Proceedings of the Sixteenth International Conference on
Principles of Knowledge Representation and Reasoning (KR 2018)

Compiling Away Soft Trajectory Constraints in Planning

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

Abstract

Soft goals in planning are optional objectives that should be achieved in the terminal state. However, failing to achieve them does not result in the plan becoming invalid. State trajectory constraints are hard requirements towards the state trajectory of the plan. Soft trajectory constraints are a combination of both: soft preferences on how the hard goals are reached, i. e., 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 $\mathsf{LTL_f}$ and deterministic finite automata. We provide two compilation schemes, with and without reward shaping, by rewarding and penalizing different states in the plan. 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. Take for instance a robot scenario where the soft goal could be to have the workbench clean after execution, whereas the main objective is to build some product. These requirements differ from classical *(hard)* goals in that violating them does not render a plan invalid. PDDL 3.0 (Gerevini et al. 2009) 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 optimizes the trade-off between action costs on the one hand and penalties for violated soft goals on the other hand. This corresponds to a constraint optimization problem, where the constraints are the hard goals and the optimization tries to fulfill the soft goals and keep action costs low.

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 discuss how soft trajectory constraints can be compiled away using linear temporal logic on finite traces ($\mathsf{LTL_f}$), deterministic finite automata, conditional effects, and state dependent action costs, generalizing the soft goal compilation introduced by Keyder and Geffner (2009). We first introduce a compilation which adds a penalty to the end of the planning process for each soft trajectory constraint not fulfilled. This is informative to the heuristic, but keeps the search uninformed regarding these constraints. Therefore, we introduce a second compilation, which instantiates the idea of reward shaping (Ng, Harada, and Russell 1999; Camacho et al. 2017) for our setting, resulting in a more informed search. Note that potential-based reward shaping typically makes rewards "more state-dependent". We therefore see an added value of our implementation in the support of such state-dependent rewards, also within the heuristic. Our approach allows us to then 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 and introduce different preference formalisms based on quantitative and qualitative languages. Using quantitative languages, the degree of preference for a given plan can be expressed 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 degree of preference for a plan can be determined over a set of properties, such soft goals that may be satisfied or violated. Such a system was implemented in PDDL3 (Gerevini et al. 2009), where preferences can be specified as temporal, or temporally extended predicates, using a subset of LTL.

Baier, Bacchus, and McIlraith (2009) describe a method of compiling problems with temporally extended preferences into simpler versions where preferences can only be expressed over the final state, and can be evaluated using an objective function. The authors achieve this by translating the preferences expressed in LTL into parametrized non-deterministic finite state automata (PNFA). 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 state. 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 for each soft goal, which can be achieved in two ways: by an action $\mathit{collect}(p)$ which has cost zero but requires the soft goal to be satisfied, or by an action $\mathit{forgo}(p)$ that has cost equal to the utility of p, but can be executed even if the soft goal is violated. These *collect* and *forgo* actions are forced to be executed at the very end of the plan. However, their work 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*.

Later work by Torres and Baier (2015) introduced a compilation for *hard trajectory constraints*, using synchronization actions between an automaton representation of the LTL constraint and the planning state. However, they do not consider the *soft trajectory constraint* case, and require additional actions for the synchronization step.

A similar approach to the one we present in this paper was presented by Camacho et al. (2017) for MDPs, where they use LTL to model non-Markovian rewards, and also employ reward shaping. The main difference between their work and ours is the overall setting (MDPs vs. classical planning). A similar approach to that of Camacho et al. (2017) was given by Brafman, De Giacomo, and Patrizi (2018). However, they focus on introducing a more expressive language LDL$_\mathsf{f}$, again in the MDP setting.

Preliminaries

Throughout this work, we assume that a finite set of *state variables* $\mathcal{V} = \{v_1, \ldots, v_n\}$ is given, 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 such that $(v, d), (v, d') \in s$ implies $d = d'$. We identify s with the corresponding conjunction of facts, viewing facts as atomic formulas. For a set of variables $\mathcal{V}' \subseteq \mathcal{V}$, if s assigns a value to each $v \in \mathcal{V}'$, s is called a *state* over $\mathcal{V}'$. By $S(\mathcal{V}')$ we refer to the set of all states over $\mathcal{V}'$, and we write S for $S(\mathcal{V})$.

