Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)

On Planning with Qualitative State-Trajectory Constraints in PDDL3 by Compiling them Away

Luigi Bonassi,[1] **Alfonso Emilio Gerevini,**[1] **Francesco Percassi,**[2] **Enrico Scala**[1]
[1] Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Brescia, Italy
[2] School of Computing and Engineering, University of Huddersfield, UK
l.bonassi005@unibs.it, alfonso.gerevini@unibs.it, f.percassi@hud.ac.uk, enrico.scala@unibs.it

Abstract

We tackle the problem of classical planning with qualitative state-trajectory constraints as those that can be expressed in PDDL3. These kinds of constraints allow a user to formally specify which temporal properties a plan has to conform with through a class of LTL formulae. We study a compilation-based approach that does not resort to automata for representing and dealing with such properties, as other approaches do, and generates a classical planning problem with conditional effects that is solvable iff the original PDDL3 problem is. Our compilation exploits a regression operator to revise the actions' preconditions and conditional effects in a way to (i) prohibit executions that irreversibly violate temporal constraints (ii) be sensitive to executions that traverse those necessary subgoals implied by the temporal specification. An experimental analysis shows that our approach performs better than other state-of-the-art approaches over the majority of the considered benchmark domains.

Introduction

The PDDL3.0 language (Gerevini et al. 2009) is a popular and standard planning formalism that can express state-trajectory constraints in planning problems. PDDL3.0 provides primitives allowing the specification of such constraints through a subclass of LTL formulas (Pnueli 1977) that has been shown to be useful in planning (Gerevini et al. 2009). In PDDL3.0, a classical planning problem extended with state-trajectory constraints is the task of synthesizing a course of actions that reaches the problem goals, and generates a sequence of states that conforms with the given state-trajectory constraints.

Two lines of research have been investigated to address this reasoning task: the former aims at efficiently supporting the trajectory constraints directly into the search engines via sophisticated heuristic adaptations (Coles and Coles 2011; Benton, Coles, and Coles 2012) or problem decomposition (Chen, Wah, and Hsu 2006; Hsu et al. 2007); the latter uses a compilation-based approach (Baier, Bacchus, and McIlraith 2009; Baier and McIlraith 2006; Edelkamp, Jabbar, and Nazih 2006; Edelkamp 2006; Torres and Baier 2015; Wright, Mattmüller, and Nebel 2018). The compilation approach of a problem with trajectory constraints aims at generating an equivalent representation without them that can be effectively handled via off-the-shelf classical planners. Although native approaches have the potential to customize search and heuristics for the problem at hand, they lack the modularity of a compilation-based schema. Compilation-based approaches have been investigated also for a class of trajectory constraints much richer than PDDL3.0 constraints, expressible through the full Linear Temporal Logic (LTL) (Pnueli 1977) interpreted over finite traces (Giacomo and Vardi 2013). However, this generality in expressing trajectory constraints can be a barrier for scaling up the planner performance. Indeed, to handle the expressive power of arbitrary nested LTL formulae (unsupported in PDDL3.0), current compilation-based mechanisms undergo a complex passage through the automata representing such formulae.

We propose a novel compilation-based schema that is specifically designed for qualitative PDDL3.0 trajectory constraints. Our compilation substantially revisits the automataless approach proposed by Percassi and Gerevini (2019) for soft trajectory constraints through the lens of *hard* trajectory constraints. We do so by revising the schema in two main important aspects: first, we characterize the state-trajectory constraints into two classes. The former class encompasses constraints for which it is possible to always prune any plan prefix that violates them. The latter class encompasses constraints that induce intermediate goals, landmarks (Richter and Westphal 2010) that every plan needs to traverse. This characterization leads to devise a simple compilation that can efficiently prune plan prefixes not complying with the first class of constraints, and that is aware of those plan prefixes where intermediate goals are reached. Then, we revisit how we anticipate the effects of the action during search, and do so by exploiting Rintanen (2008)'s notion of regression for the formulas mentioned in the trajectory constraints.

