Compiling Away Soft Trajectory Constraints in Planning

Benedict Wright and **Robert Mattmüller** and **Bernhard Nebel**
University of Freiburg
{bwright, mattmuel,nebel}@informatik.uni-freiburg.de

Abstract

Soft goals in planning describe optional goals that should be achieved in the goal state. However, failing to achieve soft goals does not result in the plan becoming invalid. State trajectory constraints describe requirements towards the way the target goal is achieved, thus describing requirements towards the state trajectory of the final plan. Soft trajectory constraints express preferences on how the hard goals are reached, thus stating optional requirements towards the state trajectory of the plan. Such a soft trajectory constraint may require that some fact should be always true, or should be true at some point during the plan. The quality of a plan is then measured by a metric which adds the sum of all action costs and a penalty for each failed soft trajectory constraint. Keyder and Geffner showed that soft goals can be compiled away. We generalize this approach and illustrate a method of compiling soft trajectory constraints into conditional effects and state dependent action costs using LTL_f and Büchi automata. With this we are able to handle such soft trajectory constraints without the need of altering the search algorithm or heuristics, using classical planners.

Introduction

Soft goals in planning are additional requirements towards the resulting plan. These requirements differ from classical (hard) goals in that violating them does not render a plan invalid. PDDL 3.0 (Gerevini and Long 2005) introduced state trajectory constraints, which add constraints towards *how* goals are achieved. These come in two flavors, as hard constraints and as soft constraints. For the rest of the paper, we will refer to optional state trajectory constraints as soft trajectory constraints. We use the term "soft goals" to mean reachability soft goals and soft trajectory constraints alike. This is justified since reachability soft goals φ can be seen as a special case of soft trajectory constraints of the form (at end φ).

For checking satisfaction of reachability soft goals, it is sufficient to test if they hold in the final state. However, for soft trajectory constraints, a more sophisticated method of checking their satisfaction is required. For example, if a soft trajectory constraint requires a fact to be always true, it is not sufficient to check if the fact is true in the final state, but it needs to be tracked to check if the fact holds at any given step of the plan.

The introduction of soft goals changes the overall quality of a plan such that a cheapest plan achieving the hard goals is not necessarily an optimal plan, as it does not take into account the achieving or failing of soft goals. For this, a metric consisting of plan cost and a penalty for violated soft goals is introduced. Thus, an optimal plan would incorporate all soft goals while minimizing the total cost of the plan. This corresponds to a constraint optimization problem, where the constraints are the hard goals and the optimization tries to fulfill the soft goals.

One issue that arises when dealing with soft goals is the trade-off between minimizing cumulative action costs along the way to a state satisfying the hard goals, and maximizing rewards for achieved soft goals. An additional challenge is how to inform the search about which paths appear promising towards optimizing this trade-off. In this paper, we show how soft trajectory constraints can be compiled away using LTL_f, Büchi automata, conditional effects, and state dependent action costs, generalizing the soft goal compilation introduced by Keyder and Geffner (2009). This allows us to use off-the-shelf classical planning heuristics to provide the required guidance.

Related work

Baier and McIlraith (2008) give an overview over planning with preferences, where they use the term preference to state a preference of one plan over another, introducing different preference formalisms based on quantitative, and qualitative languages. Using quantitative languages, the quality of a given plan can be determined by a numeric value, such as the overall reward in Markov Decision Processes (MDP). In these MDPs the reward of an action can be used to specify preferences over actions. Alternatively, the quality of a plan can be determined over a set of properties, such as satisfied preferences. Such a system was implemented in PDDL3 (Gerevini and Long 2005) where preferences can be specified as temporal, or temporally extended predicates, using a subset of LTL.

Baier *et al.* (2009) describe a method of compiling problems with temporally extended preferences into simpler versions consisting only of preferences that hold in the final state, and can be evaluated using an objective function. The authors achieve this by translating the LTL expressed preferences into parameterized non-deterministic finite state au-

tomata (PNFA). They then track the state of each object within the automaton using a predicate for each automaton, which tracks the state of each object within the automaton. Here objects can reside in more than one state of the automaton at each time step. Additionally they introduce a predicate that holds if the automaton is in an accepting state for any given object. Instead of tracking the state of the automaton by extending the existing operators, they modify their search algorithm to automatically apply the automata's state transitions for each object. The quality of their approach can then be measured using an updated objective function.

Keyder and Geffner (2009) show that soft goals can be compiled away by introducing a new hard goal p, which can be achieved in two ways: A $collect(p)$ action which has zero cost but requires the soft goal to be achieved, and a $forgo(p)$ action that has costs equal to the utility of p but can be executed when the soft goal was not achieved. These actions can only be executed after the original plan goal was reached. However their approach does not take trajectory constraints into account, focusing on reachability soft goals only. We build upon this work to generalize their approach towards soft trajectory constraints.

Preliminaries

Linear-time temporal logic on finite traces

Linear-Time Temporal Logic (LTL) is a modal logic capable of expressing logic expressions referring to time. As we will see later in this section, LTL can be used to express trajectory constraints. Let $\mathcal{V}$ be a set of finite-domain state variables with associated finite domains $\mathcal{D}_v$. We call pairs (v, d) with $v \in \mathcal{V}$ and $d \in \mathcal{D}_v$ *facts*, and we denote the set of all facts by F. Then an LTL formula φ over $\mathcal{V}$ is either an atomic fact (v, d) over $\mathcal{V}$, or of the form $\neg\varphi$, $\varphi \vee \psi$, $\bigcirc\varphi$ ("next φ"), or $\varphi\mathcal{U}\psi$ ("φ until ψ"), where φ, ψ are LTL formulas. Other propositional connectives can be defined as abbreviations in the usual way, such as conjunction ($\wedge$), implication ($\rightarrow$), bi-implication ($\leftrightarrow$), truth ($\top$), and falsity ($\bot$). Similarly, $\Diamond\varphi$ ("finally φ") can be defined as an abbreviation for $\top\mathcal{U}\varphi$, and $\Box\varphi$ ("globally φ") as an abbreviation for $\neg\Diamond\neg\varphi$. We also introduce weak until $\varphi\mathcal{W}\psi$ as an abbreviation for $\varphi\mathcal{U}\psi \vee \Box\varphi$. Then the semantics of $\mathsf{LTL_f}$ (LTL on finite traces) is defined as the interpretation over finite traces denoting a sequence of instants of time. Let $\mu = (\mu(0), \mu(1), \ldots, \mu(n))$ be such a trace with $\mu(i) \subseteq F$ for all $i = 0, \ldots, n$. Then the truth of a formula φ along trace μ is defined as follows (De Giacomo and Vardi 2013):