Linear-time temporal logic on finite traces

Linear-Time Temporal Logic (LTL) is a modal logic capable of expressing logic formulas referring to discrete linear time, and can be used to express trajectory constraints. An LTL formula φ over a set of variables $\mathcal{V}$ 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 over $\mathcal{V}$. 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 use weak until $\varphi\mathcal{W}\psi$ as an abbreviation for $\varphi\mathcal{U}\psi \vee \Box\varphi$. By $\mathcal{V}(\varphi)$ we refer to the set of variables mentioned in φ, and by $\mathcal{V}(\Phi)$ to those mentioned in any $\varphi \in \Phi$, if Φ is a set of LTL formulas. The semantics of LTL$_\mathsf{f}$ (LTL on finite traces) is defined as the interpretation over finite traces denoting a sequence of instants of time. Let φ be an LTL$_\mathsf{f}$ formula, and let $\mu = (\mu(0), \mu(1), \ldots, \mu(n))$ be such a finite trace with $\mu(i) \in S(\mathcal{V}(\varphi))$ for all $i = 0, \ldots, n$. Then the truth of φ along trace μ is defined as follows (De Giacomo and Vardi 2013):

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

Trajectory constraints as LTL$_\mathsf{f}$

PDDL 3.0 (Gerevini et al. 2009) 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, Masellis, and Montali (2014), these can be expressed using LTL$_\mathsf{f}$:

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

$$
\begin{aligned}
(\text{at-most-once } \varphi) &:= \Box(\varphi \to \varphi \mathcal{W} \neg\varphi)\\
(\text{sometime-after } \varphi\ \psi) &:= \Box(\varphi \to \Diamond\psi)\\
(\text{sometime-before } \varphi\ \psi) &:= (\neg\varphi \wedge \neg\psi)\mathcal{W}(\neg\varphi \wedge \psi)\\
(\text{sometime-within } n\ \varphi\ \psi) &:= \Box(\varphi \to \bigvee_{0 \le i \le n} \underbrace{\bigcirc \dots \bigcirc}_{i} \psi)
\end{aligned}
$$

Here, φ and ψ are propositional formulas, and n, n_1, n_2 natural numbers. The formula *last* is a shorthand for $\neg \bigcirc \top$, which characterizes the last state of the state trajectory.

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 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: $\mathcal{V}$ is a finite set of finite-domain state variables (as above). A is a set of *actions*, and each action is a pair $a = \langle \mathit{pre}, \mathit{eff} \rangle$, where pre is a partial variable assignment (or a consistent conjunction of facts) called the *precondition*, and where eff is an *effect* of the form $\mathit{eff} = \bigwedge_{i=1,\dots,n}(\mathit{pre}_i \rhd \mathit{eff}_i)$ for some number $n \in \mathbb{N}$ of *conditional effects*, each consisting of an effect condition pre_i, again a partial variable assignment, and an effect 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 $c_a : S \to \mathbb{N}$ 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 the importance we assign to satisfying φ. For states s, we use function notation $s(v) = d$ and set notation $(v, d) \in s$ interchangeably. The *change set* $[\mathit{eff}]_s$ of effect $\mathit{eff} = \bigwedge_{i=1,\dots,n}(\mathit{pre}_i \rhd \mathit{eff}_i)$ in state s is the set of facts that eff makes true if applied in s, i. e., the set $\bigcup_{i=1,\dots,n}[\mathit{pre}_i \rhd \mathit{eff}_i]_s$, 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 $s \models \mathit{pre}$ 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 $s \models s_\star$. We denote the set of goal states by $S_\star$.

Following an idea of De Giacomo, Masellis, and Montali (2014), we assume that Π contains a variable last with $\mathcal{D}_{\mathsf{last}} = \{\mathit{true}, \mathit{false}\}$ that is initially *false* and that ought to be *true* iff the last state of the state trajectory has been reached, i. e., if $\neg \bigcirc \top$ holds. To ensure that last is *true* exactly in the last state, we assume that there are two copies of each action, a regular one, and one that has last as an additional effect, marking termination. All actions then have $\neg\mathsf{last}$ as an additional precondition. Moreover, for correct synchronization with the automaton that recognizes the language of a state-trajectory constraint φ, we require a terminal action $\mathit{lastop} = \langle \mathsf{last} \wedge \neg\mathsf{done}, \mathsf{done} \rangle$ that is only applicable *after* termination has been marked and that makes another fresh auxiliary proposition done true (which is initially false), with cost $c_{\mathit{lastop}} = 0$. We assume that Π is reformulated such that done is an additional goal condition.