Our experimental analysis shows that the novel compilation substantially extends the reach of planning over the considered class of planning problems.

Background and Preliminaries

We borrow standard notions from propositional logic and the work by Gerevini et al. (2009) on PDDL3.0; the reader is referred to this work for a complete treatment of the language.

A classical planning problem is a tuple $\Pi = \langle F, A, I, G \rangle$

where F is a set of atoms, $I \subseteq F$ is the initial state, G is a formula over F, and A is a set of actions. An action $a \in A$ is a pair $\langle Pre(a), Eff(a) \rangle$, where $Pre(a)$ is a formula over F expressing the preconditions of a, and $Eff(a)$ is a set of conditional effects, each of which is a pair $c \triangleright e$ where: c is a formula and e is set of literals, both over F. With e^- and e^+ we indicate the partition of e featuring only negative and positive literals, respectively. A state s is a subset of F, with the meaning that if $p \in s$, then p is true in s, and if $p \notin s$, p is false in s. An action is applicable in s if $s \models Pre(a)$, and the application of an action a in s yields the state $s' = (s \setminus \bigcup_{\substack{c \triangleright e \in Eff(a) \\ \text{with } s \models c}} e^-) \cup \bigcup_{\substack{c \triangleright e \in Eff(a) \\ \text{with } s \models c}} e^+$. We indicate with $s' = s[a]$ the state resulting from applying action a in s, and assume conflicting effects (p and $\neg p$) are only yield by conditional effects having their conditions mutually exclusive in s. A plan π for a problem $\Pi = \langle F, A, I, G \rangle$ is a sequence of actions $\langle a_0, a_1, ..., a_{n-1} \rangle$ in Π; plan π is valid for Π iff there exists a sequence of states (state trajectory) $\langle s_0, s_1, ..., s_n \rangle$ such that $s_0 = I$, $\forall\ i \in [0, \dots, n-1]$ we have that $s_i \models Pre(a_i)$ and $s_{i+1} = s_i[a_i]$, and $s_n \models G$.

PDDL3.0 introduced state-trajectory constraints as a class of temporal logic formulae over state trajectories. Such constraints can be either soft or hard. While soft constraints are used to define plan quality, hard constraints express necessary conditions that the state trajectory of a valid plan must satisfy. In this work, we focus on hard constraints, and in particular on those that in PDDL3.0 are called "qualitative" (Gerevini et al. 2009), because involving only non-numeric terms. For convenience such a class of constraints here is called *trajectory constraints*.

In addition to the standard problem goals, a trajectory constraint can be of the following types: `always` ϕ (A_ϕ), which requires that every state traversed by the plan satisfies formula ϕ; `at-most-once` ϕ (AO_ϕ), which requires that formula ϕ is true in at most one continuous subsequence of traversed states; `sometime-before` ϕ ψ ($\mathsf{SB}_{\phi,\psi}$), which requires that if ϕ is true in a state traversed by the plan, then also ψ is true in a previously traversed state; `sometime` ϕ (ST_ϕ), which requires that there is at least one state traversed by the plan where ϕ is true; `sometime-after` ϕ ψ ($\mathsf{SA}_{\phi,\psi}$), which requires that if ϕ is true in a traversed state, then also ψ is true in that state or in a later traversed state.

More formally, a trajectory constraint c is satisfied in a state trajectory $\sigma = \langle s_0, \dots, s_n \rangle$ ($\sigma \models c$) if the following conditions hold:

$\sigma \models (\texttt{always}\ \phi)$ iff $\forall i : 0 \leq i \leq n \cdot s_i \models \phi$

$\sigma \models (\texttt{sometime}\ \phi)$ iff $\exists i : 0 \leq i \leq n \cdot s_i \models \phi$

$\sigma \models (\texttt{at-most-once}\ \phi)$ iff $\forall i : 0 \leq i \leq n \cdot$ if $s_i \models \phi$ then

$\quad \exists j : j \geq i \cdot \forall k : i \leq k \leq j \cdot s_k \models \phi$ and $\forall k : k > j \cdot s_k \models \neg\phi$

$\sigma \models (\texttt{sometime-after}\ \phi\ \psi)$

$\quad$ iff $\forall i : 0 \leq i \leq n \cdot$ if $s_i \models \phi$ then $\exists j : i \leq j \leq n \cdot s_j \models \psi$

$\sigma \models (\texttt{sometime-before}\ \phi\ \psi)$

$\quad$ iff $\forall i : 0 \leq i \leq n \cdot$ if $s_i \models \phi$ then $\exists j : 0 \leq j < i \cdot s_j \models \psi$.

