Proceedings of the Eighth International Symposium on Conbinatorial Search (SoCS-2015)

Finding and Exploiting LTL Trajectory Constraints in Heuristic Search

Salomé Simon and **Gabriele Röger**
University of Basel, Switzerland
{salome.simon,gabriele.roeger}@unibas.ch

Abstract

We suggest the use of linear temporal logic (LTL) for expressing declarative information about optimal solutions of search problems. We describe a general framework that associates LTL_f formulas with search nodes in a heuristic search algorithm. Compared to previous approaches that integrate specific kinds of path information like landmarks into heuristic search, the approach is general, easy to prove correct and easy to integrate with other kinds of path information.

Introduction

Temporal logics allow to formulate and reason about the development of logic-based systems, for example about paths in factored state spaces. These are for instance common in planning, where temporal logics have always been present. As one extreme, the entire planning task can be specified in a temporal logic language and plans are generated by theorem proving (Koehler and Treinen 1995) or model construction (Cerrito and Mayer 1998).

In a different approach, the planning system can exploit domain-specific search control knowledge, given as part of the input. Such control knowledge can be a temporal logical formula that every "meaningful" plan must satisfy, therefore reducing the size of the search space and speeding up the plan generation (Bacchus and Kabanza 2000; Doherty and Kvarnström 2001).

This is related to planning for *temporally extended goals* (e. g. Bacchus and Kabanza 1998; Kabanza and Thiébaux 2005), where a plan does not need to reach a *state* with a given goal property but the action *sequence* must satisfy a given temporal formula.

PDDL 3 *state trajectory constraints* (Gerevini and Long 2005) integrate both worlds by extending a fragment of linear temporal logic with an additional operator that allows to specify a classical goal property.

For all these approaches, the temporal formulas are part of the input. In contrast, Wang et al. (2009) generate temporal task-specific trajectory constraints in a fully automated fashion from landmark orderings (Hoffmann, Porteous, and Sebastia 2004; Richter and Westphal 2010). This information is used to derive better estimates with the FF heuristic (Hoffmann and Nebel 2001) by evaluating it on a modified task that makes the constraints visible to the heuristic.

We argue that trajectory constraints in propositional linear temporal logic on finite traces (LTL_f) are suitable for a much more extensive application in heuristic search: a unified way of describing path-dependent information inferred during the search. In planning, there are many techniques that exploit a specific type of information and maintain it with specialized data structures and implementations. For example, landmark heuristics (Richter and Westphal 2010; Karpas and Domshlak 2009; Pommerening and Helmert 2013) track the landmark status for each operator sequence. A unified formalism for these techniques would offer two main advantages: *decoupling the derivation and exploitation of information* and *easily combining different sources of information*.

Currently the derivation and exploitation of information are integrated in most cases: someone proposes a new source of information and shows how it can correctly be exploited (requiring new correctness proofs every time). In our framework, LTL_f formulas meeting a feasibility criterion provide an interface between derivation and exploitation. Thus, new sources of information only need to be proven to be feasible, while new ways of exploiting information only need to be proven to derive (path) admissible heuristics for any information meeting this criterion. This separation will also make it easier for practitioners in real-world applications to specify domain-specific knowledge and correctly exploit it without needing to know the details of heuristic search.

Due to the unified LTL_f representation in our framework, it is trivial to combine information from different sources. If heuristics are able to use any kind of feasible information, combining information will strengthen the heuristic without needing to adapt the heuristic itself.

In the rest of the paper, we first introduce the necessary background on LTL_f and the considered planning formalism. We define a feasibility criterion for LTL_f trajectory constraints in optimal planning that allows to combine constraints and to develop them over the course of actions. Afterwards, we demonstrate how feasible trajectory constraints can be derived from several established sources of information. We present a heuristic based on LTL_f constraints and show how the entire framework integrates with the A^* algorithm. We finish with an experimental evaluation.

Background

In linear temporal logic (LTL, Pnueli 1977) a *world* w over a set $\mathcal{P}$ of propositional symbols is represented as set $w \subseteq \mathcal{P}$ of the symbols that are *true* in w. LTL extends propositional logic (with operators $\neg$ and $\wedge$) with a unary operator $\bigcirc$ and a binary operator $\mathcal{U}$, which allow to formulate statements over infinite sequences of such worlds. Intuitively, $\bigcirc\varphi$ means that φ is true in the *next* world in the sequence, and $\varphi\mathcal{U}\psi$ means that in some future world ψ is true and *until* then φ is true.

In addition, we use the following common abbreviations:

- Standard abbreviations of propositional logic such as $\vee$ (*or*), $\rightarrow$ (*implies*), $\top$ (*true*), $\bot$ (*false*)
- $\Diamond\varphi = \top\mathcal{U}\varphi$ expresses that φ will *eventually* be true.
- $\Box\varphi = \neg\Diamond\neg\varphi$ expresses that from the current world on, φ will *always* be true.
- $\varphi\mathcal{R}\psi = \neg(\neg\varphi\mathcal{U}\neg\psi)$ expresses that ψ holds until φ holds as well (*releasing* ψ) or forever.[1]

LTL on finite traces (LTL$_f$, De Giacomo and Vardi 2013; De Giacomo, De Masellis, and Montali 2014) defines semantics for *finite* world sequences, using the additional abbreviation $last = \neg\bigcirc\top$, which is true exactly in the last world of the sequence.

Definition 1 (Semantics of LTL$_f$). *Let φ be an LTL$_f$ formula over a set of propositional symbols $\mathcal{P}$ and let $\boldsymbol{w} = \langle w_0, \ldots, w_n\rangle$ be a sequence of worlds over $\mathcal{P}$.*

For $0 \leq i \leq n$, we inductively define when φ is true at instant *i (written $\boldsymbol{w}, i \models \varphi$) as:*

- *For $p \in \mathcal{P}$, $\boldsymbol{w}, i \models p$ iff $p \in w_0$*
- *$\boldsymbol{w}, i \models \neg\psi$ iff $\boldsymbol{w}, i \not\models \psi$*
- *$\boldsymbol{w}, i \models \psi_1 \wedge \psi_2$ iff $\boldsymbol{w}, i \models \psi_1$ and $\boldsymbol{w}, i \models \psi_2$*
- *$\boldsymbol{w}, i \models \bigcirc\psi$ iff $i < n$ and $\boldsymbol{w}, i+1 \models \psi$*
- *$\boldsymbol{w}, i \models \psi_1\mathcal{U}\psi_2$ iff there exists a j with $i \leq j \leq n$ s.t. $\boldsymbol{w}, j \models \psi_2$ and for all $i \leq k < j$, $\boldsymbol{w}, k \models \psi_1$*

If $\boldsymbol{w}, 0 \models \varphi$, we say that φ is true in $\boldsymbol{w}$ or that $\boldsymbol{w}$ *satisfies* φ and also write this as $\boldsymbol{w} \models \varphi$.