Let $\pi = (a_0, \dots, a_{n-1})$ be a sequence of actions from A. We call π *applicable* in s_0 if there exist states $s_1, \dots, s_n$ such that a_i is applicable in s_i and $s_{i+1} = s_i[a_i]$ for all $i = 0, \dots, n-1$. We call π a *plan* for Π if it is applicable in s_0 and if $s_n \in S_\star$. In that case, we call $\mu_\pi = (s_0, s_1, \dots, s_{n-1})$ the *state trajectory induced by π in s_0*.[1] 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 $\Sigma = S(\mathcal{V}')$, where $\mathcal{V}' = \mathcal{V}(\Phi)$ is the set of variables that are relevant to the soft trajectory constraints.[2] 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 $\langle s, s|_{\mathcal{V}'}, s' \rangle$, where $s' = s[a]$ for some $a \in A$ that is applicable in s, and where $s|_{\mathcal{V}'}$ is s restricted to the relevant variables $\mathcal{V}'$. What is 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 deterministic finite automata for $\mathsf{LTL_f}$.

Deterministic finite automata for $\mathsf{LTL_f}$

De Giacomo, Masellis, and Montali (2014) provide an algorithm for creating a non-deterministic finite automaton (NFA) from a given $\mathsf{LTL_f}$ formula φ, which first requires the $\mathsf{LTL_f}$ formula φ to be in negation normal form, and which additionally requires the predicate last, which is only true in the last planning state, to be present. The resulting NFA is worst-case exponential in the size of the input formula φ. Transforming the NFA to a DFA can then be accomplished by the standard powerset construction (Rabin and Scott 1959) and results in

[1] We deliberately leave out the last state s_n reached by applying *lastop*, since it is only an artifact of our encoding and should not affect whether a given trajectory constraint is satisfied or not.

[2] For convenience, we drop action names from the transition labels. For plan reconstruction, we would have to include them here.

yet another exponential blow-up in the size of the NFA, yielding a doubly exponential blow-up overall. The input alphabet Σ of the resulting DFA $\mathcal{A}(\varphi)$ is the set of all states $S(\mathcal{V}')$, where again $\mathcal{V}' = \mathcal{V}(\Phi)$ is the set of all variables relevant to any of the trajectory constraints (including last). The DFA $\mathcal{A}(\varphi)$ accepts a finite input trace μ over Σ with last being true exactly in the last state of μ iff $\mu \models \varphi$.

Now, for a planning task Π with a *hard* state-trajectory constraint φ, the standard automaton construction considers the product automaton $\mathcal{A}^\times$ of $\mathcal{A}(\Pi)$ and $\mathcal{A}(\varphi)$. Then, a state trajectory μ is the induced trajectory of some plan π of Π satisfying φ iff μ is accepted by $\mathcal{A}^\times$. 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 (a) the product construction, (b) an assignment of action costs that reflects the satisfaction or violation of soft trajectory constraints, and (c) a compact encoding of the product automaton as a new planning task Π'.

Tracking soft trajectory constraints

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 *totalcost* as defined above. Transition costs aside, the semantics of Π are captured by the product automaton $\mathcal{A}^\times = \mathcal{A}(\Pi) \times \prod_{\varphi\in\Phi} \mathcal{A}(\varphi)$. However, when compiling away soft trajectory constraints, for the sake of a compact representation and subsequent on-the-fly plan generation, we do not want to generate an *automaton*, but rather another *planning task* Π' such that $\mathcal{A}(\Pi')$ is isomorphic to $\mathcal{A}^\times$. 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 $\mathcal{A}(\varphi)$. 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.