All ϕ/ψ are first order formulae that we assume grounded and expressed in negation normal form (NNF). In our work, we call PDDL3.0 *planning problem* a tuple $\langle \Pi, C \rangle$ where Π is a classical planning problem and C is a set of trajectory constraints; the valid plans of $\langle \Pi, C \rangle$ are all valid plans of Π whose state trajectories satisfy all constraints in C.

Our method uses an operator R for regressing a formula over the effects of an action that is inspired by the notion of regression introduced by Rintanen (2008).

Definition 1 (Regression operator)**.** *The regression $R(\phi, a)$ of a NNF formula ϕ through the effects $Eff(a)$ of action a is the formula obtained from ϕ by replacing every atom f in ϕ with $\Gamma_f(a) \vee (f \wedge \neg\Gamma_{\neg f}(a))$, where $\Gamma_l(a)$ for a literal l is defined as*

$$\Gamma_l(a) = \bigvee_{\substack{c \triangleright e\, \in\, Eff(a) \\ with\ l \in e}} c.$$

Note that Definition 1 implies that $s[a] \models \phi$ iff $s \models R(\phi, a)$, for every ϕ, s and a.

Compilation Method

This section describes a compilation schema, called TCORE (Trajectory constraints COmpilation via REgression), for translating a PDDL3.0 problem into an equivalent planning problem without constraints, that can be attempted by classical planners supporting conditional effects.

TCORE makes extensive use of the operator R (Def. 1) and of a set of *monitoring atoms*. These are used to extend the action preconditions and conditional effects in order to (i) block invalid extensions of the plan prefix generated during planning, and (ii) keep track of the truth of relevant (w.r.t. trajectory constraints) formulae in the states generated by the plan prefix. The regression makes it possible to identify what actual influence the action has over the trajectory constraint of interest. Monitoring atoms serve the purpose of collecting relevant facts on the plan state trajectory and asserting their truth/falsity. These reflect whether a constraint c has been satisfied (the $hold_c$ atom), or whether some formula ψ has ever held in some state generated by the plan prefix (the $seen_\psi$ atom).

We show how the extended action preconditions and effects are constructed by distinguishing trajectory constraints into two classes: *invariant trajectory constraints* (*ITC*) and *landmark trajectory constraints* (*LTC*). Intuitively, ITCs can be checked along any plan prefix and if they are violated, there is no way the planner can ever re-establish them; they are invariant conditions that must be maintained over the state trajectory of the plan. LTCs are constraints that require certain conditions true at *some* state over the state trajectory of the plan. The ITCs are: A_ϕ, AO_ϕ, $\mathsf{SB}_{\phi,\psi}$. For each AO_ϕ and $\mathsf{SB}_{\phi,\psi}$, we add the fresh predicates $seen_\phi$ and $seen_\psi$ to record whether ϕ and ψ have ever held. The LTCs are: ST_ϕ and $\mathsf{SA}_{\phi,\psi}$. For each LTC, we add a predicate $hold_c$. Such a predicate is meant to record whether the constraint is already satisfied or not according to the current plan prefix.

