properties in Appendix A.3.

Algorithm 1: LTL-LRTA*, a simple variant of LSS-LRTA* that solves an LSP by carrying out search over the cross-product representation.

Input : An LSP $P = (S, E, c, s_{start}, \mathcal{P}, \varphi, \mathcal{L})$, the DFA for φ defined as $(Q, 2^{\mathcal{P}}, \delta, q_{start}, F)$, a heuristic $h : S \times Q \mapsto [0, \infty)$, and a positive integer k

Effect: The agent is moved through a path from s_{start} to a goal state in G if a path exists

1 $(s_{now}, q_{now}) \leftarrow (s_{start}, q_{start})$
2 **while** $q_{now} \notin F$ **do**
3     $Open, Closed \leftarrow$ `Bounded-A*` (s_{now}, q_{now})
4   **if** *Open is empty* **then**
5   **print** "no solution"
6   **abort execution**
7     $\pi \leftarrow$ path from (s_{now}, q_{now}) to state at the top of $Open$
8   `Dijkstra-Update`$(Open, Closed)$
9   **for each** (s, q) *in* π
10         $(s_{now}, q_{now}) \leftarrow (s, q)$
11   Move agent to s_{now}

4 Real-Time Search for LSPs

We have shown that an LSP can be solved by solving its equivalent cLSP problem which does not have a temporally extended goal. Furthermore, we described an approach to compute heuristics to guide search for the cLSP. In this section we present two RTHS approaches for solving cLSPs. The first one, our baseline approach, is a straightforward modification of LSS-LRTA*. The second one is a greedy algorithm that we call *automata subgoaling*, which, as we show later, may lead to significantly improved performance when solving cLSPs. `We note that all the theoretical results from this section are proven in Appendix B.`

4.1 RTHS over the Cross-Product State Space

The first technique we propose is straightforward from Theorem 1, which establishes the equivalence between searching over an LSP or its cross product with the automaton for the $\mathrm{LTL_f}$ formula. Algorithm 1 describes LTL-LRTA*, a simple variant of LSS-LRTA* [Koenig and Sun, 2009] which receives an LSP P and searches over the corresponding cLSP.

LTL-LRTA* takes as input an LSP P, the DFA for the $\mathrm{LTL_f}$ goal φ, a heuristic function $h : S \times Q \mapsto [0, \infty)$ over the cross-product states, and a positive integer k. Even though we assume the agent moves over graph (S, E), in its main loop, LTL-LRTA* maintains variables s_{now} and q_{now} which are such that (s_{now}, q_{now}) is the state of the corresponding cLSP where the agent is at.

The only algorithmic difference between LTL-LRTA* and LSS-LRTA* is that search is carried out over the cross-product representation. Specifically for the search part of the algorithm, LTL-LRTA* invokes `Bounded-A*` in Line 3. `Bounded-A*` takes (s_{new}, q_{new}) as the root of the search, and expands nodes using the transition function for automaton A_φ; as such, to compute the neighbors of (s, q) it simply iterates over each neighbor t of s, generating a state of the form $(t, \delta(q, \mathcal{L}(t)))$. As any standard A* implementation, the priority used in the $Open$ priority queue is given by $f = g+h$, where $g(s, q)$ is the cost of the best path found so far to state (s, q), and h is the heuristic function. `Bounded-A*` stops after k expansions have been made or when the goal is at the top of $Open$. Finally, for decision-making, just like LSS-LRTA* does, LTL-LRTA* traverses path π from the current state to the state at the top of $Open$, that is, the one with the lowest f-value (Line 7; Algorithm 1). For the heuristic update (Line 8), the algorithm uses a version of Dijkstra's algorithm which receives the $Open$ and $Closed$ datastructures previously returned by `Bounded-A*`, just like LSS-LRTA* would do. After the execution of Line 8, for every element (s, q) in $Closed$ it holds that $h(s, q) = \min_{(t,r) \in N(s,q)} c(s, t) + h(t, r)$.

LTL-LRTA* was designed to be equivalent to LSS-LRTA* when run over the cLSP for P, P_φ. It therefore inherits the following property of LSS-LRTA*.

Theorem 3. *Let P be an LSP, and let $P_\varphi = (S_\varphi, E_\varphi, c_\varphi, s_{start\,\varphi}, G_\varphi)$ be its corresponding cLSP. Moreover, let h_φ be a consistent heuristic for P_φ. Then LTL-LRTA*, run over P and h_φ is guaranteed to find a solution to P if a solution exists and no dead end in (S_φ, E_φ) is reachable from $s_{start\varphi}$.*

4.2 Automata Subgoaling

The efficiency of LTL-LRTA* depends mainly on the quality of the given heuristic h_φ. Now we present a technique which exploits the structure of the automaton A_φ as an additional source of guidance. The main intuition is that we can understand automaton states as 'subgoals' and focus the search on states in which progress is made on the automaton structure.

