An Effective Polynomial Technique for Compiling Conditional Effects Away

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, United Kingdom
alfonso.gerevini@unibs.it,f.percassi@hud.ac.uk,enrico.scala@unibs.it

Abstract

The paper introduces a novel polynomial compilation technique for the sound and complete removal of conditional effects in classical planning problems. Similar to Nebel's polynomial compilation of conditional effects, our solution also decomposes each action with conditional effects into several simpler actions. However, it does so more effectively by exploiting the actual structure of the given conditional effects. We characterise such a structure using a directed graph and leverage it to significantly reduce the number of additional atoms required, thereby shortening the size of valid plans. Our experimental analysis indicates that this approach enables the effective use of polynomial compilations, offering benefits in terms of modularity and reusability of existing planners. It also demonstrates that a compilation-based approach can be more efficient, either independently or in synergy with state-of-the-art optimal planners that directly support conditional effects.

Introduction

In essence, automated planning is the problem of finding a sequence of actions to achieve certain objectives from an initial state. Many languages have been developed to model planning problems concisely, each with its strengths and weaknesses. This paper studies classical planning with conditional effects (Pednault 1989), which allows one to express state-dependent effects in the action model.

Over the years, conditional effects have proved quite useful to encode planning problems beyond classical planning, such as planning with soft and hard state trajectory constraints (e.g., Wright, Mattmüller, and Nebel 2018; Percassi and Gerevini 2019; Bonassi et al. 2021, 2023), planning under uncertainty (e.g., Palacios and Geffner 2009; Grastien and Scala 2017), matrix multiplication as planning (Speck et al. 2023) and even for compiling expressive planning formalisms like PDDL+ in numeric planning (Percassi, Scala, and Vallati 2023). Moreover, they exhibit a relationship with state-dependent action cost (Mattmüller et al. 2018; Speck et al. 2021) and numeric planning (Gigante and Scala 2023). However, despite conditional effects being introduced more than three decades ago, several planners do not directly support them, or they do so with some limitations. The majority of state-of-the-art planners are based on forward state-space heuristic search (e.g., Helmert 2006; Richter and Westphal 2010), with a few notable exceptions (e.g., Gerevini, Saetti, and Serina 2003; Wehrle and Rintanen 2007; Lipovetzky and Geffner 2017). Although there is a significant number of powerful heuristics available for these systems, only a few of them provide direct support for conditional effects with the necessary guarantees, such as the admissibility of heuristics.

One way to support conditional effects, independently from the search algorithm and heuristic of the planner, is using a compilation-based approach: the problem with conditional effects is transformed into an equivalent one without them, ensuring soundness and completeness. Compilation approaches have the nice property of allowing the use of a wide set of existing planners, providing a planner-independent approach. Two main compilation approaches have been investigated. Gazen and Knoblock (1997) proposed an approach that generates compiled actions emulating all possible contextual effects that may take place; Nebel (2000) devised a method simulating a machine that evaluates and applies conditional effects through a sequence of auxiliary actions. These approaches have different strengths and weaknesses: the former preserves the length of valid plans at the cost of an exponential worst-case blow-up in the size of the problem representation, while the second does not suffer this exponential blow-up but stretches the size of the valid plans by a polynomial factor.

In this paper, we investigate yet another polynomial compilation that addresses some of the limitations of Nebel's approach, resulting in a significantly more powerful technique. We begin by closely examining the interactions of conditional effects within each operator, and we observe that there are instances where it is possible to achieve a more concise encoding. We interpret these interactions by employing an *effect interference graph* for each action and utilising well-known algorithms for managing and reasoning about these dependencies. Our study leads to a two-step transformation. In the initial step, we process the planning problem to ensure that its interference graphs become acyclic. Then we exploit this acyclicity to generate an encoding where the additional auxiliary atoms for the compiled problem are limited to only those that are necessary, and all auxiliary actions do not in-

*Corresponding author.

troduce further branching points in the search of the planner.

The paper first presents our compilation formally and analyses its main theoretical properties. Then it reports on an extensive experimental analysis carried out using a variety of known planning benchmarks and planners. Note that, notwithstanding its worst-case exponential behaviour, Gazen and Knoblock's approach seems preferred over Nebel's polynomial approach (Hoffmann et al. 2006). Indeed, the compilation by Gazen and Knoblock does not jeopardise the properties of the heuristic (Katz 2019).