$$
\begin{array}{lll}
\mu, i \models a & \textit{iff} & a \in \mu(i) \text{ for } a \in \mathcal{V} \\
\mu, i \models \neg\varphi & \textit{iff} & \mu, i \not\models \varphi \\
\mu, i \models \varphi_1 \wedge \varphi_2 & \textit{iff} & \mu, i \models \varphi_1 \textit{ and } \mu, i \models \varphi_2 \\
\mu, i \models \bigcirc\varphi & \textit{iff} & i < n \textit{ and } \mu, i+1 \models \varphi \\
\mu, i \models \varphi_1 \mathcal{U} \varphi_2 & \textit{iff} & \exists j, i \leq j \leq n : \mu, j \models \varphi_2 \textit{ and} \\
 & & \quad \forall k, i \leq k \leq j : \ \mu, k \models \varphi_1 \\
\mu \models \varphi & \textit{iff} & \mu, 0 \models \varphi
\end{array}
$$

Trajectory constraints as LTL

PDDL 3.0 (Gerevini and Long 2005) introduced state-trajectory constraints, which are modal logic expressions that ought to be true for the state trajectory produced during the execution of the plan. As shown by De Giacomo *et al.* (2014), these can be expressed using LTL:

$$
\begin{aligned}
\text{(at end } \varphi) &:= \Diamond(\mathit{last} \wedge \varphi) \\
\text{(always } \varphi) &:= \Box\varphi \\
\text{(sometime } \varphi) &:= \Diamond\varphi \\
\text{(within } n\ \varphi) &:= \bigvee_{0 \leq i \leq n} \underbrace{\bigcirc \ldots \bigcirc}_{i} \varphi \\
\text{(hold-after } n\ \varphi) &:= \underbrace{\bigcirc \ldots \bigcirc}_{n} \Diamond\varphi \\
\text{(hold-during } n_1\ n_2\ \varphi) &:= \underbrace{\bigcirc \ldots \bigcirc}_{n_1} \Big(\bigwedge_{0 \leq i \leq n_2} \underbrace{\bigcirc \ldots \bigcirc}_{i} \varphi \Big) \\
\text{(at-most-once } \varphi) &:= \Box(\varphi \rightarrow \varphi\mathcal{W}\neg\varphi) \\
\text{(sometime-after } \varphi\ \psi) &:= \Box(\varphi \rightarrow \Diamond\psi) \\
\text{(sometime-before } \varphi\ \psi) &:= (\neg\varphi \wedge \neg\psi)\mathcal{W}(\neg\varphi \wedge \psi) \\
\text{(sometime-within } n\ \varphi\ \psi) &:= \Box\Big(\varphi \rightarrow \bigvee_{0 \leq i \leq n} \underbrace{\bigcirc \ldots \bigcirc}_{i} \psi\Big)
\end{aligned}
$$

Here, φ and ψ are propositional formulas on fluents, and n, n_1, n_2 natural numbers. The predicate last is introduced during the translation from $\mathsf{LTL_f}$ to LTL, and is true if an only if $\neg \bigcirc \top$ which is the case in the last state of the state trajectory. As the plan resulting from our planning task is always finite, we need this restriction on LTL.

Planning tasks

Since we want to compile away soft trajectory constraints using conditional effects and state-dependent action costs, we base our exposition on a formalization of planning tasks that admits all of those features. This leads us to the following definition:

A *planning task* is a tuple $\Pi = \langle \mathcal{V}, A, s_0, s_\star, \Phi \rangle$ consisting of the following components: $\mathcal{V} = \{v_1, \ldots, v_n\}$ is a finite set of *state variables*, each with an associated finite domain $\mathcal{D}_v$. A *fact* is a pair (v, d), where $v \in \mathcal{V}$ and $d \in \mathcal{D}_v$, and a *partial variable assignment* s over $\mathcal{V}$ is a consistent set of facts. If s assigns a value to each $v \in \mathcal{V}$, s is called a *state*. Let S denote the set of states of Π. A is a set of *actions*, and each action is a pair $a = \langle \textit{pre}, \textit{eff} \rangle$, where *pre* is a partial variable assignment called the *precondition*, and where *eff* is an *effect* of the form $\textit{eff} = \bigwedge_{i=1,\ldots,n} (\textit{pre}_i \rhd \textit{eff}_i)$ for some number $n \in \mathbb{N}$ of *conditional effects*, each consisting of an effect condition $\textit{pre}_i$, again a partial variable assignment, and an effect $\textit{eff}_i$, also a partial variable assignment. The state $s_0 \in S$ is called the *initial state*, and the partial state $s_\star$ specifies the *goal* condition. Each action $a \in A$ has an associated cost function $c_a : S \rightarrow \mathbb{N}$ that assigns the cost of a to each state where a is applicable. Finally, Φ is a finite set of $\mathsf{LTL_f}$ formulas over $\mathcal{V}$, the *soft trajectory constraints*. Each soft trajectory constraint $\varphi \in \Phi$ has an associated *weight* $w_\varphi \in \mathbb{N}$ specifying which importance we

assign to satisfying φ. For states s, we use function notation $s(v) = d$ and set notation $(v, d) \in s$ interchangeably. For facts we also use sets and conjunctive logical expressions interchangeably, where a set of facts is treated equivalently to a conjunction of these facts.