In our application, we do not only want to reason about *finalized* world sequences but also about possible continuations of given prefixes. Bacchus and Kabanza (2000) showed for standard LTL with infinite world sequences how we can evaluate formulas *progressively* over the beginning $\langle w_0, \ldots, w_i\rangle$ of a world sequence. The idea is to create a formula which is satisfied by arbitrary continuations $\langle w_{i+1}, w_{i+2}, \ldots\rangle$ iff the original formula is satisfied by the entire sequence $\langle w_0, \ldots, w_i, w_{i+1}, \ldots\rangle$. As long as we do not progress over the last world in the sequence, the progression rules (shown in Figure 1) also work for LTL$_f$:

Proposition 1. *For an LTL$_f$ formula φ and a world sequence $\langle w_0, \ldots, w_n\rangle$ with $n > 0$ it holds that $\langle w_1, \ldots, w_n\rangle \models$ progress(φ, w_0) iff $\langle w_0, \ldots, w_n\rangle \models \varphi$.*

[1] While $\mathcal{R}$ can be expressed in terms of the essential operators, it is necessary for the positive normal form, which we will introduce and use later.

φ	**progress**(φ, w)
$p \in \mathcal{P}$:	$\top$ if $p \in w$, $\bot$ otherwise
$\neg\psi$:	$\neg$progress(ψ, w)
$\psi \wedge \psi'$:	progress$(\psi, w) \wedge$ progress(ψ', w)
$\psi \vee \psi'$:	progress$(\psi, w) \vee$ progress(ψ', w)
$\bigcirc\psi$:	ψ
$\Box\psi$:	progress$(\psi, w) \wedge \Box\psi$
$\Diamond\psi$:	progress$(\psi, w) \vee \Diamond\psi$
$\psi\mathcal{U}\psi'$:	progress$(\psi', w) \vee$ (progress$(\psi, w) \wedge (\psi\mathcal{U}\psi')$)
$\psi\mathcal{R}\psi'$:	progress$(\psi', w) \wedge$ (progress$(\psi, w) \vee (\psi\mathcal{R}\psi')$)
last:	$\bot$
$\top$:	$\top$
$\bot$:	$\bot$

Figure 1: Progression Rules

For an example, consider formula $\varphi = \bigcirc x \wedge \Diamond z$ over $\mathcal{P} = \{x, y, z\}$ stating that in the following world proposition x should be true and at some point z should be true. The progression of φ with a world $w = \{x, z\}$ is

$$\begin{aligned}\text{progress}(\bigcirc x \wedge \Diamond z, w) &= \text{progress}(\bigcirc x, w) \wedge \\ &\quad\ \text{progress}(\Diamond z, w) \\ &= x \wedge (\text{progress}(z, w) \vee \Diamond z) \\ &= x \wedge (\top \vee \Diamond z) \equiv x\end{aligned}$$

The progression eliminates the $\Diamond z$ from the formula because it is already satisfied with w. Subformula $\bigcirc x$ gets replaced with x, which fits the requirement that x should now be true if φ is satisfied by the overall world sequence.

In the special case where a progression results to $\bot$ (or $\top$), it is clear that no (or any) continuation will result in an overall world sequence that satisfies the original formula.

We consider search problems given as *STRIPS planning tasks* with action costs. Such a planning task is a tuple $\Pi = \langle V, A, I, G\rangle$, where V is a set of propositional *state variables*, A is a set of *actions*, $I \subseteq V$ is the *initial state* and $G \subseteq V$ is the *goal description*. An *action* $a = \langle pre(a), add(a), del(a), c(a)\rangle \in A$ consists of the *precondition* $pre(a) \subseteq V$, the *add effects* $add(a) \subseteq V$, the *delete effects* $del(a) \subseteq V$ and the *action cost* $c(a) \in \mathbb{R}_0^+$. A *state* $s \subseteq V$ is defined by a subset of the variables. Action a is *applicable in state* s if $pre(a) \subseteq s$. Applying a in s leads to the *successor state* $s[a] = (s \setminus del(a)) \cup add(a)$. A *path* is a sequence of actions $\pi = \langle a_1, \ldots, a_n\rangle$ and is applicable in state s if the actions are applicable consecutively. We denote the resulting state with $s[\langle a_1, \ldots, a_n\rangle]$. For the empty path, $s[\langle\rangle] = s$. The *cost* of the path is $c(\pi) = \sum_{i=1}^{n} c(a_i)$. A *plan* is a path π such that $G \subseteq I[\pi]$. It is *optimal* if it has minimal cost.

We will formulate LTL$_f$ trajectory constraints that do not only cover state variables but also action applications. We follow the approach by Calvanese, De Giacomo, and Vardi (2002) and introduce an additional variable a for each action of the planning task. Therefore, the set $\mathcal{P}$ comprises these action variables plus the state variables of the planning task.

For the concatenation of two finite sequences σ and σ', we write $\sigma\sigma'$. For example, $\langle w_1, \ldots, w_n\rangle\langle w'_1, \ldots, w'_m\rangle = \langle w_1, \ldots, w_n, w'_1, \ldots, w'_m\rangle$. We use this notation for action and world sequences.

Feasible LTL$_f$ Formulas for Optimal Planning

Graph search algorithms like A* operate on search nodes n that associate a state $s(n)$ with a path of cost $g(n)$ from the initial state I to this state. We want to associate search nodes with LTL$_f$ trajectory constraints that characterize how the path to the node should be continued. This information can then be exploited for deriving heuristic estimates.

Such per node LTL$_f$ constraints are suitable for optimal planning if they are satisfied by any continuation of the path to the node into an optimal plan. To capture this notion, we define the world sequence induced by path $\rho = \langle a_1, \ldots, a_n\rangle$ in state s as $\boldsymbol{w}^{\boldsymbol{s}}_{\boldsymbol{\rho}} = \langle\{a_1\} \cup s[a_1], \{a_2\} \cup s[\langle a_1, a_2\rangle], \ldots, \{a_n\} \cup s[\rho], s[\rho]\rangle$ and introduce the following feasibility criterion:

Definition 2 (Feasibility for nodes). *Let Π be a planning task with goal G and optimal plan cost h^* and let n be a node in the search space that is associated with state s.*

An LTL$_f$ formula φ is feasible for n *if for all paths ρ such that*

- *ρ is applicable in s,*
- *the application of ρ leads to a goal state ($G \subseteq s[\rho]$), and*
- *$g(n) + c(\rho) = h^*$*

it holds that $\boldsymbol{w}^{\boldsymbol{s}}_{\boldsymbol{\rho}} \models \varphi$.