Our results show that exponential compilations may not always be a good idea. Indeed, over a large number of benchmarks with conditional effects collected from different sources, we observe that, for optimal planning, our new polynomial compilation behaves more effectively than both the ones by Gazen and Knoblock (1997) and Nebel (2000) in three out of four considered planning systems. Moreover, the proposed compilation is competitive and complementary with approaches handling conditional effects natively.

Preliminaries

This section provides a brief overview of the syntax and semantics of classical planning with conditional effects (CEs). We then give some notations and definitions that will be used for our compilation schemata.

A *classical planning problem* Π *with* CE*s* is the tuple $\langle F, A, I, G\rangle$, where F is a set of atoms, A is a set of actions, I is a set of atoms such that $I \subseteq F$, and G is a set of literals over F. Each action $a \in A$ is a pair $\langle \mathit{pre}(a), \mathit{post}(a)\rangle$, where $\mathit{pre}(a)$ is a set of literals over F, and $\mathit{post}(a)$ is a set of CEs. A CE e of an action a is a pair $\langle \mathit{cond}(e), \mathit{eff}(e)\rangle$, where $\mathit{cond}(e)$ and $\mathit{eff}(e)$ are both sets of literals over F. A *plan* π for a planning problem Π is a sequence $\langle a_1, \ldots, a_n\rangle$ of actions in A. The *cost of a plan* π is defined as $\mathit{cost}(\pi) = \sum_{a\in\pi} \mathit{cost}(a)$, where $\mathit{cost}(a)$ is a non-negative rational value defining the *cost of action* a. For convenience, let L be some set of literals, we superscript L with "$+$" or "$-$" to extract the positive and negative literals in a set, respectively. For example, if $L = \{p, \neg q\}$, then $L^+ = \{p\}$ and $L^- = \{q\}$. With $\mathit{atoms}(L)$ we access the set of atoms in L (e.g., $\mathit{atoms}(\{p, \neg q\}) = \{p, q\}$). Finally, we use the term *planning problem* as an abbreviation of classical planning problem with CEs.

The semantics of a planning problem $\Pi = \langle F, A, I, G\rangle$ is as follows. A state s assigns a truth value to each atom in F; we represent states as subsets of F using the closed-world assumption (if $f \in s$, then f is true in s); otherwise, f is false. The *state space* of Π is 2^F, and I is the initial state. Let L be a set of literals, $s \models L$ iff for all $l \in L^+$ ($z \in L^-$) we have that l (z) is true (false) in s. An action $a = \langle \mathit{pre}(a), \mathit{post}(a)\rangle$ of Π is *applicable* in a state s iff $s \models \mathit{pre}(a)$ and all active CEs of a do not conflict, i.e., $\bigcup_{e,e' \in \mathit{post}(a) \text{s.t.} e\neq e'} \{\mathit{eff}(e)^+ \cap \mathit{eff}(e')^- \mid s \models \mathit{cond}(e) \cup \mathit{cond}(e')\} = \emptyset$.

The application of an action a to a state s generates a successor state, denoted as $s[a]$, such that $f \in s[a]$ iff at least one of the following conditions holds: (i) in $\mathit{post}(a)$ there exists an effect $\langle C, L\rangle$ such that $f \in L^+$ and $s \models C$; (ii) $f \in s$ and in $\mathit{post}(a)$ there exists no effect $\langle C, L\rangle$ such that $p \in L^-$ and $s \models C$. In the following, we assume that each action has no conflicting CEs.

A plan $\pi = \langle a_1, \ldots, a_n\rangle$ is applicable in a state s_0 if a_1 is applicable in s_0 and, for $i \in \{1, \ldots, n\}$, every action a_i is applicable in $s[a_{i-1}]$; a plan π is a *solution* for Π if π is applicable in state s_I and $s[a_n] \models G$; Π is *solvable* if it admits a solution; an *optimal solution* for Π is a solution of Π that has minimal cost.