Formally, let $\mathcal{A}(\varphi) = \langle \Sigma, Q, \Delta, q_0, Q_a\rangle$ be a DFA for φ. Then we create the planning task $\Pi' = \langle \mathcal{V}', A', s'_0, s'_\star, \emptyset\rangle$ with $\mathcal{V}' = \mathcal{V} \cup \{\tau_\varphi\}$, with domain Q for τ_φ. The initial state s'_0 agrees with s_0 on all variables in $\mathcal{V}$, and additionally, $s'_0(\tau_\varphi) = q_0$. The actions are $A' = \{a' \mid a \in A\}$, where $a' = \langle \mathit{pre}', \mathit{eff}'\rangle$ is constructed from $a = \langle \mathit{pre}, \mathit{eff}\rangle$ as follows: $\mathit{pre}' = \mathit{pre}$ and

$$\mathit{eff}' = \mathit{eff} \wedge \bigwedge_{\langle q,\sigma,q'\rangle\in\Delta} (((\tau_\varphi = q) \wedge \sigma) \rhd \tau_\varphi := q').$$

In words, we add conditional effects to track the value of τ_φ for each transition in $\mathcal{A}(\varphi)$. Action costs are unaffected, i. e., $c_{a'} = c_a$ for all $a \in A$. Also, the goal description remains unchanged, i. e., $s'_\star = s_\star$.

States of $\mathcal{A}(\Pi')$ are then (isomorphic to) pairs (s, q) consisting of a state s of $\mathcal{A}(\Pi)$ and a state q of $\mathcal{A}(\varphi)$, where q is the DFA state *before* reading state s. Since q is always "one step behind", we still need the artificial last action lastop that reads the last state of $\mathcal{A}(\Pi)$ and advances the state of $\mathcal{A}(\varphi)$ accordingly. Additional formal machinery needed for the evaluation of the penalty term is deferred until after the following proposition.

Proposition 1. *Up to preservation of accepting states, $\mathcal{A}(\Pi')$ is isomorphic to $\mathcal{A}(\Pi) \times \mathcal{A}(\varphi)$.*

Proof sketch. We consider two automata $\mathcal{A} = \langle \Sigma, Q, \Delta, q_0, Q_a\rangle$ and $\mathcal{A}' = \langle \Sigma, Q', \Delta', q'_0, Q'_a\rangle$ over the same alphabet Σ to be isomorphic iff there is a structure-preserving bijection $\beta : Q \to Q'$ such that $\beta(q_0) = q'_0$, that $q \in Q_a$ iff $\beta(q) \in Q'_a$ for all $q \in Q$, and that $\langle q, \sigma, q'\rangle \in \Delta$ iff $\langle \beta(q), \sigma, \beta(q')\rangle \in \Delta'$ for all $q, q' \in Q$, $\sigma \in \Sigma$.

Now, let $\mathcal{A}(\Pi) = \langle \Sigma, Q, \Delta, q_0, Q_a\rangle$, $\mathcal{A}(\varphi) = \langle \Sigma, Q^\varphi, \Delta^\varphi, q^\varphi_0, Q^\varphi_a\rangle$, and $\mathcal{A}(\Pi') = \langle \Sigma, Q', \Delta', q'_0, Q'_a\rangle$. Notice that they all share the same input alphabet $\Sigma = S(\mathcal{V}(\varphi))$. Then, $\mathcal{A}^\times = \mathcal{A}(\Pi) \times \mathcal{A}(\varphi) = \langle \Sigma, Q^\times, \Delta^\times, q^\times_0, Q^\times_a\rangle$ with $Q^\times = Q \times Q^\varphi$, $q^\times_0 = (q_0, q^\varphi_0)$, $Q^\times_a = Q_a \times Q^\varphi_a$, and a transition $((q, q^\varphi), \sigma, (q', q^{\varphi\prime})) \in \Delta^\times$ iff $(q, \sigma, q') \in \Delta$ and $(q^\varphi, \sigma, q^{\varphi\prime}) \in \Delta^\varphi$.

The claimed bijection $\beta : Q^\times \to Q'$ is given by $\beta((s, q)) = s \cup \{\tau_\varphi \mapsto q\}$. Then β obviously preserves the initial state. Goal/accepting states are deliberately *not* preserved in Π', since we want to encode satisfaction or violation of φ in the plan costs for Π', not in its goal condition. Therefore, for goals, we only have that $q^\times = (q, q^\varphi) \in Q^\times_a$ iff $q \in Q_a$.