The change set $[\mathit{eff}]_s$ of effect $\mathit{eff} = \bigwedge_{i=1,\ldots,n}(\mathit{pre}_i \rhd \mathit{eff}_i)$ in state s is the set of facts that *eff* makes true if applied in s, i. e., the set $\bigcup_{i=1,\ldots,n}[\mathit{pre}_i \rhd \mathit{eff}_i]_s$, where $[\mathit{pre}_i \rhd \mathit{eff}_i]_s$ is either $\emptyset$, if $s \not\models \mathit{pre}_i$, or eff_i, if $s \models \mathit{pre}_i$. Then an action $a = \langle \mathit{pre}, \mathit{eff} \rangle$ is applicable in state s iff $\mathit{pre} \subseteq s$ and the change set $[\mathit{eff}]_s$ is consistent. Applying action a to s yields the state s' with $s'(v) = [\mathit{eff}]_s(v)$ where $[\mathit{eff}]_s(v)$ is defined, and $s'(v) = s(v)$ otherwise. We write $s[a]$ for s'. A state s is a goal state iff $s_\star \subseteq s$. We denote the set of goal states by $S_\star$. Let $\pi = (a_0, \ldots, a_{n-1})$ be a sequence of actions from A. We call π *applicable* in s_0 if there exist states $s_1, \ldots, s_n$ such that a_i is applicable in s_i and $s_{i+1} = s_i[a_i]$ for all $i = 0, \ldots, n-1$. In that case, we call $\mu_\pi = (s_0, s_1, \ldots, s_n)$ the *state trajectory induced by* π *in* s_0. We call π a *plan* for Π if it is applicable in s_0 and if $s_n \in S_\star$. The *action cost* of plan π is the sum of action costs along the induced state sequence, i.e., $\mathit{cost}(\pi) = \sum_{i=0}^{n-1} c_{a_i}(s_i)$. A plan π is penalized with penalty w_φ for each soft trajectory constraint $\varphi \in \Phi$ that is violated on its induced trajectory. Formally, the value $\mathit{penalty}(\pi, \varphi)$ for π with respect to φ is 0, if $\mu_\pi \models \varphi$, and w_φ, if $\mu_\pi \not\models \varphi$. The *overall penalty* for π is $\mathit{penalty}(\pi) = \sum_{\varphi \in \Phi} \mathit{penalty}(\pi, \varphi)$.

The *total cost* of plan π is its action costs plus its overall penalty, i. e., $\mathit{totalcost}(\pi) = \mathit{cost}(\pi) + \mathit{penalty}(\pi)$. A plan is *optimal* for Π if it minimizes $\mathit{totalcost}$ among all plans for Π.

Automata semantics of planning tasks

A deterministic finite automaton (DFA) is a tuple $\mathcal{A} = \langle \Sigma, Q, \Delta, q_0, Q_a \rangle$ consisting of an alphabet Σ, a set of states Q, a transition function $\Delta : Q \times \Sigma \to Q$, an initial state $q_0 \in Q$, and a set of accepting states $Q_a \subseteq Q$. The transition system of any planning task $\Pi = \langle \mathcal{V}, A, s_0, s_\star, \Phi \rangle$ can be understood as a DFA $\mathcal{A}(\Pi)$ as follows: the input alphabet is $\Sigma = A \times 2^F$. The set of states, the initial state, and the set of accepting/goal states of $\mathcal{A}(\Pi)$ are those of Π, i. e., $Q = S$, $q_0 = s_0$, and $Q_a = S_\star$. Finally, Δ consists of all transitions of the form $\langle s, (a, t), t \rangle$ where $a \in A$ is applicable in s and $s[a] = t$. What was lost in the translation from Π to $\mathcal{A}(\Pi)$ are the action costs and the soft trajectory constraints. Costs are trivial to handle by adding weights to the automaton, and we will come back to that later. To give an automata-based semantics to state-trajectory constraints, we need to review the theory of Büchi automata first.

Büchi automata

A deterministic Büchi automaton (Büchi 1962) $\mathcal{B} = (\Sigma, Q, \Delta, q_0, Q_a)$ consists of the same components as a DFA, and differs from a DFA only in the acceptance condition. Whereas a DFA $\mathcal{A}$ accepts a finite input word μ if after reading μ, $\mathcal{A}$ is in an accepting state, a Büchi automaton $\mathcal{B}$ accepts an infinite word μ if, while reading μ, $\mathcal{B}$ visits an accepting state infinitely often. For every LTL formula φ, there is a deterministic Büchi automaton $\mathcal{B}(\varphi)$ that accepts exactly those infinite words μ with $\mu \models \varphi$. In the case of finite traces (finite words) required by $\mathsf{LTL_f}$, the same automaton accepts the word if at the end of the word the automaton is in an accepting state (Giannakopoulou and Havelund 2001). There are multiple algorithms for constructing a Büchi automaton that accepts exactly those words that satisfy a given LTL formula (Gerth *et al.* 1996; Gastin and Oddoux 2001). Constructing an automaton from a given $\mathsf{LTL_f}$ formula φ can be achieved by first translating φ into a LTL formula as described in De Giacomo *et al.* (2014) and then applying a given construction algorithm. Simply put, this translation adds a new predicate last which is only true in the last instance of the interpretation sequence, and therefore ensuring finite traces.

Now, for a planning task Π with a *hard* state-trajectory constraint φ, the standard automaton construction considers the product automaton $\mathcal{C}$ of $\mathcal{A}(\Pi)$ and $\mathcal{B}(\varphi)$. Then, a state trajectory μ is a solution to Π satisfying φ iff μ is accepted by $\mathcal{C}$. For *soft* state-trajectory constraints, we can still perform the same product automaton construction to track which soft constraints are satisfied by a plan. Unlike with hard constraints, however, the product automaton still has to accept trajectories that violate soft constraints, and the violation has to be reflected in the plan costs, rather than in the acceptance condition of the product automaton. The next section describes the product construction, an assignment of action costs that reflects the satisfaction or violation of soft trajectory constraints, and a compact encoding of the product automaton as a new planning task Π'.