We implement this in `Bounded-A*`, by changing the priority of $Open$ to be computed as $(\Delta(q), f(s, q))$ for a state (s, q), and assuming a lexicographic ordering. As such, as soon the search finds a state (s, q) such that q is 'closer' to an accepting state than any other state in $Open$, search focuses on such states, since such states will be pushed to the top of $Open$. Since order is lexicographic, the f function still guides search within states that are equally closer to an accepting state. The pseudo-code is in Appendix C.

If the DFA has no cycles (although it might have self-loops), then LTL-LRTA*$_A$ is guaranteed to solve the LSP. This property is stated in the following theorem.

Theorem 4. *Let P be an LSP, and P_φ be its corresponding cLSP. Assume P_φ contains no dead-end states reachable from $s_{start\,\varphi}$ and that any path $\pi_\varphi = ((s_1, q_1), \ldots, (s_n, q_n))$ from $s_{start\,\varphi}$ is such that $\Delta(q_i) < \Delta(q_j)$ for every i, j such that $1 \leq i < j \leq n$ when $q_i \neq q_j$. Then LTL-LRTA*$_A$, run over P using a consistent h_φ finds a solution to P if a solution exists.*

5 Empirical Evaluation

Since no approaches to RTHS with $\mathrm{LTL_f}$ goals exist, the main objective of our empirical evaluation was to understand what was the effectiveness of the various techniques we propose. Therefore, we compare LTL-LRTA* and LTL-LRTA*$_A$ using different heuristic functions over three well-known benchmarks and five $\mathrm{LTL_f}$ goals.

Configuration		No. best solution			Sol. gap
Heuristic	Domain	LTL	Ties	LTL_A	LTL / LTL_A
h^1_φ	Rooms	1020	0	1980	1.295
	StarCraft	1213	0	1787	1.158
	Maze	1272	0	1728	1.137
h^M_φ	Rooms	841	635	1524	1.020
	StarCraft	461	2008	531	1.002
	Maze	588	1716	696	1.005
$\tilde{h}^M_\varphi$	Rooms	20	0	2980	31.912
	StarCraft	27	0	2973	11.564
	Maze	169	1	2830	3.106
	Total	5611	4360	**17029**	**2.328**

Table 1: Results for different Maps-Heuristic combinations

5.1 Experimental Setup

We tested LTL-LRTA* and LTL-LRTA*$_A$ with various problems, heuristics, and goals. We ran experiments on three families of grid problems proposed by Sturtevant [2012]: *Rooms*, *Starcraft*, and *Maze*.[2] In these domains, the agent can only move to empty cells, and each cell can be either empty or blocked. To build a search graph from the grid we connect each cell to its 4 immediate cardinal neighbors and the cost for every graph edge is 1. In addition, we randomly marked vertices of the graph with letters from the alphabet in $\{\mathtt{a}, \mathtt{b}, \ldots, \mathtt{i}\}$.

We evaluated the performance of LTL-LRTA* and LTL-LRTA*$_A$ using two cross-product heuristics: h^1_φ and h^M_φ, and one myopic heuristic: $\tilde{h}^M_\varphi$. Intuitively, $h^1_\varphi(s,q)$ is equal to the minimum distance between q and an accepting state in the DFA, where the cost of transitioning between DFA states is always 1. h^M_φ works similarly to $h^1_\varphi(s,q)$, but the cost to move between DFA states is given by the Manhattan distance between the environment states that the agent must reach to cause that transition. Finally, the myopic heuristic $\tilde{h}^M_\varphi$ is simply the Manhattan distance between the current state and the closest state that would cause a transition in the DFA. More details can be found in Section 3.4.

We evaluated the algorithms using five different LTL_f goals. These goals exploit a wide range of the features that LTL_f provides. As such, some of these goals include completing a sequences of tasks (e.g., the agent has to solve a set of tasks in order), partial order tasks (e.g., some tasks must be solved before other tasks), disjunctive tasks (e.g., the agent can either do task 1 or task 2), and safety constraints (e.g., the agent must ensure that a condition holds as it solves the main task). These five goals are formally described in Appendix D.