Finally, there is a transition $((q, q^\varphi), \sigma, (q', q^{\varphi\prime})) \in \Delta^\times$ iff $(q, \sigma, q') \in \Delta$ and $(q^\varphi, \sigma, q^{\varphi\prime}) \in \Delta^\varphi$. Now, since $(q, \sigma, q') \in \Delta$, there must be some action a that is applicable in q and leads to q'. With the construction of Π' from Π, and the definition of change sets, this implies that the modified action a' is applicable in $\beta(q, q^\varphi)$ and, because $(q^\varphi, \sigma, q^{\varphi\prime}) \in \Delta^\varphi$, leads to $\beta(q', q^{\varphi\prime})$, i. e., $(\beta(q, q^\varphi), \sigma, \beta(q', q^{\varphi\prime})) \in \Delta'$. The opposite direction can be proven similarly. □

Goal action penalty compilation

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 $\mathsf{penalized}$ to Π' that is initially false, and add $\mathsf{penalized}$ to the goal $s_\star$. This means that every plan for Π' has to include an occurrence of the new action $\mathit{penalize} = \langle \mathsf{done} \wedge \neg\mathsf{penalized}, \mathsf{penalized}\rangle$ as its last step. The cost function of the action $\mathit{penalize}$ now simply determines the penalty value $\mathit{penalty}(\pi)$ 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 DFA states in the current planning state. More formally, $c_{\mathit{penalize}} = \sum_{\varphi\in\Phi} [\tau_\varphi \notin Q^\varphi_a] w_\varphi$ where $[\tau_\varphi \notin Q^\varphi_a] = 1$ if $\tau_\varphi = q$ and $q \notin Q^\varphi_a$ for some $q \in Q^\varphi$, and 0 otherwise.

Notice that the action $\mathit{penalize}$ has state-dependent costs that are not universally supported by planning systems. How-

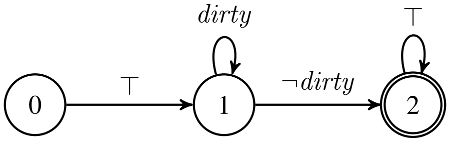


Figure 1: DFA for (hold-after 1 ¬*dirty*).

ever, those can be compiled away to state-independent costs, if this is desired (Geißer, Keller, and Mattmüller 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 $\text{is\_violated}_\varphi$ can be added to the planning task that is true iff the value of τ_φ represents a non-accepting state. Then $c_{penalize} = \sum_{\varphi \in \Phi}[\text{is\_violated}_\varphi] w_\varphi$. A natural modeling will treat $\text{is\_violated}_\varphi$ as a derived variable, and will have axioms that express $\text{is\_violated}_\varphi$ 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. Proposition 3 below).

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 (including the action penalize). Then an optimal plan for Π' is also an optimal plan for Π (without the action penalize).*

Proof sketch. 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 be an action and φ the preference (hold-after 1 ¬dirty), stating that the fact dirty should be false sometime after one step. We can then track the state in the automaton in Figure 1 by adding the following conditional effects to a:*

$$\begin{aligned} (\tau_\varphi = 0) &\;\rhd\; (\tau_\varphi := 1) \quad \wedge \\ (\tau_\varphi = 1) \wedge dirty &\;\rhd\; (\tau_\varphi := 1) \quad \wedge \\ (\tau_\varphi = 1) \wedge \neg dirty &\;\rhd\; (\tau_\varphi := 2) \quad \wedge \\ (\tau_\varphi = 2) &\;\rhd\; (\tau_\varphi := 2) \end{aligned}$$

Clearly, the two conditional effects that do not change the value of τ_φ can be dropped. Also, generally, conditions under which τ_φ obtains the same new value can be combined into a single disjunction (not shown here). The partial cost function c for this preference is $c = [\tau_\varphi \in \{0, 1\}]w_\varphi$, and it is added to the cost of the penalize action. This adds w_φ to the total plan cost if $\mathcal{A}(\varphi)$ is in one of the non-accepting states 0 or 1.

Geißer, Keller, and Mattmüller (2015) show how state-dependent action costs can be compiled away using edge-valued multi-valued decision diagrams (EVMDD) by representing the cost function as an EVMDD and introducing an auxiliary operator for each edge in the diagram. Later, Mattmüller et al. (2018) also showed how to combine this with conditional effects for a heuristic-friendly compilation. An analysis of the EVMDD compilation of our *penalize* action shows that the auxiliary operators correspond to the *collect*, *forgo*, and *end* from 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.