If the path to node n is not a prefix of an optimal plan, then any formula is feasible for n. Otherwise, φ must be true for any continuation of the path into an optimal plan but not necessarily for continuations into suboptimal plans.

If a formula is feasible for a node, its progression is feasible for the corresponding successor node.

Theorem 1. *Let φ be a feasible formula for a node n, and let n' be the successor node reached from n with action a. Then progress$(\varphi, \{a\} \cup s(n'))$ is feasible for n'.*

Proof. Let R_a be the set of paths that continue the history of n to an optimal plan (and are therefore relevant for the feasibility of φ) and in addition begin with action a. If R_a is empty, the path leading to n' cannot be continued to an optimal plan and therefore any formula is feasible for n'. Otherwise let ρ' be a path that continues the path to n' to an optimal plan. Then $\rho = \langle a\rangle\rho'$ is in R_a and $\boldsymbol{w}^{\boldsymbol{s(n)}}_{\boldsymbol{\rho}} \models \varphi$. Since $\boldsymbol{w}^{\boldsymbol{s(n)}}_{\boldsymbol{\rho}} = \langle\{a\} \cup s(n')\rangle\boldsymbol{w}^{\boldsymbol{s(n')}}_{\boldsymbol{\rho'}}$, Proposition 1 implies that $\boldsymbol{w}^{\boldsymbol{s(n')}}_{\boldsymbol{\rho'}} \models \text{progress}(\varphi, \{a\} \cup s(n'))$. □

We also have the opportunity to incorporate additional feasible trajectory constraints: if we can derive a new feasible formula φ_{new} for a node, we can safely combine it with the progressed formula $\varphi_{\text{progress}}$ to a feasible formula $\varphi_{\text{progress}} \wedge \varphi_{\text{new}}$.

Since graph search algorithms eliminate nodes with duplicate states, a strategy for combining feasible formulas from nodes with identical state is desirable. Instead of only keeping the formula of the preserved node, we can exploit the information of two paths of equal cost by combining the two feasible formulas:

Theorem 2. *Let n and n' be two search nodes such that $g(n) = g(n')$ and $s(n) = s(n')$. Let further φ_n and $\varphi_{n'}$ be feasible for the respective node. Then $\varphi_n \wedge \varphi_{n'}$ is feasible for both n and n'.*

Proof sketch. Whether a formula is feasible for a node only depends on the set of relevant paths characterized in Definition 2. The node only influences this characterization with its g-value and its associated state s. Therefore, a formula is either feasible for *all* nodes that agree on these components or for *none* of them. Feasibility of the conjunction follows directly from the LTL$_f$ semantics. □

Feasible formulas for a node only talk about the future but do not cover the current state. This can lead to an needlessly complicated specification of information that is derived before starting the search. For example, the notion of landmarks is defined for the entire state sequence traversed by a plan – including the initial state. We therefore also introduce *feasibility for tasks* that allows to naturally formulate information about this entire state sequence.

Definition 3 (Feasibility for tasks). *Let Π be a planning task with initial state I. An LTL$_f$ formula φ is* feasible *for Π if $\langle I\rangle\boldsymbol{w}^{\boldsymbol{I}}_{\boldsymbol{\pi^*}} \models \varphi$ for all optimal plans π^*.*

Progressing a feasible formula for a task with its initial state yields a feasible formula for the initial search node.

Finding Feasible LTL$_f$ Trajectory Constraints

In this section we will demonstrate how existing examples from the literature can be used to derive feasible LTL$_f$ trajectory constraints. We will present details for landmarks and unjustified action applications. To give an intuition for the scope of the framework, we will also briefly comment on further possibilities.

Fact Landmarks and Landmark Orderings

A (fact) landmark (Hoffmann, Porteous, and Sebastia 2004) is a state variable that must be true at some point in every plan. A landmark ordering specifies that before a landmark becomes true some other landmark must have been true.

There are several methods in the literature to extract such landmark information for a given planning task (Zhu and Givan 2003; Hoffmann, Porteous, and Sebastia 2004; Richter and Westphal 2010; Keyder, Richter, and Helmert 2010).

Wang, Baier, and McIlraith (2009) already encoded landmark orderings in LTL. Since in our scenario, progression notices when a landmark has been reached, we can use a slightly more compact formalization.

We base our definitions of landmarks and landmark orderings on the notions by Richter and Westphal (2010).

A state variable p is a *(fact) landmark* of a planning task Π if the application of every plan π of Π visits some state

that contains p. Therefore, $\Diamond p$ is a feasible formula for the task if p is a landmark.

There are three types of sound landmark orderings.

A *natural* ordering $l \rightarrow_{\text{nat}} l'$ states that if l' becomes true for the first time at time step i, then l must be true at some time step $j < i$. This can be expressed as $\neg l' \mathcal{U}(l \wedge \neg l')$ because l' can only become true after l was true.

A *greedy-necessary* ordering $l \rightarrow_{\text{gn}} l'$ states that if l' becomes true for the first time at step i, then l must be true at time step $i-1$. This can be formulated as $\neg l' \mathcal{U}(l \wedge \neg l' \wedge \bigcirc l')$ because l must be true directly before l' becomes true for the first time.

A *necessary* ordering $l \rightarrow_{\text{nec}} l'$ states that for each time step i where l' becomes true, l must be true at time step $i-1$. Put differently, whenever l' becomes true in the next step, then currently l must be true: $\Box(\neg l' \wedge \bigcirc l' \rightarrow l)$.

There are important other landmark-related notions for which we are not aware of any previous LTL encodings.

One such relevant source of information are the first achievers of a landmark. A *first achiever set* FA_l for a landmark l is a set of actions such that in any plan one of these actions makes the landmark true for the first time. For a first achiever set FA_l the formula $l \vee \bigvee_{a \in FA_l} \Diamond a$ describes that if the landmark is not initially true, then one of the actions in the set needs to be applied.