Goal action penalty compilation

Let $\Pi = \langle \mathcal{V}, A, s_0, s_\star, \Phi \rangle$ be the original planning task with soft trajectory constraints Φ and with objective function $\mathit{totalcost}$ as defined above. Transition costs aside, the semantics of Π are captured by the product automaton $\mathcal{C} = \mathcal{A}(\Pi) \times \prod_{\varphi \in \Phi} \mathcal{B}(\varphi)$. However, when compiling away soft trajectory constraints, we do not want to generate an *automaton*, but rather another *planning task* Π' such that $\mathcal{A}(\Pi')$ is isomorphic to $\mathcal{C}$. We now describe this construction. For simplicity of exposition, we assume that Φ consists of a single constraint φ only. Generalization to more than one soft trajectory constraint is straightforward.

The idea behind the construction of Π' is to add a new tracking variable τ_φ to Π that keeps track of the current state of $\mathcal{B}(\varphi)$. The actions in Π' are those from Π, augmented with conditional effects that take care of the correct evolution of the value of τ_φ, thus encoding the soft trajectory constraints into the actions. Action costs stay the same. Finally, a terminal action $\mathit{last\_op}$ is added to Π' that marks termination. Only after termination has been marked, we may start evaluating the penalty term for unsatisfied soft goals.

Formally, let $\mathcal{B}(\varphi) = (Q, \Sigma, \Delta, q_0, Q_a)$ be the deterministic Büchi automaton that accepts φ. Then we create planning task $\Pi' = \langle \mathcal{V}', A', s'_0, s'_\star, \emptyset \rangle$ with $\mathcal{V}' = \mathcal{V} \cup \{\mathit{last}\} \cup \{\tau_\varphi\}$, with a propositional domain for last and domain Q for τ_φ. The initial state s'_0 agrees with s_0 on all variables in $\mathcal{V}$, and additionally, $s'_0(\mathit{last}) = \bot$ and $s'_0(\tau_\varphi) = q_0$.

The actions are $A' = \{o' \mid o \in A\} \cup \{last\_op\}$ where $o' = \langle pre', eff' \rangle$ is constructed from $o = \langle pre, eff \rangle$ as follows: $pre' = pre$ and

$$eff' = eff \wedge \bigwedge_{\substack{\langle q,s,q' \rangle \in \Delta \text{ with} \\ last \notin s}} ((\tau_\varphi = q \wedge P) \rhd \tau_\varphi := q'),$$

where $P = q' \setminus eff$. In words, we add conditional effects to track the value of τ_φ for each transition in $\mathcal{B}(\varphi)$. The facts in s are either already true in q as ensured by the effect conditions P or are set to true by the original actions effect *eff*. As an exception, if s contains the keyword $last$, which can only be true in the last step of the plan, we add the new action $last\_op = \langle s_\star, last := \top \rangle$ instead. To ensure that this action has to be executed as the last step of any plan for Π', we replace the original goal condition $s_\star$ by the new goal condition $s'_\star = last$. As action costs, we have $c_{o'} = c_o$ for all $o \in A$, and $c_{last\_op} = 0$. Additional formal machinery needed for the evaluation of the penalty term is deferred until after the following proposition.

Proposition 1. *Up to transitions with action last_op, the transition system $\mathcal{A}(\Pi')$ is isomorphic to the product of the transition system $\mathcal{A}(\Pi)$ and trajectory constraints LTL_f automaton $\mathcal{B}(\varphi)$.*

Proof. For this proof we slightly alter $\mathcal{B}(\varphi)$ such that each transition not only consists of a partial variable assignment, but a tuple of a state and an action. We replace each transition $\langle q, s, q' \rangle$ in $\mathcal{B}(\varphi)$ by a set of new transitions $\langle q, (o, t), q' \rangle$ for all $(t, o) \in S \times A$ such that $s \subseteq t$, where o is an action that after applying to q results in q'. Doing this creates an automaton with the same signature as $\mathcal{A}$. From this altered Büchi automaton we can now easily construct $\mathcal{A}(\Pi) \times \mathcal{B}(\varphi)$, by simply creating the Cartesian product of the two automata (Baier and Katoen 2008). A transition $\langle s'^q, (o', t'^{q'}), t'^{q'} \rangle$ is contained in $\mathcal{A}(\Pi')$ if and only if o is applicable in $s \in \Pi$ and $t = apply(o, s)$ and $(\tau_\varphi, q) \in s'^q$ and $(\tau_\varphi, q') \in t'^{q'}$. Then $\langle s, (o, t), t \rangle \in \mathcal{A}(\Pi)$ and $\langle q, (o, t), q' \rangle \in \mathcal{B}(\varphi)$ if and only if $\langle (s, q), (o, t), (t, q') \rangle \in \mathcal{A}(\Pi) \times \mathcal{B}(\varphi)$, thus $\mathcal{A}(\Pi')$ is isomorphic to $\mathcal{A}(\Pi) \times \mathcal{B}(\varphi)$. □

Now that we can track the state of each soft trajectory constraint within the planning task Π', we need to add penalties for all constraints not achieved in the reached terminal state. For this we add another propositional variable in_goal to Π' that is initially false, and change the goal $s'_\star$ from $last$ to in_goal. This means that every plan for Π' has to include an occurrence of the new action $penalize = \langle last \wedge \neg in\_goal, in\_goal \rangle$ as its last step. The cost function of the action $penalize$ now simply determines the penalty value $penalty(\pi)$ based on which soft trajectory constraints $\varphi \in \Phi$ are violated by testing whether the corresponding tracking variables τ_φ encode accepting or non-accepting Büchi automata states in the current planning state. More formally, $c_{penalize} = \sum_{\varphi \in \Phi} [\tau_\varphi \notin Q_a^\varphi] w_\varphi$ where $[\tau_\varphi \notin Q_a^\varphi] = 1$ if $\tau_\varphi = q$ and $q \notin Q_a^\varphi$ for some $q \in Q^\varphi$, and 0 otherwise.

