Real-Time Heuristic Search with LTLf Goals

Jaime Middleton\textsuperscript{1, 4}, Rodrigo Toro Icarte\textsuperscript{1, 2}, Jorge A. Baier\textsuperscript{1, 2, 3}

\textsuperscript{1}Department of Computer Science, Pontificia Universidad Católica de Chile, Santiago, Chile
\textsuperscript{2}Centro Nacional de Inteligencia Artificial CENIA, Santiago, Chile
\textsuperscript{3}Instituto Milenio Fundamentos de los Datos, Santiago, Chile
\textsuperscript{4}Tucar SpA, Santiago, Chile

jamidd@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 (LTLf), 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 LTLf goals are supported as the result of searching over the cross product of the search graph and the automaton for the LTLf goal; specifically, we describe LTL-LRTA*, a version of LSS-LRTA*. Second, we propose an approach to produce heuristics for LTLf goals, based on existing goal-dependent heuristics. Finally, we propose a greedy strategy for RTHS with LTLf 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 LTLf 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 \textit{et al}., 2009; Simon and Röger, 2015], synthesis [De Giacomo and Vardi, 2015; Bonet \textit{et al}., 2020], and reinforcement learning [Toro Icarte \textit{et al}., 2018; Camacho \textit{et al}., 2019; Vaezipoor \textit{et al}., 2021].

In this paper \textsuperscript{1} we consider adding goals expressed in linear temporal logic over finite traces (LTLf) [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 LTLf formula $\varphi$ 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 LTLf formula $\varphi$ has a corresponding finite-state automaton $A_\varphi$ which accepts the traces defined by $\varphi$. 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 $A_\varphi$.

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 LTLf goals. First, we present a method to construct a heuristic function for any LTLf goal assuming we have a goal-independent heuristic $h$ in hand, which is such that $h(s, g)$ 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 LTLf 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 $A_\varphi$, and using $A_\varphi$’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.

\textsuperscript{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*$_f$ 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 subgoal, 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 Thiebaux, 2005], but in a way that is not compatible with RTHS. Furthermore, the idea of using subgoal has also been considered before in regular RTHS [Bulitko and Björnsson, 2009; Hernández and Baier, 2011]. The way in which we incorporate subgoal, 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 $P = (S, E, c, s\text{start}, G)$ is a tuple, where $(S, E, c)$ is a search graph, $S$ is a set of states, $E \subseteq S \times S$ is a set of directed edges, $c : S \times S \rightarrow [0, \infty]$ is a cost function, $s\text{start} \in S$ 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 $\pi$ 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 $i \in \{1, \ldots, n-1\}$. Any path from $s\text{start}$ to a goal state $G$ in $S$ is a solution to the search problem $P = (S, E, c, s\text{start}, G)$. The cost of path $\pi$, denoted by $c(\pi)$, is the cumulative cost of all the edges traversed in $\pi$; 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 $h : S \rightarrow [0, \infty]$, such that $h(s)$ estimates the cost of a path from $s$ to a goal state in $G$. A heuristic function $h$ is consistent if $h(s) = 0$ 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 if $h(s) \leq h^*(s)$ 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) 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 $P$. However, unlike propositional logic, LTL$_f$ formulas are satisfied with respect to $traces$, which are sequences of truth value assignments to the propositions in $P$.

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 $L : S \rightarrow 2^P$, which maps each state $s \in S$ to a set of propositions in $P$, which are true in $s$.

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

1. $\pi[i] \models p$ iff $p \in L(s_i)$, where $p \in P$.
2. $\pi[i] \models \neg \varphi$ if $\pi[i] \not\models \varphi$.
3. $\pi[i] \models (\varphi \land \psi)$ if $\pi[i] \models \varphi$ and $\pi[i] \models \psi$.
4. $\pi[i] \models \Diamond \varphi$ if $i < n$ and $\pi[i + 1] \models \varphi$.
5. $\pi[i] \models \Box \varphi$ if $i < n$ implies that $\pi[i + 1] \models \varphi$.
6. $\pi[i] \models \varphi U \psi$ 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 $\lor$ (or), $\square$ (always), and $\Diamond$ (eventually) can be defined in terms of the above, as $(\varphi \lor \psi) \equiv \neg(\neg\varphi \land \neg\psi)$, $\varphi \land \psi \equiv \neg(\neg\varphi \lor \neg\psi)$, $\varphi \land \psi \equiv \neg(\neg\varphi \lor \neg\psi)$, $\Diamond \varphi \equiv \true \lor \varphi$, and $\Box \varphi \equiv \neg \Diamond \neg \varphi$.