Algorithm 1 describes the full compilation. As a very first step, we create the necessary atoms (line 3) and setup the initial state so as to reflect the current status of the trajectory constraints; in particular we need to capture if a LTC is already achieved in I, or if a formula (ϕ or ψ) that is necessary

Algorithm 1: TCORE

```
   Input : A PDDL3.0 Planning Problem $\langle\langle F, A, I, G\rangle, C\rangle$
1  $A' = \{\}$
2  $G' = \top$
3  $F' = monitoringAtoms(C)$ ;    // Additional atoms generation
4  $I' = \bigcup_{c:\mathsf{ST}_\phi \in C} \{hold_c \mid I \models \phi\} \cup \bigcup_{c:\mathsf{SA}_{\phi,\psi} \in C} \{hold_c \mid I \models \psi \vee \neg\phi\} \cup$
       $\bigcup_{\mathsf{SB}_{\phi,\psi} \in C} \{seen_\psi \mid I \models \psi\} \cup \bigcup_{\mathsf{AO}_\phi \in C} \{seen_\phi \mid I \models \phi\}$
5  if $\exists \mathsf{SB}_{\phi,\psi} \in C . I \models \phi \vee \exists \mathsf{A}_\phi \in C . I \models \neg\phi$ then
6      return Unsolvable Problem;    // It exists an unsatisfiable ITC
7  foreach $a \in A$ do
8      $E = \{\}$
9      foreach ITC $c \in C$ do
10         if $c$ is $\mathsf{A}_\phi$ then
11             $\rho = R(\neg\phi, a)$
12         else if $c$ is $\mathsf{AO}_\phi$ then
13             $\rho = R(\phi, a) \wedge seen_\phi \wedge \neg\phi$
14             $E = E \cup \{R(\phi, a) \rhd \{seen_\phi\}\}$
15         else if $c$ is $\mathsf{SB}_{\phi,\psi}$ then
16             $\rho = R(\phi, a) \wedge \neg seen_\psi$
17             $E = E \cup \{R(\psi, a) \rhd \{seen_\psi\}\}$
18         $Pre(a) = Pre(a) \wedge \neg\rho$
19     foreach LTC $c \in C$ do
20         if $c$ is $\mathsf{ST}_\phi$ then
21             $\rho = R(\phi, a)$
22         else if $c$ is $\mathsf{SA}_{\phi,\psi}$ then
23             $E = E \cup \{R(\phi, a) \wedge \neg R(\psi, a) \rhd \{\neg hold_c\}\}$
24             $\rho = R(\psi, a)$
25         $E = E \cup \{\rho \rhd \{hold_c\}\}$
26     $Eff(a) = Eff(a) \cup E$
27     $A' = A' \cup \{\langle Pre(a), Eff(a)\rangle\}$
28 foreach LTC $c \in C$ do
29     $G' = G' \wedge hold_c$
30 return Classical Planning Problem $\langle F \cup F', A', I \cup I', G \wedge G'\rangle$
```

for the evaluation of an ITC is already true in I. Depending on the kind of input constraint, we set the associated monitoring atom: for each $c = \mathsf{ST}_\phi$ such that ϕ is already true in I, and for each $c = \mathsf{SA}_{\phi,\psi}$ such that ψ is already true in I or ϕ is false in I, we set $hold_c$ true in I'. Analogously, for all constraints $\mathsf{SB}_{\phi,\psi}$ (AO_ϕ) we set $seen_\psi$ ($seen_\phi$) true in I' if ψ (ϕ) is true in I. Then we check whether any ITC is already unsatisfied; if so, the problem is unsolvable.

After the initialization phase, the algorithm iterates over all actions and constraints to modify each original action model (preconditions and effects) by considering the interactions between the constraints and the action model. If the constraint is a ITC, the algorithm determines, by regression, a condition ρ such that, if ρ holds in the state where the action is applied, the execution of such an action will violate the constraint. Condition ρ models whether (i) the action makes formula ϕ false (in the case of A_ϕ, line 10), (ii) the action makes formula ϕ true for the second time (in the case of AO_ϕ line 12), or (iii) the action makes formula ϕ true while $seen_\psi$ is false (in the case of $\mathsf{SB}_{\phi,\psi}$, line 15). The regressed condition ρ is negated and then conjoined with the precondition of the action. This way, if the action will violate the constraint in a given state, such an action is deemed inapplicable by the planner. In the case of ITCs, conditional effects are added to keep track of whether relevant formulae have ever held in the state trajectory of the plan prefix. For instance, $\mathsf{SB}_{\phi,\psi}$ requires to deal with the truth of $seen_\psi$: if an action makes ψ true, then the action must make $seen_\psi$ true too. This way, we can prevent to apply an action a when it makes ϕ true and ψ has not held before in current plan state trajectory (lines 15–18).