The landmark count heuristic (Richter and Westphal 2010) counts how many landmarks have not yet been reached (i. e., they have not been true on the path to the node) and how many reached landmarks are required again. A reached landmark l is required again if l is false in the current state and there is a greedy-necessary (or necessary) ordering $l \rightarrow l'$ where l' has not yet been reached. Additionally, any reached landmark l that is a goal proposition and false in the current state is required again.

We can formulate these conditions for required again landmarks as $(\Diamond l)\mathcal{U}l'$ for greedy-necessary orderings $l \rightarrow_{\text{gn}} l'$ and as $(\Diamond g)\mathcal{U} \bigwedge_{g' \in G} g'$ for goal landmarks g and goal specification G.

All these formulas can be combined into a single LTL_f formula that is feasible for the task:

Proposition 2. *For planning task* Π *with goal* G*, let* L *be a set of landmarks and let* O_{nat}, O_{gn}*, and* O_{nec} *be sets of natural, greedy-necessary and necessary landmark orderings, respectively. Let* FA *be a set of first achiever sets for landmarks. The following formula* φ *is feasible for* Π*.*

$$\varphi = \varphi_{lm} \wedge \varphi_{fa} \wedge \varphi_{nat} \wedge \varphi_{gn} \wedge \varphi_{nec} \wedge \varphi_{ra} \wedge \varphi_{goal}$$

where

- $\varphi_{lm} = \bigwedge_{l \in L} \Diamond l$
- $\varphi_{fa} = \bigwedge_{FA_l \in FA} \left(l \vee \bigvee_{a \in FA_l} \Diamond a\right)$
- $\varphi_{nat} = \bigwedge_{l \rightarrow_{nat} l' \in O_{nat}} \left(\neg l' \mathcal{U}(l \wedge \neg l')\right)$
- $\varphi_{gn} = \bigwedge_{l \rightarrow_{gn} l' \in O_{gn}} \left(\neg l' \mathcal{U}(l \wedge \neg l' \wedge \bigcirc l')\right)$
- $\varphi_{nec} = \bigwedge_{l \rightarrow_{nec} l' \in O_{nec}} \Box(\neg l' \wedge \bigcirc l' \rightarrow l)$
- $\varphi_{ra} = \bigwedge_{l \rightarrow l' \in O_{gn} \cup O_{nec}} \left((\Diamond l)\mathcal{U}l'\right)$
- $\varphi_{goal} = \bigwedge_{g \in G} \left((\Diamond g)\mathcal{U} \bigwedge_{g' \in G} g'\right)$

The subformula $\varphi_{\text{lm}} \wedge \varphi_{\text{ra}} \wedge \varphi_{\text{goal}}$ also provides an alternative way of computing the inadmissible landmark count heuristic, which can only be used for planning without optimality guarantee: We determine this task-feasible formula from the same precomputation as performed by the landmark count heuristic and progress it in our framework. The heuristic estimate for a node can then be determined from the associated feasible formula as the cardinality of the set of all state variables that occur within a $\Diamond$ formula but are false in the state of the node.

Action Landmarks

A *(disjunctive) action landmark* is a set of actions of which at least one must occur in any plan.

If $\mathcal{L}$ is a set of action landmarks for state s' then $\varphi_{\mathcal{L}} = \bigwedge_{L \in \mathcal{L}} \Diamond(\bigvee_{a \in L} a)$ is feasible for all nodes n with $s(n) = s'$.

One example for an action landmark is the set of first achievers for a landmark that is currently not true. Also the LM-Cut heuristic (Helmert and Domshlak 2009) derives a set of action landmarks as a byproduct of the heuristic computation. The *incremental* LM-Cut heuristic (Pommerening and Helmert 2013) stores and progresses this set to speed up the LM-Cut computation for the successor states. At the application of an action a, incremental LM-Cut creates a new landmark set for the successor state by removing all action landmarks that contain a. Let $\mathcal{L}$ and $\mathcal{L}'$ be the respective landmark sets before and after the action application. The progression of $\varphi_{\mathcal{L}}$ over a is logically equivalent to $\varphi_{\mathcal{L}'}$, so LTL_f progression reflects the landmark-specific progression by incremental LM-Cut.

Unjustified Action Applications

The key idea behind *unjustified action applications* (Karpas and Domshlak 2011; 2012) is that every action that occurs in a plan should contribute to the outcome of the plan – by enabling another action in the plan or by making a goal proposition finally true.

The definition is based on the notion of *causal links*: a path $\rho = \langle a_1, \ldots, a_n \rangle$ has a *causal link* between the i-th and the j-th action application if $i < j$, a_i adds a proposition p which stays true and is not added again until step $j-1$, and p is a precondition of a_j. If there is a link between the i-th and a later action application in a plan π or the i-th action adds a goal proposition that is not added again later, then the i-th action application is *justified*, otherwise it is *unjustified*.

A plan π with an unjustified application of a positive-cost action a cannot be optimal because removing this action application from π results in a cheaper plan π'.

The notion of unjustified action applications is defined for *entire plans* but during search the question is whether the current path can be extended to a plan without unjustified action applications and how we can characterize such an extension. The relevant information can easily be encoded within the LTL_f framework: if the last action application is justified, at least one of the add effects must stay true and cannot be added again until it is used as a precondition or for satisfying the goal. Since we want to preserve *all* optimal plans, we do not exploit this information for zero-cost actions.

Theorem 3. *Let $\Pi = \langle V, A, I, G\rangle$ be a planning task and let n be a search node that was reached with a positive-cost action a. Then the following formula φ_a is feasible for n:*

$$\varphi_a = \bigvee_{e \in add(a) \setminus G} \Big((e \wedge \bigwedge_{\substack{a' \in A \text{ with} \\ e \in add(a')}} \neg a') \mathcal{U} \bigvee_{\substack{a' \in A \text{ with} \\ e \in pre(a')}} a' \Big) \vee \bigvee_{e \in add(a) \cap G} \Big((e \wedge \bigwedge_{\substack{a' \in A \text{ with} \\ e \in add(a')}} \neg a') \mathcal{U} (\mathit{last} \vee \bigvee_{\substack{a' \in A \text{ with} \\ e \in pre(a')}} a') \Big)$$

Proof sketch. Let ρ denote the path that lead to n. Assume that φ_a is not feasible for n, so there is a path ρ' that satisfies the conditions from Definition 2 but $\boldsymbol{w}_{\rho'}^{\boldsymbol{s(n)}} \not\models \varphi_a$. Then ρ' does not use any add effect of a before it gets deleted or added again by another action application. Also each goal atom added by a is deleted or added again by ρ'. Therefore the application of a in the plan $\rho\rho'$ is unjustified. As $c(a) > 0$ there exists a cheaper plan and $g(n)+c(\rho')$ cannot be equal to the optimal plan cost of Π. This is a contradiction to ρ' satisfying the conditions from Definition 2. □

The original implementation of unjustified action applications requires an analysis of the causal links and resulting causal chains of the action sequence. All information required for this reasoning is encoded in the LTL$_f$ formulas and standard LTL$_f$ progression replaces this analysis of the causal interactions.