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 LTL2DFA [Fuggitti, 2019]. In the context of real-time search, translating an LTL$_f$ formula $\varphi$ into a DFA results in a DFA $A_\varphi = (Q, 2^P, \delta, q_{\text{start}}, F)$, where $Q$ is a finite set of automaton states, $2^P$ is the power set of the propositions in $P$ (i.e., the range of the labelling function), $\delta : Q \times 2^P \rightarrow Q$ is the transition function, $q_{\text{start}} \in Q$ 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 $A_\varphi$ iff $q_n \in F$, where $q_{i+1} = \delta(q_i, L(s_i))$ for all $i \in \{1, \ldots, n-1\}$ and $q_1 = q_{\text{start}}$. In other words, $A_\varphi$ accepts $\pi$ iff, starting from $q_{\text{start}}$ and updating the DFA state
An LTL tuple, where instead of reaching a particular goal state.

Definition 2
Moreover, the P iff and φ.

Intuitively, a search problem with an LTL f. 3.3 LTL a cell marked with a standard goal in RTHS. However, LTL a goal asks the agent to solve all the subtasks in alphabetical order. The second LTL goal is like any search problem but where the objective for the agent is some exam-

Now we formally define what is a solution to an LSP. 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, sstart, P, φ, L) , let the DFA for φ be Aφ = (Q, 2P, δ, qstart, F). Finally, let Pφ be the cross-product LSP (cLSP) defined as (Sφ, Eφ, cφ, sstartφ, Gφ) such that:
1. Sφ = S × Q.
2. ((s, q), (t, r)) ∈ Eφ iff (s, t) ∈ E and r = δ(q, L(t)),
3. cφ((s, q), (t, r)) = c(s, t),
4. sstartφ = (sstart, qstart), and
5. (s, q) ∈ Gφ iff q ∈ F.

Then path π = (s1, . . . , sn) is a solution to P iff πφ = ((s1, q1), . . . , (sn, qn)) is a solution to Pφ, where q1 = qstart, s1 = sstart, and qi+1 = δ(qi, L(s+1)), for every i ∈ {1, . . . , n − 1}. In addition, c(π) = cφ(πφ).

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 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 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, Pancake problem) goal-dependent heuris-

tics 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 ad-


Now we show how given a heuristic h(s, t) for search graph (S, E, c) and an LTL formula φ we can construct a heuristic hφ.

Definition 3 (The cross-product heuristic).
Given an LSP P = (S, E, c, sstart, P, φ, L) and a goal-independent heuristic h : S × S → [0, ∞) for G = (S, E, c), we define the cross-product heuristic hφ : S × Q → [0, ∞) as follows. Let Aφ = (Q, 2P, δ, q0, F) be the DFA representation of the LTL formula φ. And let Sφ ⊆ S be the set of all states that cause a transition from q ∈ Q to q′ ∈ Q such that q ̸= q′ while an accepting state can be reached from q′ in Aφ. That is, Sφ = {s ∈ S | q′ = δ(q, L(s)), q ̸= q′, ∆(q′) < ∞}. Then,

• hφ(s, q) = 0 if q ∈ F.
• hφ(s, q) = ∞ if ∆(q) = ∞.
### Properties of the cross-product heuristic

Let \( P = (S, E, c, s_{\text{start}}, \varphi, L) \) be an LSP, \( P_{\varphi} \) be the cLSP for \( P \), and \( \hat{h} : S \times S \rightarrow [0, \infty) \) be a heuristic for \( G = (S, E, c) \). Let \( h_{\varphi} \) be the cross-product heuristic constructed from \( P \) and \( \hat{h} \). Then, the following two properties hold.

- \( h_{\varphi} \) is admissible for \( G \) if and only if \( \hat{h} \) is admissible for \( P_{\varphi} \).