5.2 Comparison of RTHS Methods for LSPs

A summary of our results is shown in Table 1. In total, we ran $27,000$ experiments. These considered 5 LTL_f goals, 3 domains (i.e., rooms, starcraft, and maze), 10 maps per domain, 5 problem instances per map (where each instance is a different placement of the letters in the map), 6 lookahead values (with $k \in \{32, 64, 128, 256, 512, 1024\}$), and 3 heuristics ($h^1_\varphi$, h^M_φ, and $\tilde{h}^M_\varphi$.). When placing letters in the maps, we either placed three letters of each type (i.e., three $\mathtt{a}$'s, three $\mathtt{b}$'s, etc) or twenty-five.

Table 1 compares the performance of LTL-LRTA* and LTL-LRTA*$_A$ using two main metrics. The first metric is the *number of best solution*, which is the number of times that an algorithm found a better than the other. In the table, column *LTL* refers to the number of problems where LTL-LRTA* found a better solution than LTL-LRTA*$_A$ and column *LTL*$_A$ refers to the number of problems where LTL-LRTA*$_A$ found a better solution than LTL-LRTA*. We also included the number of ties. As the table shows, LTL-LRTA*$_A$ tends to find better solutions than LTL-LRTA*$_A$. Indeed, LTL-LRTA*$_A$ found the best solution in $17,029$ problems out of $27,000$ (and tied in $4,360$). The table also shows that the advantage of LTL-LRTA*$_A$ over LTL-LRTA* is problem dependent. For instance, the performance of both methods is quite similar in StarCraft when using h^M_φ, whereas LTL-LRTA*$_A$ solves almost every problem faster than LTL-LRTA* in Rooms when using $\tilde{h}^M_\varphi$.

The second performance metric is the *solution gap*. This metric compares the quality of the solutions found by each method. Specifically, we computed the solution gap by dividing the cost of the solution found by LTL-LRTA* by the cost of the solution found by LTL-LRTA*$_A$, and then reported the *geometric mean* of these ratios across all the experiments. As Table 1 shows, LTL-LRTA*$_A$ found solutions that were over 2.3 times better than the solutions found by LTL-LRTA* on average. However, we only see large gaps for the case of the myopic heuristic $\tilde{h}^M_\varphi$. When using the cross-product heuristic h^1_φ, LTL-LRTA*$_A$ finds solutions that are around 1.2 times better. And when using h^M_φ, LTL-LRTA*$_A$ is only marginally better than LTL-LRTA*. We believe that part of the reason why LTL-LRTA*$_A$ performs similar to LTL-LRTA* in this latter case is that h^M_φ is a well-informed (although expensive) heuristic. In particular, h^M_φ already propagates back the information that moving to DFA states that are closer to an accepting state decreases the heuristic value. As such, ordering $Open$ by Δ does not make a large difference in the algorithm's performance.

We performed a per-lookahead analysis, customary in the RTHS literature. We observed that when a significant difference in solution cost exists between LTL-LRTA* and LTL-LRTA*$_A$ such differences are similar across different lookahead values. In addition, we observed no relevant differences between goal types. A detailed analysis is included in the Appendix, Tables 5-14.

Finally, we note that LTL-LRTA* and LTL-LRTA*$_A$ have identical computational complexity given the same heuristic. For that reason, the fact that LTL-LRTA*$_A$ tends to find better solutions than LTL-LRTA* also implies that LTL-LRTA*$_A$ finds solutions faster than LTL-LRTA*.

[2]In Rooms we used all the maps with 8×8 rooms. In Maze we used the maps with hallways of width 32. In Starcraft we used the maps: BlastFurnance, CatwalkAlley, Crossroads, Enigma, Infermo, FloodedPlains, SpaceAtoll, Octopus, ValleyofRe, WheelofWar.

Configuration		Time gap w.r.t. h^1_φ		
# items	Domain	h^M_φ	$h^M_\varphi + A$	$\tilde{h}^M_\varphi + A$
3 items	Rooms	**9.542**	9.256	7.963
	StarCraft	**3.161**	3.084	2.922
	Maze	**1.016**	0.994	1.003
25 items	Rooms	3.395	**3.420**	3.327
	StarCraft	2.449	2.381	**2.490**
	Maze	1.532	1.518	**1.631**
100 items	Rooms	2.233	2.379	**2.444**
	StarCraft	2.359	2.303	**2.428**
	Maze	1.936	1.992	**2.340**
	Total	2.512	2.501	**2.538**

Table 2: Runtime results using different heuristics. # items corresponds to the number of instances of the same letter that appear in the map. Time gap is defined as the ratio between the runtime obtained by LTL-LRTA*$_A$ using h^1_φ and the corresponding algorithm.