We could even go further: Instead of adding a feasible formula φ_a after each application of action a, we could also extend the feasible formula for the initial node with a conjunction $\bigwedge_{a \in A} a \rightarrow \bigcirc \varphi_a$, ranging over the set of all actions A, and let the progression do the rest. However, since planning tasks can have thousands of actions, this would lead to significant overhead in the progression.

Other Sources of Information

The scope of our approach is by far not limited to the previously introduced sources of information. Since space is limited, we only briefly mention some other ideas.

One obvious possibility is the integration of previous LTL methods like hand-written (Bacchus and Kabanza 2000; Doherty and Kvarnström 2001) or learned (de la Rosa and McIlraith 2011) LTL search control knowledge.

Also invariants such as mutex information can be added to the LTL$_f$ formula to make them explicit to the heuristic computation. The same holds for the fact that at the end the goal G must be true ($\Diamond(\mathit{last} \wedge \bigwedge_{g \in G} g)$).

The recent flow-based heuristics (van den Briel et al. 2007; Bonet 2013; Pommerening et al. 2014) build on the observation that a variable cannot be (truly) deleted more often than it is made true (with some special cases for the initial state and the goal) but they ignore the order of the corresponding action applications. With LTL$_f$ formulas we can express statements of the same flavor but preserving parts of the ordering information. For an intuition, consider a formula that states that whenever we apply an action deleting p and later apply an action requiring p, we in between have to apply an action adding p.

In principle, we could go as far as encoding the entire planning task in the LTL formula (Cerrito and Mayer 1998). The challenge with the framework will be to find a suitable balance of the incurred overhead and the gain in heuristic information.

Deriving Heuristic Estimates from Feasible LTL$_f$ Trajectory Constraints

Deriving heuristic estimates from feasible LTL$_f$ trajectory constraints is an interesting research question, which cannot finally be answered in this paper. For the moment, we only present a proof-of-concept heuristic that extracts landmarks, essentially ignoring the temporal information carried by the LTL$_f$ formula. However, the temporal aspects of the LTL$_f$ formulas are still important for the heuristic estimate because they preserve information over the course of progression. In the future we also want to investigate methods that are based on LTL reasoning and which are therefore able to derive stronger estimates from the temporal information.

Although we extract landmarks from the input LTL$_f$ formula, this does not mean that this LTL$_f$ formula must stem from landmark information. The heuristic is correct for any kind of feasible LTL$_f$ formulas.

The heuristic computation first derives so-called *node-admissible disjunctive action landmarks* from the LTL$_f$ formula. The heuristic estimate is then determined from these landmarks with the landmark heuristic by Karpas and Domshlak (2009).

As introduced earlier, a disjunctive action landmark for a state s is a set of actions such that *every* path from s to a goal state contains at least one action from the set. We use a weaker path-dependent notion that covers all *optimal* plans:

Definition 4. *Let $\Pi = \langle V, A, I, G\rangle$ be a planning task and n be a search node. A set $L \subseteq A$ is a* node-admissible disjunctive action landmark *for n if every continuation of the path to n into an optimal plan contains an action from L.*

Using such node-admissible disjunctive action landmarks in the landmark heuristic gives admissible estimates for all nodes that correspond to a prefix of an optimal plan. Karpas and Domshlak (2012) call this property *path admissible*.

Our method of extracting node-admissible landmarks from LTL$_f$ formulas requires the formula to be in positive normal form (also called negation normal form), where $\neg$ only appears in literals or before *last*. This is uncritical because any LTL$_f$ formula can efficiently be transformed into positive normal form with De Morgan's law and the following equivalences:

$$\neg\bigcirc\varphi \equiv \mathit{last} \vee \bigcirc\neg\varphi \qquad \neg\Box\varphi \equiv \Diamond\neg\varphi$$
$$\neg(\varphi_1 \mathcal{U} \varphi_2) \equiv (\neg\varphi_1)\mathcal{R}(\neg\varphi_2) \qquad \neg\Diamond\varphi \equiv \Box\neg\varphi$$
$$\neg(\varphi_1 \mathcal{R} \varphi_2) \equiv (\neg\varphi_1)\mathcal{U}(\neg\varphi_2)$$

Moreover, progression preserves the normal form.

For the landmark extraction from the feasible formula, we first derive an implied LTL$_f$ formula (Proposition 3) in conjunctive normal form. We then extract node-admissible disjunctive action landmarks from its clauses (Theorem 4).

Proposition 3. *Let φ be an LTL$_f$ trajectory constraint in negation normal form. The following function lm defines an LTL$_f$ formula such that $\varphi \models lm(\varphi)$:*

$$\begin{aligned} lm(x) &= \Diamond x \text{ for literals } x \\ lm(last) &= \Diamond last \\ lm(\neg last) &= \Diamond \neg last \\ lm(\varphi \wedge \psi) &= lm(\varphi) \wedge lm(\psi) \\ lm(\varphi \vee \psi) &= lm(\varphi) \vee lm(\psi) \\ lm(\bigcirc \varphi) &= lm(\Box \varphi) = lm(\Diamond \varphi) = lm(\varphi) \\ lm(\varphi \mathcal{U} \psi) &= lm(\varphi \mathcal{R} \psi) = lm(\psi) \end{aligned}$$

The proposition can easily be checked from the semantics of LTL$_f$. By distributing $\vee$ over $\wedge$, we can transform the formula into *conjunctive normal form* (CNF) $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} \Diamond \varphi_{i,j}$, where each $\varphi_{i,j}$ is a literal, *last*, or $\neg$*last*. Clauses containing $\Diamond$*last* are tautologies and therefore not valuable for the heuristic. Clauses containing $\Diamond\neg$*last* are trivially true for each world sequence of length at least two, which is the case for the induced world sequences for any path leading from a non-goal state to a goal state. Therefore, we derive the action landmarks only from the remaining clauses.