Definition 1 (Simple Action and Simple Planning Problem)**.** *Given a planning problem* $\Pi = \langle F, A, I, G\rangle$*, an action* $a \in A$ *is a* simple action *if, for every* CE $\langle C, L\rangle \in \mathit{post}(a)$*,* $C = \emptyset$*; a problem* Π *is a* simple planning problem *(SP) if all actions in* A *are simple.*

Compiling Conditional Effects Away

This section shows how to compile a classical planning problem with CEs into a simple planning problem. The compilation schema substitutes each action of the original problem into a number of simple actions that mimic the execution of the associated CEs. In order to do so in a way that preserves the semantics of the original action, this process takes into account the inter-dependencies that arise among CEs. We capture this formally through the notion of the *effect interference graph*, in turn, based on the notion of *effect interference*.

Definition 2 (Effect Interference)**.** *Let* e *and* e' *be two distinct* CE*s of an action* a*;* e interferes *with* e' *in* a *(written* $e \triangleright e'$*) iff* $\mathit{atoms}(\mathit{eff}(e)) \cap \mathit{atoms}(\mathit{cond}(e')) \neq \emptyset$*.*

Definition 3 (Effect Interference Graph)**.** *The* effect interference graph *of an action* a *is the directed graph* $\mathbb{G}_a = \langle V, E\rangle$ *where* $V = \{v_e \mid e \in \mathit{post}(a)\}$ *and* $E = \{(v_e, v_{e'}) \mid e, e' \in \mathit{post}(a), e \neq e',$ *and* $e \triangleright e'\}$*.*

Definition 4 (Effect Acyclic Action and Acyclic Planning Problem)**.** *An action* a *is* effect acyclic *iff* $\mathbb{G}_a$ *is acyclic; a planning problem is an* acyclic planning problem *(AP) iff all actions in* Π *are effect acyclic.*

Let $\mathbb{G}_a^T$ be the transpose graph of $\mathbb{G}_a$ obtained by reversing the direction of all its edges, it is easy to see that, if $\mathbb{G}_a$ is acyclic, a topological ordering $\omega = \langle v_{e_1}, \ldots, v_{e_m}\rangle$ of the vertices of $\mathbb{G}_a^T$ induces an *interference-free* sequence of effects $\langle e_1, \ldots, e_m\rangle$, i.e., no effect is conditioned by the outcome of an effect preceding it in ω. A problem that is not acyclic according to Definition 4 is denoted as ***CP***.

We present the compilation of CEs in a modular way. Firstly, we show how to compile CEs of an acyclic problem, denoted by τ_{SP}^{AP} (from ***AP*** to ***SP***). Then we show how any planning problem can be made acyclic (if needed) through another compilation called τ_{AP}^{CP} (from arbitrary classical planning problems with CEs ***CP*** to ***AP***). The whole pipeline τ_{AP}^{CP} plus τ_{SP}^{AP} can be used for compiling away any kind of CEs.

Transformation from Acyclic to Simple Planning Problems

As hinted at above, τ_{SP}^{AP} substitutes each action $a \in A$ with a number of fresh simple actions, linked together to mimic the original semantics of a. Two special actions, i.e., a^{start} and

a^{end}, mark the beginning and the end of the execution process for action a. The remaining ones, denoted as *intermediate actions*, are used to sequentially compute the successor state obtained by applying a. τ^{AP}_{SP} ensures that the start of a is followed by a sequence of sets of mutually exclusive simple actions, one for each CE $\langle C, L\rangle$ of a. Each set contains (i) a simple action that is executable when C holds and updates the state consistently with L, and (ii) another action for each literal in C that is executable when the corresponding literal does *not* hold in the state. Importantly, τ^{AP}_{SP} exploits the order of CEs given by $\mathbb{G}^T_a$. τ^{AP}_{SP} makes use of this ordering information to sequence all the intermediate actions in a way, that when an intermediate action has to be taken there is no previous intermediate action affecting its precondition.

The actions of the transformation are organised into four sets: A^{start} and A^{end} contain all starting and ending actions, while $A^{\text{CE}+}$ and $A^{\text{CE}-}$ contain all intermediate actions. Specifically, $A^{\text{CE}+}$ includes actions that mimic the activation of CEs and the subsequent state update. In contrast, $A^{\text{CE}-}$ are executed when the corresponding CEs do not get activated.

τ^{AP}_{SP} uses a number of fresh atoms: *pause* and, for each action a, $d_{i,a}$ for $i \in \{0, \ldots, |post(a)|\}$. The $d_{i,a}$-atoms mean that the i-th effect of a has been processed, and they are used to enforce an order for the simple actions associated with a specific action. The atom *pause* is used to avoid the interleaved execution of simple actions corresponding to different original actions and thus forbid their overlapping.