For each LTC $c \in C$, the algorithm yields a formula ρ that is true only in those states where the action achieves the targeted formula expressed in c. Note here the slightly different treatment for the two types of LTCs. While ST_ϕ only requires ϕ to be true, for $\mathsf{SA}_{\phi,\psi}$ we need to signal the necessity of ψ only when ϕ becomes satisfied; we do so by introducing two conditional effects (line 23 and 24) affecting the additional goal $hold_c$ of G' (line 29). Also observe that ϕ can become true multiple times, and each state satisfying ϕ needs to be followed by a state such that ψ is true again; this state can also be the same state in which ϕ holds, as prescribed by the semantics of PDDL3.0.

Note that Algorithm 1 can add irrelevant preconditions and conditional effects that can easily be omitted by looking at whether regression leaves a formula unaltered. E.g., for A_ϕ, if $\rho = R(\neg\phi, a) = \neg\phi$, there is no need to extend *Pre*(a) with $\neg\rho = \phi$ at line 18. Such optimizations are implemented but omitted here for clarity and compactness.

As trajectory constraints are monitored along the entire plan, and regression through effects provides sufficient conditions for ensuring that no ITC is violated by an action and no LTC remains unsatisfied at the plan end, it is easy to see that the compiled problem always finds a solution that conforms with the trajectory constraints of the problem. Moreover, since the exploited regression establishes necessary conditions too, the existence of a solution in the compiled problem implies that the original problem is solvable.

Theorem 1. *Let* $\Pi = \langle\langle F, I, A, G\rangle, C\rangle$ *be a* PDDL3.0 *planning problem, and let* $\Pi' = \langle F', I', A', G'\rangle$ *be the classical planning problem generated by Algorithm 1.* Π *admits a solution iff so does* Π'.

Experimental Results

Our experimental analysis compares the performance of planning using TCORE with other five state-of-the-art approaches handling trajectory constraints. Specifically, we consider the last available version of three native PDDL3.0 planners, i.e., SGPLAN (Hsu et al. 2007), OPTIC (Benton, Coles, and Coles 2012), MIPS-XXL (Edelkamp, Jabbar, and Nazih 2006), and two compilation-based approaches supporting LTL constraints over finite traces, i.e., the exponential (LTL-E) and the polynomial (LTL-P) compilation by Baier and McIlraith (2006) and Torres and Baier (2015), respectively. We will refer to MIPS-XXL simply as MIPS. TCORE is implemented in Python, and it can be downloaded from https://bit.ly/3sSbFCU. The compiled problems generated by TCORE, LTL-E and LTL-P were solved using LAMA (Richter and Westphal 2010).

Our benchmark suite involves domains from the 5th International Planning Competition (https://lpg.unibs.it/ipc-5/). Since the instances of the competition did not contain qualitative state-trajectory constraints, but only preferences, we generated a new set of instances as follows. For each instance with preferences, we ran some planners supporting preferences, i.e. LAMA and Mercury (Domshlak, Hoffmann, and Katz 2015) with the compiler by Percassi and Gerevini (2019), and LPRPG-P by Coles and Coles (2011), and we collected all plans generated within 30 CPU minutes.

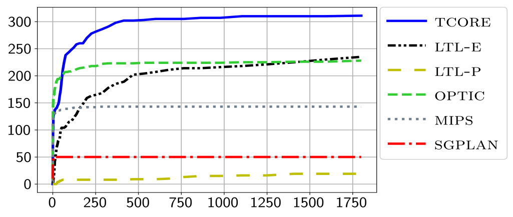

Figure 1: Coverage (y-axis) versus planning time (x-axis).