Notice that the action $penalize$ has *state-dependent costs* that are not universally supported by planning systems. However, those can be compiled away to state-independent costs, if this is desired (Geißer *et al.* 2015). Notice further that determining the value $[\tau_\varphi \notin Q_a^\varphi]$ is also simple. It can either be rewritten as $\sum_{q \in Q^\varphi \setminus Q_a^\varphi} [\tau_\varphi = q]$, where $[\tau_\varphi = q]$ is 1 if $s(\tau_\varphi) = q$, and 0 otherwise; alternatively, another new propositional variable $is\_violated_\varphi$ can be added to the planning task that is true iff the value of τ_φ represents a non-accepting state. Then $c_{penalize} = \sum_{\varphi \in \Phi} [is\_violated_\varphi] w_\varphi$. A natural modeling would treat $is\_violated_\varphi$ as a derived variable, and would have axioms that express $is\_violated_\varphi$ in terms of τ_φ. We mention this latter possibility since it makes the relation between our proposed compilation and that of Keyder and Geffner (2009) obvious (cf. Remark 1 below).

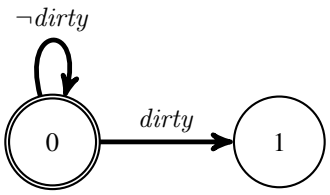


Figure 1: Büchi automaton for $\Box\neg dirty$

In any case, it is clear that adding this action preserves the original objective function.

Proposition 2. *Let Π' be the compiled task from Π. Then an optimal plan for Π' is also an optimal plan for Π (without the penalize action).*

Proof. From Proposition 1 we get that the compilation is sound and complete. The objective function of the original task is $penalty(\pi) + cost(\pi)$. Up until the $penalize$ action, the objective function sums up all action costs, as the cost functions for each action are not altered by the compilation. The $penalize$ action then adds a penalty for each soft trajectory constraint that is not satisfied, resulting in an objective function identical to the original objective function. □

Example 1. *Let $a_1 = \langle \top, dirty \rangle$ be an action and φ the preference $\Box\neg dirty$. We can then track the state in the automaton in Figure 1 by adding the conditional effect $(\tau_\varphi = 0 \rhd \tau_\varphi := 1)$ to action a_1, as can be derived from Figure 2. Let a_2 be another action that does not have $dirty$ among its effects. Then we need to add the conditional effect $(\tau_\varphi = 0 \wedge dirty \rhd \tau_\varphi := 1)$ to a_2 that transitions from state 0 to state 1 if $dirty$ is true regardless of the effect of a_2.[1] The partial cost function c for this preference is $c = [\tau_\varphi = 1] w_\varphi$ and is added to the cost of the $penalize$ action. This adds w_φ to the total plan cost if $\mathcal{B}(\varphi)$ is in the non-accepting state 1.*

An analysis of the $penalize$ action shows that after applying the EVMDD compilation (Geißer *et al.* 2015), the resulting operations correspond to the operations $collect$, $forgo$,

[1] It is an invariant of this planning task that, whenever $dirty$ is true, τ_φ is 1. Therefore, the effect condition $\tau_\varphi = 0 \wedge dirty$ can never be satisfied. However, detecting this, and then removing conditional effects whose condition is inconsistent with an invariant, and thus simplifying the constructed conditional effects, is beyond the scope of this work.

$$\tau_\varphi := \begin{cases} 1: & \tau_\varphi = 0 \wedge (\mathit{dirty} \in [\mathit{eff}]_s \vee \\ & \qquad (s \models \mathit{dirty} \wedge \neg \mathit{dirty} \notin [\mathit{eff}]_s)) \\ 1: & \tau_\varphi = 1 \\ 0: & \tau_\varphi = 0 \wedge (\neg \mathit{dirty} \in [\mathit{eff}]_s \vee \\ & \qquad (s \models \neg \mathit{dirty} \wedge \mathit{dirty} \notin [\mathit{eff}]_s)) \end{cases}$$

Figure 2: Derivation of conditional effects for τ_φ from Figure 1, where s is the current state of the search

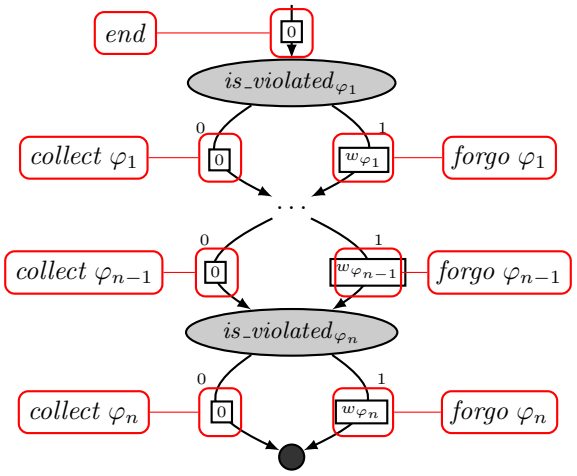


Figure 3: EVMDD compilation of *penalize* action with derived variables $\mathit{is\_violated}_{\varphi_i}$, which are true if τ_{φ_i} is in a non-accepting state. Numbers on edges are partial costs (= costs of compiled actions).

and *end* described in the compilation by Keyder and Geffner (2009). This immediately implies that our approach generalizes the soft trajectory constraint compilation by Keyder and Geffner (2009) to support trajectory constraints.