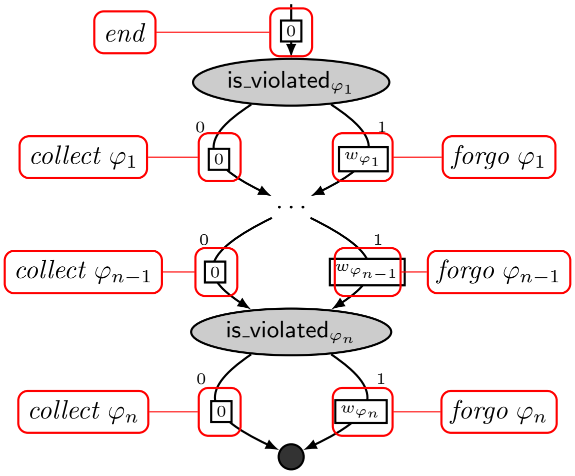


Figure 2: EVMDD compilation of the *penalize* action with derived variables $\text{is\_violated}_{\varphi_i}$, which are true if τ_{φ_i} is in a non-accepting state. Numbers on edges are partial costs (= costs of compiled actions).

Proposition 3. *The EVMDD-based action compilation of Geißer, Keller, and Mattmüller (2015), applied to the action penalize, is essentially the soft goal compilation by Keyder and Geffner (2009).*

Proof sketch. For each soft trajectory constraint φ_i, we introduce an auxiliary variable $\text{is\_violated}_{\varphi_i}$ which is true iff the corresponding DFA is in a non-accepting state. We can then express the cost of the *penalize* action as $c_{penalize} = \sum_{\varphi \in \Phi}[\text{is\_violated}_\varphi]w_\varphi$. Expressed as an edge-valued multi-valued decision diagram (EVMDD) (Geißer, Keller, and Mattmüller 2015), $c_{penalize}$ looks as depicted in Figure 2 (without the red annotations). The EVMDD-based action compilation of Geißer, Keller, and Mattmüller (2015) now turns each edge of the EVMDD into a new auxiliary action, and adds some bookkeeping machinery to ensure that the EVMDD is traversed exactly once from top to bottom. These new auxiliary 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, up until the *penalize* action, the achievement of any soft trajectory constraint is only represented by the h-value (the heuristic estimate of the remaining cost to reach a goal state). A more desirable compilation would provide the search with a more accurate g-value (the cost of reaching the current node from the initial state), 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 more accurate g-values that reflect the current acceptance status of φ. 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 $\mathcal{A}(\varphi)$, 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 $\mathcal{A}(\varphi)$.

Whenever an action a changes the value of τ_φ, thus transitioning from one state q to another state q' in $\mathcal{A}(\varphi)$, 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 $\mathcal{A}(\varphi)$, 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 add a reward. The partial cost of a transition from s to s' associated with trajectory constraint $\varphi \in \Phi$ is then $\omega_\varphi(q, q')$, where q and q' are the values of τ_φ in s and s', respectively, and where $\omega_\varphi(q, q')$ is a pre-specified penalty or reward term. 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. Transitions that preserve the acceptance status should neither be penalized nor rewarded. For the concrete value of $\omega_\varphi(q, q')$, we use the value from the original soft trajectory constraint'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 $\mathcal{A}(\varphi)$ by applying negative costs. Note, that $\omega_\varphi(q, q')$ only accounts once in the total cost, as we can never add $\omega_\varphi(q, q')$ twice in a row without subtracting it in between first.

By construction, minimizing $totalcost$ in the compiled task Π' amounts to the same as minimizing $totalcost$ in the original task Π. One minor detail to take in to account is if the initial state of $\mathcal{A}(\varphi)$ 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 approach sketched above can also be seen as a form of potential-based reward shaping (Ng, Harada, and Russell 1999) from Markov Decision Processes (MDPs), where $R(s, a, s')$ is the reward gathered by traversing from state s to s' with action a. In our case, we have negative costs for rewards, and positive costs for penalties. The shaped reward function then is $R'(s, a, s') = R(s, a, s') + F(s, a, s')$, where F is the shaping function defined over the current and next state: $F(s, a, s') = \gamma \cdot \theta(s') - \theta(s)$, where θ is the potential function defined over states. For us, $\theta(s) = \sum_{\varphi \in \Phi} [\mathsf{is\_violated}_\varphi] w_\varphi$. We use a discount factor $\gamma = 1$.