Given a planning problem $\Pi = \langle F, A, I, G\rangle$, τ^{AP}_{SP} translates it into the problem $\tau^{AP}_{SP}(\Pi) = \langle F', A', I, G'\rangle$ where

$$F' = F \cup \bigcup_{a\in A} \{d_{i,a} \mid i \in \{0, \ldots, |post(a)|\}\} \cup \{pause\}$$

$$A' = A^{start} \cup A^{end} \cup A^{\text{CE}+} \cup A^{\text{CE}-}$$

$$A^{start} = \bigcup_{a\in A} \{a^{start} = \langle pre(a) \cup \{\neg pause\}, \{\langle \emptyset, \{d_{0,a}, pause\}\rangle\}\rangle\}$$

$$A^{end} = \bigcup_{a\in A} \{a^{end} = \langle d_{|post(a)|,a}, \{\langle \emptyset, \{\neg d_{|post(a)|,a}, \neg pause\}\rangle\}\rangle\}$$

$$A^{\text{CE}+} = \bigcup_{a\in A} A^{\text{CE}+}_a \quad A^{\text{CE}-} = \bigcup_{a\in A} A^{\text{CE}-}_a \quad G' = G \cup \{\neg pause\}.$$

Referring to $A^{\text{CE}+}$ and $A^{\text{CE}-}$, consider an action $a \in A$ and let $\langle v_{e_1}, \ldots, v_{e_m}\rangle$ be a topological ordering of the m vertices of $\mathbb{G}^T_a$ (where v_{e_i} denotes a CE e_i of a). A^{ce+}_a has an action $a^+_{e_i}$ for each vertex v_{e_i} of $\mathbb{G}_a$ such that

$$pre(a^+_{e_i}) = cond(e_i) \cup \{d_{i-1,a}\}$$
$$post(a^+_{e_i}) = \{\langle \emptyset, eff(e_i) \cup \{d_{i,a}, \neg d_{i-1,a}\}\rangle\}.$$

$A^{\text{CE}-}_a$ contains the set of actions $\{a^l_{e_i} \mid l \in cond(e_i)\}$, one for each literal, such that v_{e_i} is a vertex of $\mathbb{G}_a$ and

$$pre(a^l_{e_i}) = \{d_{i-1,a}, \neg l\}$$
$$post(a^l_{e_i}) = \{\langle \emptyset, \{d_{i,a}, \neg d_{i-1,a}\}\rangle\}.$$

Each action in $A^{\text{CE}+} \cup A^{\text{CE}-} \cup A^{end}$ has cost 0, while each action in A^{start} has cost equal to $cost(a)$.

The presented compilation transforms actions with CEs and simple actions without distinction, even though the latter do not necessitate such processing. However, the compilation can be enhanced by ignoring the simple actions.

Theorem 1 (Problem Equivalence for τ^{AP}_{SP})**.** *Let* Π *be an acyclic planning problem.* Π *is solvable iff so is* $\tau^{AP}_{SP}(\Pi)$.

Proof. ($\tau^{AP}_{SP}(\Pi)$ is solvable $\Rightarrow$ Π is solvable) Let π' be any solution plan for $\tau^{AP}_{SP}(\Pi)$. By construction of $\tau^{AP}_{SP}(\Pi)$, π' is formed by m sub-sequences $\langle\langle a^1_1, \ldots, a^{k_1}_1\rangle, \ldots, \langle a^1_m, \ldots, a^{k_m}_m\rangle\rangle$ where each subsequence $\langle a^1_i, \ldots, a^{k_i}_i\rangle$, hereafter denoted as $\pi'(i)$, consists of simple actions generated by τ^{AP}_{SP}, referring to the original action $a_i \in A$. Specifically, $a^1_i = a^{start}_i \in A^{start}$, $a^{k_i}_i = a^{end}_i \in A^{end}$, while all the intermediate actions a^j_i, with $j \in \{2, \ldots, k_i - 1\}$, belong to $A^{\text{CE}+}_{a_i} \cup A^{\text{CE}-}_{a_i}$. Let $\pi = \langle a_1, \ldots, a_m\rangle$ be the sequence of actions in A obtained from π' by replacing each $\pi'(i)$ with the associated a_i. Moreover, let $\tau = \langle I = s_0, \ldots, s_m\rangle$ and $\tau' = \langle I' = s'_0, \ldots, s'_m\rangle$ be the state trajectories generated by iteratively applying π and π', respectively, where τ' is generated considering the sub-sequences of π' as if they were single actions. In the following, we prove that for each $i \in \{0, \ldots, m\}$, s_i and s'_i are equivalent w.r.t. F, writing this as $s_i \equiv_F s'_i$. The proof is by induction on i.