Theorem 4. *Let $\Pi = \langle V, A, I, G \rangle$ be a planning task and let φ be a feasible LTL$_f$ trajectory constraint for node n. Let further $\psi = \bigvee_{j=1}^{m} \Diamond x_j$ (with x_j being literals) be a formula such that $\varphi \models \psi$.*

If progress$(\psi, s(n)) \not\equiv \top$, then $L = \bigcup_{j=1}^{m}$ support(x_j) with

$$\textit{support}(x) = \begin{cases} \{a \in A \mid x \in add(a)\} & \text{if } x \in V \\ \{a \in A \mid x \in del(a)\} & \text{if } \bar{x} \in V \\ \{x\} & \text{if } x \in A \end{cases}$$

is a node-admissible disjunctive action landmark for n.

Proof. Since $\varphi \models \psi$, ψ also is feasible for n. By the semantics of $\Diamond$, at least one x_j must be true in some world in any continuation to an optimal plan. If the progression is not valid, then no state-variable literal x_j is already true in the current state.[2] Thus, one of the x_j needs to become true in any optimal plan. For a proposition p, this means that p must be added by an action. Similarly, for a negated proposition $\neg p$, the proposition p must be deleted. An action variable a requires the action to be applied. Therefore, any continuation of the path to n into an optimal plan contains an action from L. □

Our proposed heuristic is the landmark heuristic computed from all node-admissible disjunctive action landmarks that can be derived from the trajectory constraints as described in Proposition 3 and Theorem 4. In the special case where the LTL$_f$ formula is detected unsatisfiable (simplifies to $\bot$), the heuristic returns ∞ because the path to the node cannot be extended into an optimal plan.

[2]Moreover, none of the x_j is a negated action variable. Although these would not prevent the clause from generating an action landmark, the landmark would be very weak.

A* with LTL$_f$ Trajectory Constraints

Since LTL$_f$ trajectory constraints are path-dependent and the heuristic is not admissible but only path admissible, we need to adapt the A* algorithm so that it still guarantees optimal plans. This is only a small modification: whenever a cheaper path to a state has been found, we need to recompute the heuristic estimate with the new feasible information instead of using a cached estimate (Karpas and Domshlak 2012). This leads to a different treatment of infinite heuristic estimates. We also use the opportunity to show the integration of feasible LTL$_f$ formulas.

Algorithm 1 shows the adapted A* algorithm. We formulated the algorithm with "eager" duplicate elimination so that there is always at most one node for each state. Besides the planning task, the algorithm takes a path admissible heuristic function as input that computes the estimate from a state and an LTL$_f$ formula.

We use a method *taskFeasibleFormula*(Π) that returns a feasible formula for Π, and a method *feasibleFormula*(Π, π) that generates a (path-dependent) feasible formula for the node reached by path π.

The feasible formula for the task (line 3) is progressed with the initial state to receive a feasible formula φ for the initial search node, which can be further strenghened with any other feasible formula for this node (line 4). If the heuristic for the initial state and this formula is ∞ then the task is unsolvable. Otherwise, we create the initial node and add it to the open list (lines 6–8).

When generating the successor n' of node n with action a, we first progress the LTL$_f$ formula of n to get a new feasible formula for n', based on Theorem 1 (line 17). Based on Theorem 2, this formula can be strengthened with a newly derived LTL$_f$ formula that is feasible for this node (line 18). If we do not want to incorporate additional information, the method can simply return $\top$.

If we encounter the successor state for the first time, we create a new node n' with the respective properties (line 21). Otherwise, we distinguish three cases for the previously best node for this state. If we already found a better path to this state, we skip this successor (line 24). If we previously found an equally good path to the state, we strengthen the LTL$_f$ formula of the previous node based on Theorem 2 and recompute the heuristic estimate (lines 26–28). If the previously best path to the state was worse, we update the node with the data from the newly found path (lines 30–33).

In contrast to an admissible heuristic, with a path admissible heuristic it is not safe to prune a *state* from the search space if the heuristic estimate via one path (of cost g') is ∞. However, we can conclude that no optimal plan traverses the state with a cost of at least g'. To exploit this pruning power for nodes encountered later, we do not discard a node with infinite heuristic estimate but store it in the closed list (lines 35–36). If the node has a finite estimate, it gets enqueued in the open list (line 38).

Experimental Evaluation

For the experimental evaluation, we implemented the LTL$_f$ framework on top of Fast Downward (Helmert 2006). We

Algorithm 1: A* with LTL$_f$ trajectory constraints

input : Planning task $\Pi = \langle V, A, I, G \rangle$ and path-admissible heuristic function h
output: Optimal plan π or *unsolvable* if Π is unsolvable

1 *open* $\leftarrow$ empty priority queue of nodes
2 *closed* $\leftarrow \emptyset$
3 $\varphi_\Pi \leftarrow$ *taskFeasibleFormula*(Π)
4 $\varphi \leftarrow$ *progress*$(\varphi_\Pi, I) \wedge$ *feasibleFormula*$(\Pi, \langle\rangle)$
5 *hval* $\leftarrow h(I, \varphi)$
6 **if** *hval* $\neq \infty$ **then**
7 $\quad n \leftarrow$ new node with $n.state = I$, $n.g = 0$, $n.h =$ *hval*, $n.\varphi = \varphi$ and $n.parent = \perp$
8 $\quad$ add n to *open* with priority *hval*
9 **while** *open is not empty* **do**
10 $\quad n \leftarrow$ remove min from open
11 $\quad s \leftarrow n.state$
12 $\quad$ **if** s *is goal state* **then**
13 $\quad\quad$ **return** *extractPlan*(n)
14 $\quad$ add n to *closed*
15 $\quad$ **for** *all actions* a *applicable in* s **do**
16 $\quad\quad s' \leftarrow s[a]$
17 $\quad\quad \varphi' \leftarrow$ *progress*$(n.\varphi, \{a\} \cup s')$
18 $\quad\quad \varphi' \leftarrow \varphi' \wedge$ *feasibleFormula*$(\Pi,$ path to s' via $n)$
19 $\quad\quad g' \leftarrow n.g + c(a)$
20 $\quad\quad$ **if** *there exists no node* n' *with* $n'.state = s'$ **then**
21 $\quad\quad\quad n' \leftarrow$ new node with $n'.state = s'$, $n'.g = g'$, $n'.h = h(s', \varphi')$, $n'.\varphi = \varphi'$ and $n'.parent = n$
22 $\quad\quad$ **else**
23 $\quad\quad\quad n' \leftarrow$ unique node in *open* or *closed* with $n'.state = s'$
24 $\quad\quad\quad$ **if** $g' > n'.g$ **then continue**
25 $\quad\quad\quad$ remove n' from *open*/*closed*
26 $\quad\quad\quad$ **if** $g' = n'.g$ **then**
27 $\quad\quad\quad\quad n'.\varphi \leftarrow n'.\varphi \wedge \varphi'$
28 $\quad\quad\quad\quad n'.h \leftarrow h(s', n'.\varphi)$
29 $\quad\quad\quad$ **else**
30 $\quad\quad\quad\quad n'.\varphi \leftarrow \varphi'$
31 $\quad\quad\quad\quad n'.g \leftarrow g'$
32 $\quad\quad\quad\quad n'.h \leftarrow h(s', \varphi')$
33 $\quad\quad\quad\quad n'.parent \leftarrow n$
34 $\quad\quad$ **end**
35 $\quad\quad$ **if** $n'.h = \infty$ **then**
36 $\quad\quad\quad$ add n' to *closed*
37 $\quad\quad$ **else**
38 $\quad\quad\quad$ add n' to *open* with priority $n'.g + n'.h$
39 $\quad\quad$ **end**
40 $\quad$ **end**
41 **end**
42 **return** unsolvable