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. 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. 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 *totalcost*$(\pi') \geq$ *totalcost*(π), 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.

Proposition 4. *Let Π' be the compiled task from Π with metric preserving costs (Table 1a). Then an optimal plan for Π' is also an optimal plan for Π (without the $penalize$ action).*

Proof sketch. From Proposition 1 we get that the compilation is sound and complete. The objective function of the original task is $penalty(\pi) + cost(\pi)$, where $cost(\pi)$ is the sum of all action costs and $penalty(\pi)$ is the sum of all penalties for not achieved soft trajectory constraints. Using metric preserving costs, we get that at each step of the plan, the current total cost is equal to the sum of all applied action costs plus the sum of all penalties for entering a non-accepting soft trajectory state minus all rewards for entering an accepting state of the soft trajectory constraints. Thus at each step the total cost is equal to the applied action costs plus penalties for currently not achieved soft trajectory constraints. In the goal state this is equal to $penalty(\pi) + cost(\pi)$ the original objective function. □

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

(a) Metric Preserving Costs

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

(b) Positively Shifted Costs

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

(c) Adapted Positively Shifted Costs

Table 1: State Transition Costs

Following from Proposition 4 we can show that using positively shifted or adapted positively shifted costs does not preserve optimality. This is due to the fact that for each step in the plan for which a soft trajectory constraint is in a non-accepting state, we add the penalty to the total cost. For tasks without hard goals it is easy to see that the empty plan executing only the *penalize* action gathering all penalties only once is preferred over plans that require more than one action to fulfill the soft trajectory constraints, which accumulate penalties for each state where soft trajectory constraints are not fulfilled.

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 subset of the benchmark problems from the fifth International Planning Competition (IPC-5) plus the Rovers domain from the IPC-3. The overall results of the experiments show that our approach is not only sound in theory, but also provides sufficient results in practice. We first discuss the detailed 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%, 30%, 50%, 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 the blind heuristic h^{blind} that assigns estimate 1 to all states except for goal states, to which it assigns estimate 0, the maximum heuristic h^{max} (Bonet and Geffner 2001), and the canonical pattern database heuristic h^{cpdb} (Haslum et al. 2007) for the optimal track. For the satisficing benchmarks, we used the additive heuristic h^{add} (Bonet and Geffner 2001) and the FF heuristic h^{FF} (Hoffmann and Nebel 2001) 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 pattern database heuristic, as this heuristic does not support axioms, which are introduced by the translate step in the Fast Downward planner.

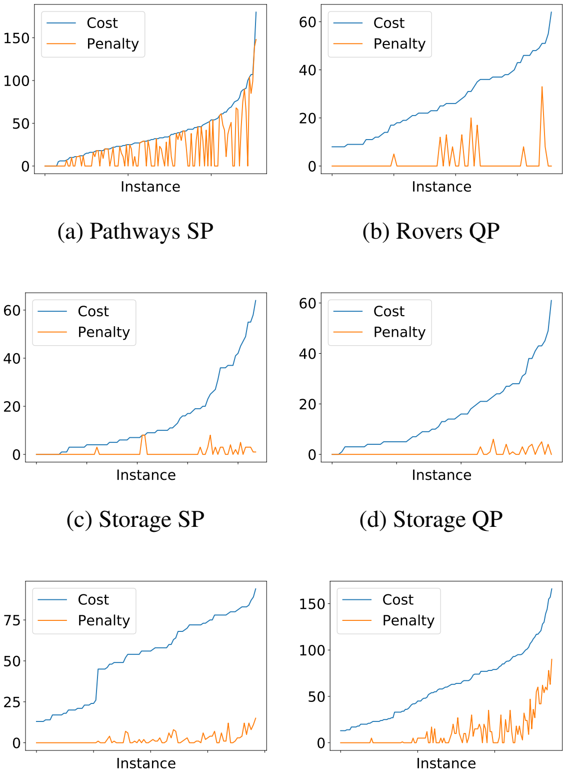
As can be seen in Figure 3, the satisficing benchmark performed rather well on the Rovers, Storage, and Trucks SP domain, as their penalty is always close to zero. The quality of the Trucks QP domain is slightly worse as fulfilling all soft trajectory constraints becomes more difficult, the more complex 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, 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.