For the base case ($i = 0$), we observe that the atoms of $\tau^{AP}_{SP}(\Pi)$ extend those from Π, and $I = I'$, resulting in $s_0 \equiv_F s'_0$. For the inductive step, we assume the statement holds for any $i < |\tau|$ and prove it for $i + 1$. Since $s_i \equiv_F s'_i$ by the inductive hypothesis, and a^{start}_i is applicable in s'_i, we have that a_i is also applicable in s_i ($pre(a_i) \subset pre(a^{start}_i)$). In the continuation of the proof, we focus on the intermediate actions belonging to $A^{\text{CE}+}$ since those belonging to $A^{\text{CE}-}$ do not affect the atoms F. Let a^j_i with $j \in \{2, \ldots, k_i - 1\}$ be an intermediate action in $\pi'(i)$, associated to a CE $e \in post(a_i)$, that affects the state w.r.t. to F, i.e., $a^j_i = a^+_e \in A^{\text{CE}+}_{a_i}$, and let $F^e = atoms(cond(e))$. Let s'' be an intermediate state between s'_i and s'_{i+1} in which a^j_i is applied. Since all the intermediate actions in $\pi'(i)$ are properly sorted according to a topological ordering of $\mathbb{G}^T_{a_i}$ to obtain an interference-free sequence of actions, we know that $s'_i \equiv_{F^e} s''$. Moreover, the truth values of $atoms(eff(e))$ in $s''[a^j_i]$ persist until s'_{i+1} is reached, otherwise, there would be at least a pair of conflicting CEs in a_i. Since $s_i \equiv_F s'_i$ (inductive hypothesis) and $s'_i \equiv_{F^e} s''$, $s_i \equiv_{F^e} s''$. Therefore, the effect e of a_i is activated in s_i, modifying in s_{i+1} the truth values of atoms in F as a^j_i does in $s''[a^j_i]$. If we extend this reasoning to every intermediate actions in $\pi'(i)$, it follows that $s_{i+1} \equiv_F s'_{i+1}$.
(Π is solvable $\Rightarrow$ $\tau^{AP}_{SP}(\Pi)$ is solvable) Let $\pi = \langle a_1, \ldots, a_m\rangle$ be any solution plan for Π. From π, we can derive a solution π' for $\tau^{AP}_{SP}(\Pi)$ by replacing each action a_i with the subsequence $\langle a^1_i, \ldots, a^{k_i}_i\rangle$ of simple actions as defined earlier. We proceed as done for the opposite direction, i.e., considering τ and τ' generated by π and π', and showing their equivalence w.r.t. F, and that each state in τ' implies $\neg pause$.

For the case base, again, we know that $I = I'$, so $s_0 \equiv_F s'_0$ and moreover $s'_0 \models \neg pause$. For the inductive step, we assume that $s_i \equiv_F s'_i$ and $s'_i \models \neg pause$. Since π is a valid plan, we know that $s_i \models pre(a_i)$. It follows that $a^1_i = a^{start}_i$ is applicable as $pre(a^{start}_i) = pre(a_i) \cup \{\neg pause\}$. Then, to show that the remaining sequence converges to a

state s'_{i+1} that is equivalent to s_{i+1} w.r.t. F, we can use a similar argument as we did in the opposite direction. In particular, for every $e \in \mathit{post}(a_i)$ such that $s_i \models \mathit{cond}(e)$, in $\pi'(i)$ there is an intermediate action $a_i^j = a_e^+ \in A^{\text{CE}+}$ that produces the same effect as e does. Furthermore, for every $e \in \mathit{post}(a_i)$ such that $s_i \not\models \mathit{cond}(e)$, in $\pi'(i)$ there is an action $a_i^j = a_e^l \in A^{\text{CE}-}$ that has no effect on the F atoms. Finally, the last intermediate action of $\pi'(i)$ makes a_i^{end} the only applicable action, and this action makes *pause* false. It follows that $s'_{i+1} \equiv_F s_{i+1}$ and $s'_{i+1} \models \neg \mathit{pause}$. □