Remark 1. *We've seen that we can express the cost of the penalize action as* $c_{penalize} = \sum_{\varphi \in \Phi} [\mathit{is\_violated}_\varphi] w_\varphi$. *Expressed as an edge-valued multi-valued decision diagram (EVMDD) (Geißer* et al. *2015),* $c_{penalize}$ *looks as depicted in Figure 3 (without the red annotations). The EVMDD-based action compilation of Geißer* et al. *(2015) now turns each edge of the EVMDD into a new auxiliary action. These new actions are exactly the end, collect, and forgo actions from Keyder and Geffner (2009) (indicated as the red annotations).*

One limitation of this approach is that the achievement of any soft trajectory constraint is only represented by the h-value (up until the *penalize* action). A more desirable compilation would provide the search with a more accurate g-value, thus informing the search when a soft trajectory constraint is achieved. In the following section we will demonstrate a possible solution to this problem.

General action penalty compilation

In this section we will show how the above approach can be extended to provide the search with a more accurate g-value. The main reason for the uninformedness in relation to the g-value is the fact that any penalty is only applied in the very last step of the search in the *penalize* action. However, while tracking the soft trajectory constraint's automaton $\mathcal{B}(\varphi)$, we already have information about the current acceptance status of each soft trajectory constraint. We will now show how this information can be used to add penalties and rewards to the individual actions changing the state of $\mathcal{B}(\varphi)$.

Whenever an action a changes the value of τ_φ, thus transitioning from one state q to another state q' in $\mathcal{B}(\varphi)$, we add a penalty or a reward depending on the type of transition. When q is an accepting state and q' a non-accepting state in $\mathcal{B}(\varphi)$, we add a penalty to the action cost. If, on the other hand, q' is an accepting state and q is a non-accepting state, we can add a reward. The partial cost function for transitions in $\mathcal{B}(\varphi)$ then takes the form $\sum_{q \in Q^\varphi} [\tau_\varphi = q] \omega_\varphi(q, q')$, where $\omega_\varphi(q, q')$ is a penalty or reward term and is added to the cost function c of a. For transitions from accepting to non-accepting states, we set $\omega_\varphi(q, q')$ to a positive penalty term and for transitions from non-accepting to accepting states, we set $\omega_\varphi(q, q')$ to a reward in the form of a negative value. Similarly, the partial cost functions for each type of transition can be formulated, setting $\omega_\varphi(q, q')$ accordingly. For the actual value of $\omega_\varphi(q, q')$, we use the value from the original soft trajectory's weight w_φ. The total cost function of each action is then the sum of the partial cost functions plus the original action cost.

This way, we penalize actions resulting in a transition from accepting to non-accepting states by giving them higher costs, and reward actions that result in an accepting state of $\mathcal{B}(\varphi)$ by applying negative costs. Note, that $\omega_\varphi(q, q')$ only accounts once in the total cost, as we can never add $\omega_\varphi(q, q')$ without subtracting it beforehand.

By construction, minimizing *totalcost* in the compiled task Π' amounts to the same as minimizing the *totalcost* of the original task Π. One minor detail to take in to account is if the initial state of $\mathcal{B}(\varphi)$ is in a non-accepting state, we need to add a penalty to account for this. We do this by adding an additional penalty to the *penalize* action.

The problem now is that we have introduced negative action costs. As we can ensure that we do not have any negative cycles in our search, resulting in a total plan cost ≥ 0, we can use planners that support negative action costs. Note that having such negative-cost cycles would result in arbitrarily low *totalcost*, and the non-termination of the search, as each node in the cycle can be reached by a yet cheaper path. Currently, Fast Downward (Helmert 2006) with blind heuristic supports negative action costs. However, for more sophisticated heuristics, or planners not supporting negative action costs, negative action costs need to be removed.

To remove negative action costs, we introduce a state transition cost (Table 1), where we specify the penalty/reward for each possible transition type. This transition cost table gives us greater control over the implications a state transition in $\mathcal{B}(\varphi)$ has towards fulfilling the soft trajectory con-

Table 1: State Transition Costs

(a) Metric Preserving Costs

From \ To	Accepting	¬ Accepting
Accepting	0	w_φ
¬ Accepting	$-w_\varphi$	0

(b) Positively Shifted Costs

From \ To	Accepting	¬ Accepting
Accepting	w_φ	$2w_\varphi$
¬ Accepting	0	w_φ

(c) Adapted Positively Shifted Costs

From \ To	Accepting	¬ Accepting
Accepting	0	$2w_\varphi$
¬ Accepting	0	w_φ

straint. For instance, by setting the penalty/reward $\omega_\varphi(q, q')$ of a transition from an accepting state to another (or the same) accepting state to $\omega_\varphi(q, q') = 0$ and all other transitions to $\omega_\varphi(q, q') > 0$, we can model the preference of staying in an accepting state over all other possibilities. Additionally, we can set the cost for leaving an accepting and entering a non-accepting state higher as to penalize these actions.

The transition cost table (Table 1a) corresponds to the cost function described above. Table 1b shows the cost function where the costs have been shifted by w_φ to remove negative costs. As one can see this has the negative effect of penalizing state transitions from accepting to accepting states. Therefore, we introduce transition Table 1c, where transitions from accepting to accepting sates are also not penalized. Transitions leaving an accepting state, however, are highly penalized, whereas remaining in a non-accepting state is only penalized by a lower cost.

This cost function is informative regarding h and g values, regardless of the actually used cost table, however the total cost of the compiled task is greater than the original plans total cost $\mathit{totalcost}(\pi') \geq \mathit{totalcost}(\pi)$, where π, π' are plans from Π and Π' respectively. This is due to the fact that penalties from staying in a non-accepting state are added multiple times.

Experiments

We implemented our compilation into a recent version of the Fast Downward planning system supporting state dependent action costs. The evaluation was executed on a a subset of the fifth International Planning Competition (IPC-5) plus the Rovers domain from the IPC-3. We will now first discuss the results for the goal action penalty compilation, followed by the general action penalty compilation, finalizing with a discussion and comparison of the two approaches. In the domain names, SP and QP stand for Simple Preferences and Qualitative Preferences, respectively. The difference in these being that simple preferences use goal state preferences of the form (at end φ) only, and qualitative preferences use more complex state trajectory constraints. As the competition was for satisficing planning only, and many instances were too hard for optimal planning, which we are interested in, we generated additional simpler instances by randomly sampling subsets of the soft trajectory constraints. From each instance, we generated six new instances with 1%, 5%, 10%, 20%, 40%, and 100% of the soft trajectory constraints. We did not alter the hard goals of the original instances, which led to the exclusion of the openstacks domain, as finding optimal solutions for more than the very simple instances proved to be too hard.