Domain	h^{blind}	h^{max}	h^{cpdb}
pathways SP	8.10%	10.00%	8.10%
rovers QP	17.14%	21.43%	15.17%
storage SP	34.81%	39.26%	24.44%
storage QP	24.78%	29.20%	23.01%
trucks SP	23.71%	27.84%	n/a
trucks QP	18.84%	23.19%	n/a

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.

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}, and h^{cpdb} was used, as it requires negative



(a) Pathways SP

(b) Rovers QP

(c) Storage SP

(d) Storage QP

(e) Trucks SP

(f) Trucks QP

Figure 3: Plan quality of the satisficing benchmarks, ordered by *total cost* using goal action penalty compilation and h^{FF} heuristic.

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 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 did not change significantly on these two compilations. However, one needs to keep in mind, that the optimal plan for these compilations is not the same as for the metric preserving cost compilation. The accumulative penalty of staying in non accepting states, leads to the shortest plan being favoured over plans fullfilling soft trajectory constraints. 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^{cpdb}
pathways SP	39.05%	n/a	38.57%
rovers QP	16.43%	n/a	15.00%
storage SP	21.37%	n/a	20.51%
storage QP	23.40%	n/a	23.40%
trucks SP	20.62%	n/a	n/a
trucks QP	19.23%	n/a	n/a

(a) Metric Preserving Costs

Domain	h^{blind}	h^{max}	h^{cpdb}
pathways SP	40.48%	39.05%	39.05%
rovers QP	16.55%	17.99%	15.11%
storage SP	29.46%	28.68%	24.81%
storage QP	26.42%	23.58%	24.47%
trucks SP	18.39%	19.54%	n/a
trucks QP	21.82%	21.82%	n/a

(b) Positively Shifted Costs

Domain	h^{blind}	h^{max}	h^{cpdb}
pathways SP	40.48%	39.05%	39.05%
rovers QP	16.43%	17.86%	15.00%
storage SP	29.46%	28.68%	27.81%
storage QP	25.47%	22.64%	24.53%
trucks SP	18.39%	19.54%	n/a
trucks QP	18.90%	19.69%	n/a

(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 satisfied soft trajectory constraints, 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 satisfied 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

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.

optimize towards their soft goals, whereas harder instances where not solved at all.

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 finite state 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, and the penalty can be added directly in the *penalize* action. Additionally, optimizations to the cost functions can be made. We expect these measures to improve the overall performance of our approach In our compilation, we introduced negative action costs, 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.

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

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

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

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence Journal (AIJ)* 129(1-2):5–33.

Brafman, R. I.; De Giacomo, G.; and Patrizi, F. 2018. LTL$_f$/LDL$_f$ non-Markovian rewards. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*, 1771–1778.

Camacho, A.; Chen, O.; Sanner, S.; and McIlraith, S. A. 2017. Non-Markovian rewards expressed in LTL: Guiding search via reward shaping. In *Proceedings of the 10th International Symposium on Combinatorial Search (SoCS)*, 159–160.

De Giacomo, G., and Vardi, M. Y. 2013. Linear temporal logic and linear dynamic logic on finite traces. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, 854–860.

De Giacomo, G.; Masellis, R. D.; and Montali, M. 2014. Reasoning on LTL on finite traces: Insensitivity to infiniteness. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI)*, 1027–1033.

Geißer, F.; Keller, T.; and Mattmüller, R. 2015. Delete relaxations for planning with state-dependent action costs. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, 1573–1579.

Gerevini, A.; Haslum, P.; Long, D.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the Fifth International Planning Competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence* 173(5-6):619–668.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; Koenig, S.; et al. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI)*, 1007–1012.

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

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research (JAIR)* 14:253–302.

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

Mattmüller, R.; Geißer, F.; Wright, B.; and Nebel, B. 2018. On the relationship between state-dependent action costs and conditional effects in planning. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*, 6237–6245.

Ng, A. Y.; Harada, D.; and Russell, S. J. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the 16th International Conference on Machine Learning (ICML)*, 278–287.

Rabin, M. O., and Scott, D. 1959. Finite automata and their decision problems. *IBM Journal of Research and Development* 3(2):114–125.

Torres, J., and Baier, J. A. 2015. Polynomial-time reformulations of LTL temporally extended goals into final-state goals. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, 1696–1703.