- \( h_{\varphi} \) is consistent for \( G \) if and only if \( \hat{h} \) is consistent for \( P_{\varphi} \).

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’s algorithm 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, \( h_{\varphi} \) can be precomputed for all cross-product states \( (s, q) \) in \( S \times Q \) in just a few seconds. However, computing \( h_{\varphi} \) 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 \( h_{\varphi} \) using methods other than Dijkstra’s. For instance, \( h_{\varphi}^{1} \) can be computed by only looking at the DFA, meaning that the complexity of computing \( h_{\varphi}^{1} \) can go down to \( O(|Q|^2) \) regardless of the size of the \( S_{q} \) sets.

Another option is to approximate \( h_{\varphi} \) by removing the recursion from Definition 3. That is, to define \( \hat{h}_{\varphi}(s, q) \) exactly as \( h_{\varphi}(s, q) \), but when \( q \not\in P \) and \( \Delta(q) < \infty \), then \( \hat{h}_{\varphi}(s, q) = \min\{c(s, t) \mid t \in S_{q}\} \). We will refer to \( \hat{h}_{\varphi} \) as a myopic heuristic. \( \hat{h}_{\varphi} \) has two nice properties. First, Theorem 2 also applies to \( \hat{h}_{\varphi} \). Second, the complexity of computing \( \hat{h}_{\varphi}(s, q) \) is just \( O(|S_{q}|) \). However, \( \hat{h}_{\varphi} \) is weaker than \( h_{\varphi} \), since \( \hat{h}_{\varphi}(s, q) \leq h_{\varphi}(s, q) \) 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 \( P = (S, E, c, s_{start}, P, \varphi, L) \), the DFA for \( \varphi \) defined as \((Q, 2^P, \delta, q_{start}, F)\), a heuristic \( h : S \times Q \rightarrow [0, \infty) \), 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. \( \text{if Open is empty then}
4. \quad \text{print "no solution"}
5. \quad \text{abort execution}
6. \( \pi \leftarrow \text{path from } (s_{now}, q_{now}) \text{ to state at the top of } \text{Open}\)
7. \( \text{Dijkstra-Update (Open, Closed)}\)
8. for each \((s, q) \in \pi\)
9. \( (s_{now}, q_{now}) \leftarrow (s, q)\)
10. 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 LTL* 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 LTL* goal \( \varphi \), a heuristic function \( h : S \times Q \rightarrow [0, \infty) \) 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 \( A_\varphi \); as such, to compute the neighbors of \((s, q)\) it simply iterates over each neighbor \( t \) of \( s \), generating a state of the form \((t, \delta(q, c(t)))\). 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 \( \pi \) 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 \((s, q) \in \text{Closed}\) it holds that \( h(s, q) = \min_{(t, r) \in \text{N}(s, q)} c(s, t) + h(t, r) \).

LTL-LRTA* was designed to be equivalent to LSS-LRTA* when run over the cLSP for \( P, P_\varphi \). 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 \( h_\varphi \) be a consistent heuristic for \( P_\varphi \). Then LTL-LRTA*, run over \( P \) and \( h_\varphi \) is guaranteed to find a solution to \( P \) if a solution exists and no dead end in \((S_\varphi, E_\varphi)\) is reachable from \( s_{start_\varphi} \).

4.2 Automata Subgoaling

The efficiency of LTL-LRTA* depends mainly on the quality of the given heuristic \( h_\varphi \). Now we present a technique which exploits the structure of the automaton \( A_\varphi \) 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 \( P_\varphi \) be its corresponding cLSP. Assume \( P_\varphi \) 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 \( h_\varphi \) finds a solution to \( P \) if a solution exists.

5 Empirical Evaluation

Since no approaches to RTHS with LTL\(_\varphi\) 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 LTL\(_\varphi\) goals.
A summary of our results is shown in Table 1. In total, we present 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 ($h_1^M$, $h_1^M$, and $h_1^M$). When placing letters in the maps, we either placed three letters of each type (i.e., three $a$’s, three $b$’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 solution 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 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*. 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 $h_1^M$, whereas LTL-LRTA*_A solves almost every problem faster than LTL-LRTA* in Rooms when using $h_1^M$.