Out of this set, we took up to 5 plans (those with the larger number of satisfied preferences) and, for each of them, we generated a new problem instance having all satisfied preferences converted in qualitative state-trajectory constraints. This gave us a grand total of 416 instances: 79 for `Trucks`, 90 for `Openstack`, 55 for `Storage`, 94 for `Rover`, and 98 for `TPP`. For LTL-E and LTL-P, we used the most effective translation from PDDL3.0 to LTL among those provided by the tools, and by De Giacomo et al. (2014).

We measured the number of instances solved by each system in each domain (coverage) and the CPU time spent to find a solution, if any. For compilation-based approaches, CPU time is compilation time plus planning time. All experiments ran on an Xeon Gold 6140M 2.3 GHz, with time and memory limits of 1800s and 8GB, respectively.

Domain	TCORE	LTL-E	LTL-P	OPTIC	MIPS	SGPLAN
`Rover` (94)	**92**	35	0	33	11	0
`TPP` (98)	22	**58**	1	9	1	1
`Trucks` (79)	**78**	41	0	67	29	35
`Openstack` (90)	88	78	10	**89**	**89**	0
`Storage` (55)	**31**	23	8	30	13	14
Total (416)	**311**	235	19	228	143	50

Table 1: Coverage of the considered systems. In parenthesis, the number of benchmark instances for a given domain.

Coverage Analysis. Table 1 gives an overall picture of the coverage obtained by all systems. The system obtaining the highest coverage is TCORE, which solved more instances than the competitors in three out of 5 domains, and achieved a substantially higher total coverage. The performance of TCORE in `Rover` is remarkable; here TCORE solves three times more the instances solved by LTL-E (the second best performer). Yet, there seems to be some complementary between LTL-E and TCORE. Indeed, in `TPP`, LTL-E achieves the best coverage, overcoming TCORE by a substantial margin. Since TCORE is tailored for PDDL3.0, one could expect that it performs always better against both LTL-E and LTL-P. TCORE works on the ground representation of the problem, and this requires to ground actions upfront before compiling them. In `TPP`, TCORE did not manage to even trespass the grounding phase for several instances, causing a substantial drop in coverage. Instead, LTL-E works directly on the first-order representation, and this turned out to be very prolific. To gain more insights on the difference of performance between LTL-E and TCORE, we measured the number of atoms and effects generated by these two compilations. Our findings show that the grounded instances generated by TCORE contain always less atoms and effects than those generated by LTL-E; TCORE generates from 20.6% to 86.9% less atoms than LTL-E, and the difference of effects is from 1 to 3 orders of magnitude. Finally, we related performance with number and types of constraints (Fig. 2); the PDDL3.0 native systems (OPTIC, SGPLAN) provide poor performances over instances involving LTCs, but they perform better when the constraints are only ITCs (e.g., in `Openstack` for OPTIC).

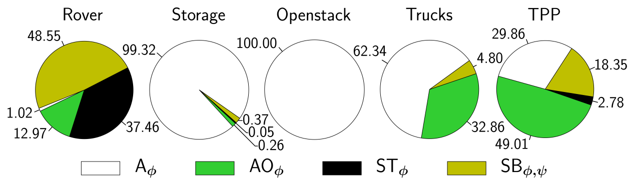

Figure 2: Constraints grouped by type across all domains.

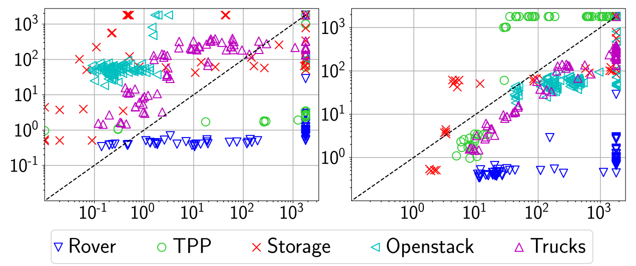

Figure 3: Left: TCORE (y-axis) vs OPTIC (x-axis); Right: TCORE (y-axis) vs LTL-E (x-axis). Points represent CPU seconds for the two systems over all instances.