conducted all experiments with a memory limit of 4 GB and a time limit of 30 minutes (excluding Fast Downward's translation and preprocessing phase).

Our heuristic derives disjunctive action landmarks from LTL$_f$ constraints as input for the admissible landmark heuristic (Karpas and Domshlak 2009). To evaluate the overhead of our approach, we compare it to the standard implementation of the landmark heuristic with specialized data structures exploiting the same (initial) landmark information. In both cases, the landmark heuristic applies optimal cost-partitioning for computing the estimate.

For the landmark generation, we use the same approach as the BJOLP planner (Domshlak et al. 2011), combining landmark information from two generation methods (Richter and Westphal 2010; Keyder, Richter, and Helmert 2010).

For the LTL$_f$ approach, we generate an initial feasible LTL$_f$ formula from the first achiever and the required again formulas (corresponding to $\varphi_{fa}, \varphi_{ra}$, and φ_{goal} in Proposition 2) for these landmarks. We do not use the landmark and ordering formulas because they would not additionally contribute to the heuristic estimates. We use this LTL$_f$ formula as shown in Algorithm 1, not incorporating additional information in line 18. In the following, we refer to this setup as LTL-A* with $h^{\text{LM}}_{\text{AL}}$.

A meaningful application of the standard implementation of the landmark heuristic requires the search algorithm LM-A* (Karpas and Domshlak 2009) that extends A* with multi-path support for landmarks. We refer to this setup as LM-A* with h_{LA}.

The comparison of these two approaches is not entirely fair because LM-A* combines information from *all* paths to a state while LTL-A* only combines formulas from equally expensive paths. Thus, with a comparable search history, LM-A* can sometimes derive better heuristic estimates.

Table 1 shows results for the STRIPS benchmarks of the International Planning Competitions 1998–2011.

Overall, LM-A* with h_{LA} solves 723 tasks and LTL-A* with $h^{\text{LM}}_{\text{AL}}$ finds solutions for 711 tasks. All unsolved instances of the former are due to the time limit. With the LTL$_f$ implementation 11 instances fail due to the memory limit with 9 of them being airport instances.

To get a clearer idea of the memory overhead of the approach, we summed up the memory consumption of all commonly solved instances of each domain. The percentage in parentheses shows the fraction of these two values where numbers above 100% indicate that the LTL$_f$ approach required more memory. A positive surprise is that more often than not our approach requires less memory. However, there are also cases, where the increase in memory consumption is significant, for example in the logistics-00 domain where the LTL$_f$ implementation needs more than three times the amount of the specialized implementation. This result is not caused by the unfavorable comparison of the approaches because the expansion numbers in both cases are identical. Nevertheless, the memory consumption only is responsible for a single unsolved task in this domain because 7 of the 8 affected instances fail due to a timeout.

	LM-A*	LTL-A*		
	h_{LA}	$h_{\text{AL}}^{\text{LM}}$		$h_{\text{AL}}^{\text{LM+UAA}}$
airport (50)	31	28	(335%)	26
barman (20)	0	0	(−%)	0
blocks (35)	26	26	(107%)	26
depot (22)	7	7	(86%)	7
driverlog (20)	14	14	(88%)	14
elevators-08 (30)	14	14	(78%)	13
elevators-11 (20)	11	11	(77%)	11
floortile (20)	2	2	(95%)	4
freecell (80)	52	51	(123%)	50
grid (5)	2	2	(108%)	2
gripper (20)	6	6	(187%)	6
logistics-00 (28)	20	20	(327%)	20
logistics-98 (35)	5	5	(99%)	5
miconic (150)	141	141	(116%)	141
mprime (35)	19	19	(90%)	20
mystery (30)	15	15	(83%)	15
nomystery (20)	18	17	(147%)	16
openstacks-08 (30)	14	12	(200%)	12
openstacks-11 (20)	9	7	(229%)	7
openstacks (30)	7	7	(107%)	7
parcprinter-08 (30)	15	14	(149%)	14
parcprinter-11 (20)	11	10	(152%)	10
parking (20)	1	1	(121%)	1
pathways (30)	4	4	(98%)	4
pegsol-08 (30)	26	26	(155%)	26
pegsol-11 (20)	16	16	(174%)	16
pipesworld-notan (50)	17	17	(91%)	17
pipesworld-tan (50)	9	10	(98%)	10
psr-small (50)	49	49	(87%)	49
rovers (40)	7	7	(91%)	7
satellite (36)	7	7	(86%)	7
scanalyzer-08 (30)	10	9	(111%)	9
scanalyzer-11 (20)	6	6	(114%)	6
sokoban-08 (30)	22	21	(76%)	22
sokoban-11 (20)	18	18	(76%)	18
tidybot (20)	14	14	(103%)	13
tpp (30)	6	6	(95%)	6
transport-08 (30)	11	11	(90%)	11
transport-11 (20)	6	6	(86%)	6
trucks (30)	7	7	(82%)	7
visitall (20)	16	16	(136%)	16
woodworking-08 (30)	14	14	(80%)	14
woodworking-11 (20)	9	9	(78%)	9
zenotravel (20)	9	9	(93%)	9
Sum (1396)	723	711		709

Table 1: Results for LM-A* with the standard landmark heuristic and for LTL-A* using a feasible landmark-based constraint ($h_{\text{AL}}^{\text{LM}}$) and using additional feasible LTL$_f$ constraints from unjustified action applications ($h_{\text{AL}}^{\text{LM+UAA}}$). The percentage in parentheses shows the memory consumption on the commonly solved instances compared to the first configuration. All other numbers show coverage results.