Transformation to Acyclic Planning Problems

The problem of finding the smallest set of vertices in a directed cyclic graph, whose removal makes the graph acyclic, is known as the problem of finding a minimal feedback vertex set (MFVS) (Younger 1963). In our context, where the graph vertices represent CEs, we reformulate MFVS as the problem of *finding a minimal feedback propositional set*.

Definition 5 (Minimal Feedback Propositional Set)**.** *A minimal feedback propositional set* (MFPS) *of an action a is the smallest set of atoms that appear in the conditions of the effects post(a) such that, if we remove from these conditions all literals involving these atoms, the effect interference graph $\mathbb{G}_a$ becomes acyclic.*

Let $a \in A$, and let $\text{MFPS}(a)$ denote a MFPS for a. The core idea of τ_{AP}^{CP} is to use $\text{MFPS}(a)$ to remove the cyclic effect interferences in a by (i) substituting, in the effects conditions of a, all literals involving atoms in $\text{MFPS}(a)$ with fresh atoms, and (ii) enforcing that such literals are true if and only if so are the original literals. To do so, the problem is extended with a *twin atom* t_p for each atom $p \in \text{MFPS}(a)$. The purpose of twin atoms is to track the truth value of the corresponding original atoms. The truth values of twin atoms related to action a are set by an additional *setup action* for a. The truth is determined by the truth of the atoms in the state where a is applied. Each revised action is then enforced to be executed immediately after its setup action, allowing safe evaluation of the CEs' conditions in the action.

More precisely, given a planning problem $\Pi = \langle F, A, I, G \rangle$, the transformation $\tau_{AP}^{CP}(\Pi) = \langle F', A', I, G' \rangle$ of Π is defined as follows:

$$F' = F \cup \bigcup_{a \in A} \{t_p \mid p \in \text{MFPS}(a)\} \cup \bigcup_{a \in A} \{\mathit{run}_a\} \cup \{\mathit{set}\}$$
$$A' = \{a^{\mathit{setup}} \mid a \in A\} \cup \{a^{\mathit{run}} \mid a \in A\}$$
$$G' = G \cup \{\neg \mathit{set}\} \cup \{\neg \mathit{run}_a \mid a \in A\}$$

where a^{setup} and a^{run} are defined as:

$$\mathit{pre}(a^{\mathit{setup}}) = \mathit{pre}(a) \cup \{\neg \mathit{set}\}$$
$$\mathit{eff}(a^{\mathit{setup}}) = \bigcup_{p \in \text{MFPS}(a)} \{\langle \{p\}, \{t_p\} \rangle, \langle \{\neg p\}, \{\neg t_p\} \rangle, \langle \emptyset, \{\mathit{run}_a, \mathit{set}\} \rangle\}$$
$$\mathit{pre}(a^{\mathit{run}}) = \{\mathit{run}_a\}$$
$$\mathit{eff}(a^{\mathit{run}}) = \bigcup_{e \in \mathit{post}(a)} \{\langle \mathcal{S}(\mathit{cond}(e), a), \mathit{eff}(e) \rangle\} \cup \{\langle \emptyset, \{\neg \mathit{run}_a, \neg \mathit{set}\} \rangle\}$$

where $\mathcal{S}(c, a)$ is a function substituting in a set of literals c each atom $p \in \text{MFPS}(a)$ with the twin atom t_p. That is, $\mathcal{S}(c, a) = \{\theta(l, a) \mid l \in c\}$, with

$$\theta(l, a) = \begin{cases} t_p & \text{if } l = p \in \text{MFPS}(a) \\ \neg t_p & \text{if } l = \neg p \text{ and } p \in \text{MFPS}(a) \\ l & \text{otherwise.} \end{cases}$$

To preserve the plan cost, τ_{AP}^{CP} assigns a cost of 0 to all actions $a \in A^{\mathit{run}}$, while actions in A^{setup} are given the same cost as the original domain actions from which they are generated.