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 $h_1^M$. When using the cross-product heuristic $h_1^M$, LTL-LRTA*_A finds solutions that are around 1.2 times better. And when using $h_1^M$, 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 $h_1^M$ is a well-informed (although expensive) heuristic. In particular, $h_1^M$ 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 $\Delta$ 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*.

### Table 1: Results for different Maps-Heuristic combinations

<table>
<thead>
<tr>
<th>Configuration</th>
<th>No. best solution</th>
<th>Sol. gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heuristic</td>
<td>LTL</td>
<td>Ties</td>
</tr>
<tr>
<td>$h_1^M$</td>
<td>Rooms</td>
<td>1020</td>
</tr>
<tr>
<td></td>
<td>StarCraft</td>
<td>1213</td>
</tr>
<tr>
<td></td>
<td>Maze</td>
<td>1272</td>
</tr>
<tr>
<td>$h_1^M$</td>
<td>Rooms</td>
<td>841</td>
</tr>
<tr>
<td></td>
<td>StarCraft</td>
<td>461</td>
</tr>
<tr>
<td></td>
<td>Maze</td>
<td>588</td>
</tr>
<tr>
<td>$h_1^M$</td>
<td>Rooms</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>StarCraft</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>Maze</td>
<td>169</td>
</tr>
<tr>
<td>Total</td>
<td>5611</td>
<td>4360</td>
</tr>
</tbody>
</table>

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. In these domains, the agent can only move to empty cells, and each cell can either be 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 $\{a, b, \ldots, l\}$.

We evaluated the performance of LTL-LRTA* and LTL-LRTA*_A using two cross-product heuristics: $h_1^M$ and $h_1^M$, and one myopic heuristic: $\tilde{h}_1^M$. Intuitively, $h_1^M(s, q)$ 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. $h_1^M$ works similarly to $\tilde{h}_1^M(s, q)$, 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 $\tilde{h}_1^M$ 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$_I$ goals. These goals exploit a wide range of the features that LTL$_I$ 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$_I$ goals, 3
<table>
<thead>
<tr>
<th>Configuration</th>
<th>Time gap w.r.t. $h^1_{φ}$</th>
<th>Configuration</th>
<th>Solution gap w.r.t. $h^2_{φ}$</th>
</tr>
</thead>
<tbody>
<tr>
<td># items</td>
<td>Domain</td>
<td>$h^M_{φ}$</td>
<td>$h^M_{φ} + A$</td>
</tr>
<tr>
<td>3 items</td>
<td>Rooms</td>
<td>9.542</td>
<td>9.256</td>
</tr>
<tr>
<td></td>
<td>StarCraft</td>
<td>3.161</td>
<td>3.084</td>
</tr>
<tr>
<td></td>
<td>Maze</td>
<td>1.016</td>
<td>0.994</td>
</tr>
<tr>
<td></td>
<td>StarCraft</td>
<td>2.449</td>
<td>2.381</td>
</tr>
<tr>
<td></td>
<td>Maze</td>
<td>1.532</td>
<td>1.518</td>
</tr>
<tr>
<td>100 items</td>
<td>Rooms</td>
<td>2.233</td>
<td>2.379</td>
</tr>
<tr>
<td></td>
<td>StarCraft</td>
<td>2.359</td>
<td>2.303</td>
</tr>
<tr>
<td></td>
<td>Maze</td>
<td>1.936</td>
<td>1.992</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>2.512</td>
<td>2.501</td>
</tr>
</tbody>
</table>

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.

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 $O(|S_w|)$ whereas the complexity of computing the cross-product heuristic is $O(v^n)$, 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 $\tilde{h}^M_{φ}$. 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 $h^M_{φ}$
- $h^M_{φ} + A$ refers to LTL-LRTA*$_A$ using $h^M_{φ}$, and
- $\tilde{h}^M_{φ} + A$ refers to LTL-LRTA*$_A$ using $h^M_{φ}$.

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 $\tilde{h}^M_{φ}$ 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$_d$ and formally defined search problem with LTL$_d$ goals, which we called LSPs. We exploited the relationship between LTL$_d$ 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