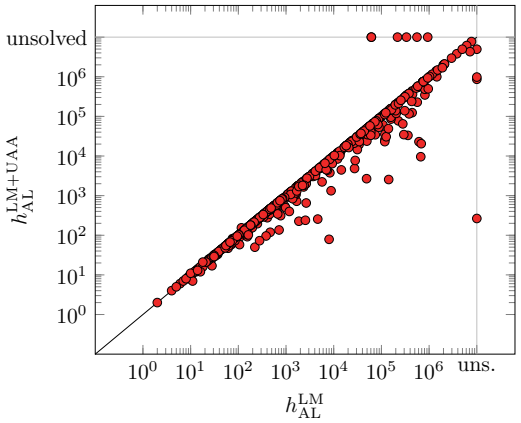

Figure 2: Comparison of expansions.

In a second experiment, we include in addition feasible LTL$_f$ trajectory constraints as described in the section on unjustified action applications. We denote the resulting heuristic $h_{\text{AL}}^{\text{LM+UAA}}$. Coverage results are shown in the last column of Table 1. Overall, the inclusion of the additional feasible information leads to two fewer solved instances. However, there are also domains where the coverage increases. One main reason for failure is a higher memory consumption leading to 83 instances that failed due to the memory limit. Another reason is a time overhead that leads to 13.4% fewer evaluations per second on the commonly solved instances. On the positive side, the heuristic is indeed better informed which translates to a reduction in the number of expanded nodes (cf. Figure 2).

Conclusion

We propose a clean and general LTL$_f$ framework for optimal planning that is easy to prove correct. It is based on a feasibility criterion for LTL$_f$ trajectory constraints and there are plenty of possibilities to derive such feasible constraints from established planning methods.

We presented a baseline heuristic from such constraints, based on the extraction of disjunctive action landmarks. This heuristic does not yet fully exploit the potential of the approach because it does not consider the temporal information of the constraints. We plan to change this in future work where we will to a greater extend exploit LTL$_f$ reasoning methods in the heuristic computation. We also will investigate the potential for strengthening other heuristic computations with the information from LTL$_f$ trajectory constraints, similar to what Wang, Baier, and McIlraith (2009) have done with landmark orderings and the FF heuristic. Another research direction will be the examination of further sources of information and of possible positive interactions of their combination.

Acknowledgments

This work was supported by the European Research Council as part of the project “State Space Exploration: Principles, Algorithms and Applications”.

References

Bacchus, F., and Kabanza, F. 1998. Planning for temporally extended goals. *Annals of Math. and AI* 22(1,1):5–27.

Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *AIJ* 116(1–2):123–191.

Bonet, B. 2013. An admissible heuristic for SAS$^+$ planning obtained from the state equation. In *Proc. IJCAI 2013*, 2268–2274.

Calvanese, D.; De Giacomo, G.; and Vardi, M. Y. 2002. Reasoning about actions and planning in LTL action theories. In *Proc. KR 2002*, 593–602.

Cerrito, S., and Mayer, M. C. 1998. Using linear temporal logic to model and solve planning problems. In Giunchiglia, F., ed., *Artificial Intelligence: Methodology, Systems, and Applications (AIMSA 98)*, volume 1480 of *LNCS*, 141–152. Springer-Verlag.

De Giacomo, G., and Vardi, M. Y. 2013. Linear temporal logic and linear dynamic logic on finite traces. In *Proc. IJCAI 2013*, 854–860.

De Giacomo, G.; De Masellis, R.; and Montali, M. 2014. Reasoning on LTL on finite traces: Insensitivity to infiniteness. In *Proc. AAAI 2014*, 1027–1033.

de la Rosa, T., and McIlraith, S. 2011. Learning domain control knowledge for TLPlan and beyond. In *ICAPS 2011 Workshop on Planning and Learning*, 36–43.

Doherty, P., and Kvarnström, J. 2001. TALplanner: A temporal logic based planner. *AI Magazine* 22(3):95–102.

Domshlak, C.; Helmert, M.; Karpas, E.; Keyder, E.; Richter, S.; Röger, G.; Seipp, J.; and Westphal, M. 2011. BJOLP: The big joint optimal landmarks planner. In *IPC 2011 planner abstracts*, 91–95.

Gerevini, A., and Long, D. 2005. Plan constraints and preferences in PDDL3. Technical Report R. T. 2005-08-47, Dipartimento di Elettronica per l'Automazione, Università degli Studi di Brescia.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. ICAPS 2009*, 162–169.

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *JAIR* 22:215–278.

Kabanza, F., and Thiébaux, S. 2005. Search control in planning for temporally extended goals. In *Proc. ICAPS 2005*, 130–139.

Karpas, E., and Domshlak, C. 2009. Cost-optimal planning with landmarks. In *Proc. IJCAI 2009*, 1728–1733.

Karpas, E., and Domshlak, C. 2011. Living on the edge: Safe search with unsafe heuristics. In *ICAPS 2011 Workshop on Heuristics for Domain-Independent Planning*, 53–58.

Karpas, E., and Domshlak, C. 2012. Optimal search with inadmissible heuristics. In *Proc. ICAPS 2012*, 92–100.

Keyder, E.; Richter, S.; and Helmert, M. 2010. Sound and complete landmarks for and/or graphs. In *Proc. ECAI 2010*, 335–340.

Koehler, J., and Treinen, R. 1995. Constraint deduction in an interval-based temporal logic. In Fisher, M., and Owens, R., eds., *Executable Modal and Temporal Logics*, volume 897 of *LNCS*, 103–117. Springer-Verlag.

Pnueli, A. 1977. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS 1977)*, 46–57.

Pommerening, F., and Helmert, M. 2013. Incremental LM-cut. In *Proc. ICAPS 2013*, 162–170.

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2014. LP-based heuristics for cost-optimal planning. In *Proc. ICAPS 2014*, 226–234.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR* 39:127–177.

van den Briel, M.; Benton, J.; Kambhampati, S.; and Vossen, T. 2007. An LP-based heuristic for optimal planning. In *Proc. CP 2007*, 651–665.

Wang, L.; Baier, J.; and McIlraith, S. 2009. Viewing landmarks as temporally extended goals. In *ICAPS 2009 Workshop on Heuristics for Domain-Independent Planning*, 49–56.

Zhu, L., and Givan, R. 2003. Landmark extraction via planning graph propagation. In *ICAPS 2003 Doctoral Consortium*, 156–160.