Goal action penalty compilation results

For the goal action penalty compilation, we used h^{blind}, $h^{\max}$, and $h^{\text{M\&S}}$ for the optimal track. For the satisficing benchmark, we used h^{add} and h^{FF} with iterative eager greedy search with three iterations. No significant differences where found between the two heuristics in the satisficing benchmark, with a slightly better performance by h^{FF}. In the remaining evaluation, we therefore only consider h^{FF}.

As can be seen in Table 2, the performances varied over the domains. This is a consequence of finding an optimal solution to the hard goals even without considering the soft trajectory constraints. The trucks domain did not execute on the merge and shrink heuristic, as this heuristic does not support axioms, which are introduced by the translate step in the Fast Downward planner.

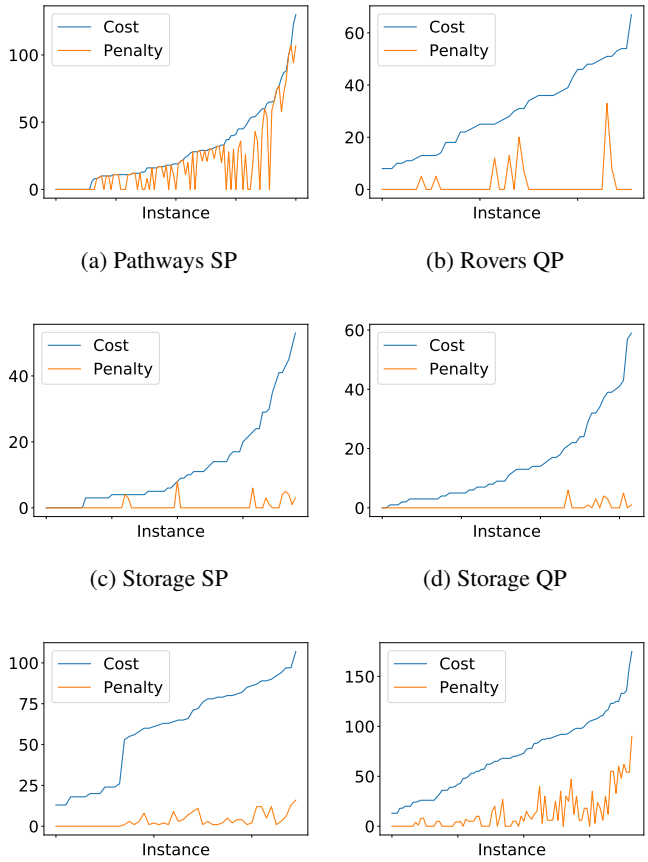
As can be seen in Figure 4, the satisficing benchmark performed rather well on the Rovers, Storage, and Trucks SP domain, as their penalty is always close to zero. The quality of the Trucks QP domain is slightly worse as fulfilling all soft trajectory constraints becomes more difficult, the more complex the instance is. For the pathways domain, we increased the penalty for not achieving soft goals by a factor of 10, as otherwise the optimal plan would be to ignore the soft trajectory constraints. As this domain has no hard goals, but soft trajectory constraints only, this would have resulted in an empty plan. As can be seen in some cases this was not sufficient and the resulting penalty is equal to the total cost, indicating that no soft trajectory constraints where satisfied. The storage domain also has no hard goals, but the penalties where already high in comparison to the action costs, requiring no alteration of the penalties.

General action penalty compilation results

Here we compare the results using the different configurations from Table 1. The experimental setup is identical to the above with the slight exception to configuration from Table 1a where only h^{blind} was used, as it requires negative action costs. As can be seen in Table 3a, the increased informedness of the general action compilation together with the metric preserving cost function did not significantly increase the amount of optimally solved instances. This is a result of the relative uninformedness of the blind heuristic, and the fact that the cost function needs to be evaluated for

Domain	h^{blind}	$h^{\max}$	$h^{\text{M\&S}}$
pathways SP	12.22%	18.33%	12.22%
rovers QP	18.33%	14.17%	16.67%
storage SP	33.33%	39.17%	32.50%
storage QP	25.49%	32.35%	24.51%
trucks SP	23.53%	15.29%	na
trucks QP	19.83%	14.66%	na

Table 2: Coverage of goal action penalty compilation of the IPC-5 benchmark set with additional instances with randomly sampled soft trajectory constraints, A* search for optimal solution.



(a) Pathways SP (b) Rovers QP

(c) Storage SP (d) Storage QP

(e) Trucks SP (f) Trucks QP

Figure 4: Plan quality of the satisficing benchmarks, ordered by *total cost* using goal action penalty compilation and h^{FF} heuristic

each action. As we currently use a relative unoptimized internal representation of the cost function, this significantly increases the search time, leading to timeouts before a solution could be found.

As can be seen in Tables 3b and 3c, the coverage increased significantly on these two cost compilations. This, however, is an artifact of the introduced error, as all actions become more expensive to execute. This results in the penalty for not achieving the soft trajectory constraints to become relatively low compared to the action costs. Thus, the empty plan becomes the optimal plan where no hard goals are specified, and the shortest plan becomes the optimal plan where hard goals are specified. This could be improved by a scaling function, which increases the penalty for not achieving the soft trajectory constraints and/or decreases the action costs.