CPU-time Analysis. Figure 1 analyzes the relationship between coverage and time allotted to each planner. The compilation-based approaches tend to increase coverage more slowly than the other approaches; this is due to the time that they spent for compilation. Native PDDL3.0 systems achieve high coverage in a handful of seconds, with SGPLAN exacerbating this trend to the point that its highest coverage is reached only after 4 seconds. Figure 3 shows a pairwise CPU-time comparison between the two best performing compilation-based methods, and between TCORE and OPTIC. TCORE is generally faster than LTL-E (which in turns dominates LTL-P), while it is slower than OPTIC due to the overhead caused by the compilation. Yet OPTIC exceeds the time limit for many instances (in `Rover`, `TPP` and `Trucks`) that are solved by TCORE.

Conclusions

We have presented a new method for compiling away the class of PDDL3 qualitative state-trajectory constraints. The compilation exploits the structure of the problem actions to obtain an encoding that is more efficient than those relying upon explicit constructions of automatons for the general class of LTL-based constraints. Experimental results show that planning through our method can achieve better performance also with respect to planners that natively support PDDL3. Future work includes investigating the compilation of trajectory constraints in metric-temporal domains.

Acknowledgments

We thank all the anonymous reviewers for their helpful comments.

References

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

Baier, J. A.; and McIlraith, S. A. 2006. Planning with First-Order Temporally Extended Goals using Heuristic Search. In *AAAI*, 788–795. AAAI Press.

Benton, J.; Coles, A. J.; and Coles, A. 2012. Temporal Planning with Preferences and Time-Dependent Continuous Costs. In *ICAPS*. AAAI.

Chen, Y.; Wah, B. W.; and Hsu, C. 2006. Temporal Planning using Subgoal Partitioning and Resolution in SGPlan. *J. Artif. Intell. Res.* 26: 323–369.

Coles, A. J.; and Coles, A. 2011. LPRPG-P: Relaxed Plan Heuristics for Planning with Preferences. In *ICAPS*. AAAI.

Domshlak, C.; Hoffmann, J.; and Katz, M. 2015. Red-black planning: A new systematic approach to partial delete relaxation. *Artif. Intell.* 221: 73–114. doi:10.1016/j.artint.2014.12.008. URL https://doi.org/10.1016/j.artint.2014.12.008.

Edelkamp, S. 2006. On the Compilation of Plan Constraints and Preferences. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling, ICAPS 2006*, 374–377.

Edelkamp, S.; Jabbar, S.; and Nazih, M. 2006. Large-scale optimal PDDL3 planning with MIPS-XXL. *5th International Planning Competition Booklet (IPC-2006)* 28–30.

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.

Giacomo, G. D.; Masellis, R. D.; and Montali, M. 2014. Reasoning on LTL on Finite Traces: Insensitivity to Infiniteness. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 1027–1033.

Giacomo, G. D.; and Vardi, M. Y. 2013. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *IJCAI*, 854–860. IJCAI/AAAI.

Hsu, C.; Wah, B. W.; Huang, R.; and Chen, Y. 2007. Constraint Partitioning for Solving Planning Problems with Trajectory Constraints and Goal Preferences. In *IJCAI*, 1924–1929.

Percassi, F.; and Gerevini, A. E. 2019. On Compiling Away PDDL3 Soft Trajectory Constraints without Using Automata. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2019*, 320–328.

Pnueli, A. 1977. The Temporal Logic of Programs. In *FOCS*, 46–57. IEEE Computer Society.

Richter, S.; and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39: 127–177.

Rintanen, J. 2008. Regression for Classical and Nondeterministic Planning. In *ECAI*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, 568–572. IOS Press.

Torres, J.; and Baier, J. A. 2015. Polynomial-Time Reformulations of LTL Temporally Extended Goals into Final-State Goals. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015*, 1696–1703.

Wright, B.; Mattmüller, R.; and Nebel, B. 2018. Compiling Away Soft Trajectory Constraints in Planning. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference, KR 2018*, 474–483. AAAI Press.