Configuration		Solution gap w.r.t. h^1_φ		
# items	Domain	h^M_φ	$h^M_\varphi + A$	$\tilde{h}^M_\varphi + A$
3 items	Rooms	30.012	**30.256**	21.056
	StarCraft	4.11	**4.112**	3.587
	Maze	**1.008**	1.006	0.980
25 items	Rooms	11.162	**11.530**	9.526
	StarCraft	5.381	**5.402**	5.155
	Maze	1.847	**1.867**	1.862
100 items	Rooms	4.791	**5.037**	4.535
	StarCraft	4.94	**4.959**	4.709
	Maze	2.602	2.675	**3.063**
	Total	4.558	**4.628**	4.240

Table 3: Solution cost analysis using different heuristics. # items corresponds to the number of instances of the same letter that appear in the map. Solution gap is defined as the ratio between the solution cost obtained by LTL-LRTA*$_A$ using h^1_φ and the corresponding algorithm.

5.3 Comparison of Heuristics for LSPs

Now we compare the performance of different heuristics for solving LSPs. In Section 3.4, we proposed two ways to compute heuristics for LSPs: the cross-product heuristics and the myopic heuristics. The myopic heuristics are weaker than the cross-product heuristics, but they are faster to compute. In particular, the complexity of computing the myopic heuristic is $O(|S_q|)$ whereas the complexity of computing the cross-product heuristic is $O(v^2)$, where $v = 1 + \sum_{q \in Q} |S_q|$. Recall that $S_q \subseteq S$ is the subset of states that change the current DFA state $q \in Q$ to some $q' \in Q$, such that $\Delta(q') < \infty$. Thus, we would expect that a cross-product heuristic will usually find better solutions than a myopic heuristic but, depending on the size of S_q, a myopic heuristic might find solutions faster than a cross-product heuristic. To verify this behavior empirically, we ran experiments using different numbers of duplicated letters on each map. As the number of letters increases, the size of S_q also increases since there are more locations that the agent could reach in order to change the current DFA state.

Table 2 shows a runtime comparison between the cross-product heuristic h^M_φ and the myopic heuristic $\tilde{h}^M_\varphi$. That is, it reports how fast each method is able to find a solution for the LSP, without carrying about the quality of such a solution. The table normalizes the performance of each method with respect to the performance of LTL-LRTA*$_A$ using h^1_φ. Thus, a performance of 9.5 means that the method solved the LSP 9.5 times faster than LTL-LRTA*$_A$ using h^1_φ. Specifically, the table reports the performance of three methods:

- h^M_φ refers to LTL-LRTA* using h^M_φ,
- $h^M_\varphi + A$ refers to LTL-LRTA*$_A$ using h^M_φ, and
- $\tilde{h}^M_\varphi + A$ refers to LTL-LRTA*$_A$ using $\tilde{h}^M_\varphi$.

The results in Table 2 show the following. When we only have three instances of each letter (i.e., $|S_q|$ is relatively small), the cross-product heuristic h^M_φ dominates. However, as we increase the number of instances of each letter, the myopic heuristic $\tilde{h}^M_\varphi$ tends to solve LSPs faster (on average) than the cross-product heuristic h^M_φ. In addition, solutions found by the cross-product heuristic are usually better than the solutions found by the myopic heuristic.

Table 3 reports the solution gap between each method with respect to LTL-LRTA*$_A$ using h^1_φ, for the same algorithm configurations as Table 2. It shows that the cross-product heuristic h^M_φ tends to find better solutions than myopic heuristics, regardless of the number of instances of each letter. This behavior is expected since the cross-product heuristic is stronger than the myopic heuristic.

Therefore, the decision of using a cross-product heuristic or a myopic heuristic is application- (and problem-) dependent. If the goal is to find the best possible solution, it is better to use a cross-product heuristic. However, if the goal is to get any solution as fast as possible and the S_q sets are large, then it might be better to use a myopic heuristic.

6 Conclusion

In this paper, we studied how to incorporate temporally extended goals into RTHS. We proposed to encode such goals using LTL$_\mathrm{f}$ and formally defined search problem with LTL$_\mathrm{f}$ goals, which we called LSPs. We exploited the relationship between LTL$_\mathrm{f}$ and DFAs to construct a cross-product version of an LSP (cLSP). A cLSP can be solved by any off-the-shelf RTHS method. We studied different ways in which the structure of the DFA can be used to improve the performance of RTHS methods when solving a cLSP. Specifically, we (i) showed how to use the DFA to compute heuristics and (ii) proposed LTL-LRTA*$_A$, an algorithm that guides the search exploiting the DFA's structure. We studied the theoretical properties of our heuristics and algorithm. Our empirical results showed the benefits of exploiting the DFA structure for LSP solving.