Domain	h^{blind}	$h^{\max}$	$h^{\text{M\&S}}$
pathways SP	18.00%	na	na
rovers QP	12.00%	na	na
storage SP	12.00%	na	na
storage QP	8.16%	na	na
trucks SP	16.15%	na	na
trucks QP	26.45%	na	na

(a) Metric Preserving Costs

Domain	h^{blind}	$h^{\max}$	$h^{\text{M\&S}}$
pathways SP	36.67%	77.22%	7.22%
rovers QP	19.17%	11.67%	8.33%
storage SP	78.33%	78.33%	8.33%
storage QP	77.45%	76.47%	4.90%
trucks SP	23.53%	12.97%	na
trucks QP	20.54%	8.93%	na

(b) Positively Shifted Costs

Domain	h^{blind}	$h^{\max}$	$h^{\text{M\&S}}$
pathways SP	36.67%	77.22%	7.78%
rovers QP	16.67%	15.00%	10.00%
storage SP	78.33%	78.33%	8.33%
storage QP	77.45%	77.45%	4.90%
trucks SP	23.53%	12.94%	na
trucks QP	20.54%	12.50%	na

(c) Adapted Positively Shifted Costs

Table 3: Coverage of general action penalty compilation with the configurations from Table 1

Comparison to zero penalty compilation

Finally, we executed the same test set without a penalty action cost on goal action penalty compilation with blind heuristics for optimal solutions, and compared it to the above results regarding the average fulfilled soft trajectory con-

Domain	penalty	no penalty
pathways SP	97.19%	46.10%
rovers QP	47.05%	20.20%
storage SP	99.50%	54.20%
storage QP	99.90%	48.40%
trucks SP	98.10%	75.20%
trucks QP	100.00%	100.00%

Table 4: Comparison of average fulfilled soft trajectory constraints with and without penalty cost, only regarding instances for which a solution was found

straints, as shown in Table 4. Here, no penalty corresponds to the accidental fulfillment of the soft trajectory constraint, as the search is not guided towards them. As can be seen, the percentage of fulfilled soft trajectory constraints is significantly higher with cost guidance. The trucks domain does not show significant difference. This is a result of the overall hardness of finding an optimal solution as can be seen in Figure 2, as instances for which a solution was found were also easy to optimize towards their soft goals, whereas harder instances where not solved at all. Harder instances where not solved and thus not accounted for in Table 4.

Conclusion

In this paper, we introduced a method of compiling soft trajectory constraints into actions with conditional effects and state dependent action costs. For this, we created Büchi automata for each grounded soft trajectory constraint and modified the original planning task to track the state of each automaton during the state trajectory of the current partial plan. We then used state-dependent action costs to inform the heuristic guiding the search towards an optimal solution considering the soft trajectory constraints. We then conducted experiments using the IPC-5 benchmark set with additional generated instances. We showed that this approach enables classical planners to search for optimal solutions, taking soft trajectory constraints into account, without altering the search algorithm or implementing special heuristics.

Future work

One issue we found was that some soft trajectory constraints are simply not reachable or contradict hard goals. Therefore, these soft trajectory constraints can be removed from the search completely, and the penalty can be added directly in the $penalize$ action. We expect this to improve the overall performance of our approach, as the effort needed to track the states and calculate the costs is reduced.

Additionally, the cost function and automata tracking can be simplified by applying optimizations on the generation of the Büchi automata.

In the action penalty compilation, we introduced negative action costs. In our setting, using the Fast Downward planner (Helmert 2006), we were only able to use the blind heuristic, as it does not fail on negative action costs. An analysis of alternative heuristics concerning negative action costs could significantly improve the performance of our approach.

Going beyond what is already supported by PDDL 3, conditional preference networks (CP-nets) can express relations between preferences (Baier and McIlraith 2008). We would like to extend this notion to express relations between soft goals in planning such that we can state things like *if* A *then* B, where A is a fact that can become true and B is a soft goal. For example, we could express the soft goal *if in Paris, visit the Eiffel Tower*, where being in Paris may be a hard or a soft goal, or even just a intermediate location, and visiting the Eiffel Tower is not a hard goal but a soft goal. This increases a planner's capability of creating more user centric plans, by incorporating these preferences into a planning instance.

Acknowledgments. This work was partly supported by BrainLinks-BrainTools, Cluster of Excellence funded by the German Research Foundation (DFG, grant number EXC 1086). We thank the the anonymous reviewers for their insightful comments, helping in improving the overall quality of the paper.

References

Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.

Jorge A. Baier and Sheila A. McIlraith. Planning with preferences. *AI Magazine*, 29(4):25–36, 2008.

Jorge A. Baier, Fahiem Bacchus, and Sheila A. McIlraith. A heuristic search approach to planning with temporally extended preferences. *Artificial Intelligence Journal (AIJ)*, 173(5–6):593–618, 2009.

J. Richard Büchi. On a decision method in restricted second order arithmetic. In *Proceedings of the International Congress on Logic, Methodology, and Philosophy of Science*, pages 1–11, 1962.

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)*, pages 854–860, 2013.

Giuseppe De Giacomo, Riccardo De Masellis, and Marco Montali. Reasoning on LTL on finite traces: Insensitivity to infiniteness. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI 2014)*, pages 1027–1033, 2014.

Paul Gastin and Denis Oddoux. Fast LTL to Büchi automata translation. In *Proceedings of the 13th International Conference on Computer Aided Verification (CAV 2001)*, pages 53–65, 2001.

Florian Geißer, Thomas Keller, and Robert Mattmüller. Delete relaxations for planning with state-dependent action costs. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, 2015.

Alfonso Gerevini and Derek Long. Plan constraints and preferences in PDDL3: The language of the 5th international planning competition. Technical report, Department of Electronics for Automation, University of Brescia, Italy, 2005.

Rob Gerth, Doron Peled, Moshe Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proceedings of the 15th International Symposium on Protocol Specification, Testing and Verification*, pages 3–18, 1996.

Dimitra Giannakopoulou and Klaus Havelund. Automata-based verification of temporal properties on running programs. In *Proceedings of the 16th Annual International Conference on Automated Software Engineering (ASE 2001)*, pages 412–416, 2001.

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

Emil Keyder and Hector Geffner. Softgoals can be compiled away. *Journal of Artificial Intelligence Research (JAIR)*, pages 547–556, 2009.