Acknowledgments

We gratefully acknowledge funding from the National Center for Artificial Intelligence CENIA FB210017, Basal ANID.

References

[Bacchus *et al.*, 1997] Fahiem Bacchus, Craig Boutilier, and Adam J. Grove. Structured solution methods for non-markovian decision processes. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI)*, pages 112–117, 1997.

[Baier and McIlraith, 2006] Jorge A. Baier and Sheila A. McIlraith. Planning with temporally extended goals using heuristic search. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 342–345, 2006.

[Bienvenu *et al.*, 2006] Meghyn Bienvenu, Christian Fritz, and Sheila A. McIlraith. Planning with qualitative temporal preferences. In *Proceedings of the 10th International Conference on Knowledge Representation and Reasoning (KR)*, pages 134–144, June 2006.

[Bonet and Geffner, 2000] Blai Bonet and Hector Geffner. Planning with incomplete information as heuristic search in belief space. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Systems (AIPS)*, pages 52–61, 2000.

[Bonet *et al.*, 2020] Blai Bonet, Giuseppe De Giacomo, Hector Geffner, Fabio Patrizi, and Sasha Rubin. High-level programming via generalized planning and LTL synthesis. In Diego Calvanese, Esra Erdem, and Michael Thielscher, editors, *Proceedings of the 17th International Conference on Knowledge Representation and Reasoning (KR)*, pages 152–161, 2020.

[Bulitko and Björnsson, 2009] Vadim Bulitko and Yngvi Björnsson. knn lrta*: Simple subgoaling for real-time search. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 4, 2009.

[Camacho *et al.*, 2019] Alberto Camacho, Rodrigo Toro Icarte, Toryn Q Klassen, Richard Anthony Valenzano, and Sheila A McIlraith. LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, volume 19, pages 6065–6073, 2019.

[Cresswell and Coddington, 2004] Stephen Cresswell and Alexandra M. Coddington. Compilation of LTL goal formulas into PDDL. In Ramon López de Mántaras and Lorenza Saitta, editors, *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)*, pages 985–986, Valencia, Spain, August 2004. IOS Press.

[De Giacomo and Vardi, 2013] Giuseppe De Giacomo and Moshe Y Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

[De Giacomo and Vardi, 2015] Giuseppe De Giacomo and Moshe Y. Vardi. Synthesis for LTL and LDL on finite traces. In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1558–1564. AAAI Press, 2015.

[Duret-Lutz *et al.*, 2016] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0—A Framework for LTL and ω-Automata Manipulation. In *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, pages 122–129. Springer, 2016.

[Fuggitti, 2019] Francesco Fuggitti. Ltlf2dfa, March 2019.

[Gerevini *et al.*, 2009] Alfonso Gerevini, Patrik Haslum, Derek Long, Alessandro Saetti, and Yannis Dimopoulos. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, 173(5-6):619–668, 2009.

[Helmert, 2006] Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[Hernández and Baier, 2011] Carlos Hernández and Jorge A. Baier. Fast subgoaling for pathfinding via real-time search. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS)*, Freiburg, Germany, June 2011.

[Hoffmann and Nebel, 2001] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[Kabanza and Thiébaux, 2005] Froduald Kabanza and Sylvie Thiébaux. Search control in planning for temporally extended goals. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 130–139, 2005.

[Koenig and Sun, 2009] Sven Koenig and Xiaoxun Sun. Comparing real-time and incremental heuristic search for real-time situated agents. *Autonomous Agents and Multi-Agent Systems*, 18(3):313–341, 2009.

[Korf, 1990] Richard E. Korf. Real-time heuristic search. *Artificial Intelligence*, 42(2):189–211, 1990.

[Simon and Röger, 2015] Salomé Simon and Gabriele Röger. Finding and exploiting LTL trajectory constraints in heuristic search. In Levi Lelis and Roni Stern, editors, *Proceedings of the 8th Symposium on Combinatorial Search (SoCS)*, pages 113–121. AAAI Press, 2015.

[Sturtevant, 2012] Nathan Sturtevant. Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games*, 4(2):144 – 148, 2012.

[Toro Icarte *et al.*, 2018] Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. Teaching multiple tasks to an RL agent using LTL. In *Proceedings of the 17th International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS)*, pages 452–461, 2018.

[Vaezipoor *et al.*, 2021] Pashootan Vaezipoor, Andrew C Li, Rodrigo Toro Icarte, and Sheila A Mcilraith. Ltl2action: Generalizing ltl instructions for multi-task rl. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, pages 10497–10508, 2021.

[Wolsey, 1998] Laurence A Wolsey. *Integer programming*. John Wiley & Sons, 1998.