τ_{AP}^{CP} generates a planning problem that is equivalent to the original one in terms of its solutions.

Theorem 2 (Problem Equivalence for τ_{AP}^{CP})**.** *Let Π be a classical problem with* CEs*. $\tau_{AP}^{CP}(\Pi)$ is solvable iff so is Π.*

Proof Sketch. Given a valid plan π for Π, we can derive a valid plan π' for $\tau_{SP}^{AP}(\Pi)$ by replacing each action $a \in A$ in π with the corresponding pair of actions $\langle a^{\mathit{setup}}, a^{\mathit{run}} \rangle$, where $a^{\mathit{setup}} \in A^{\mathit{setup}}$ and $a^{\mathit{run}} \in A^{\mathit{run}}$ in $\tau_{SP}^{AP}(\Pi)$. This is because, given a state, and by the construction of $\tau_{SP}^{AP}(\Pi)$, the application of $\langle a^{\mathit{setup}}, a^{\mathit{run}} \rangle$ is equivalent to applying a and yields the same outcome. More in detail, a^{setup} does not affect the atoms F and copies the truth values of a subset of F (possibly empty if $\mathbb{G}_a$ is acyclic), i.e., $\text{MFPS}(a)$, into their twin atoms $F^a = \{t_p \mid p \in \text{MFPS}(a)\}$. Additionally, a^{run} has the same CEs of a, except for a subset of them (possibly empty if $\mathbb{G}_a$ is acyclic), where $\mathit{cond}(e)$ is defined over the atoms F^a. Since the values of F^a are synchronised with F by a^{setup}, the application of a^{run} updates the state consistently with a.

Likewise, we can derive a plan π for Π from a valid plan π' for $\tau_{AP}^{CP}(\Pi)$ by replacing each pair of actions $\langle a^{\mathit{setup}}, a^{\mathit{run}} \rangle$ with their corresponding original action $a \in A$ from Π. To demonstrate the validity of π, we can use a similar argument as that used in the opposite direction. □

The full compilation of a cyclic planning problem Π into a classical planning problem Π' without CEs consists of chaining the two proposed transformations, i.e., $\Pi' = \tau_{SP}^{AP}(\tau_{AP}^{CP}(\Pi))$. By Theorems 1–2 and the relative proofs, it is easy to see that the following corollary also holds:

Corollary 1. *Let Π be a classical problem with* CEs*. $\tau_{SP}^{AP}(\tau_{AP}^{CP}(\Pi))$ is solvable iff so is Π.*

As finding a MFVS is NP-COMPLETE, so is finding a MFPS. This makes τ_{AP}^{CP} expensive when actions have many CEs. To overcome this issue, in what follows we over-approximate MFPS preserving Theorems 1-2. Figure 1 presents the algorithm, which works as follows: it iteratively removes vertices from the effect interference graph $\mathbb{G}_a$ of the input action a, until $\mathbb{G}_a$ becomes acyclic, and returns in output all atoms in the set of literals $\mathit{cond}(e)$ for every removed vertex v_e. At each iteration, one of the strongly connected components of $\mathbb{G}_a$ ($\text{SCCs}(\mathbb{G}_a)$) with at least two vertices is randomly selected, if available. [1] Then, from this component, we heuristically select a vertex that corresponds to a CE involved in the largest number of interferences while limiting the number of atoms that it contributes to the output

[1] $\text{SCCs}(\mathbb{G}_a)$ can be computed by Tarjan's algorithm in linear time w.r.t. the size of $\mathbb{G}_a$ (Tarjan 1972).

Algorithm 1: Computing a size-approximated MFPS.

Input: Action a
Output: A set of atoms

$R \leftarrow \emptyset$
$\mathbb{G}_a \leftarrow$ build the effect interference graph of a
while true **do**
  $T \leftarrow \{g \mid g = \langle V, E\rangle \in \text{SCCs}(\mathbb{G}_a) \text{ and } |V| > 1\}$
  if $T = \emptyset$ **then**
    break
  end if
  $g = \langle V, E\rangle \leftarrow$ a random element from T
  $u \leftarrow \arg\max_{v_e \in V} \frac{in(v_e)+out(v_e)}{|cond(e)|}$
  $R \leftarrow R \cup \{u\}$
  remove u from $\mathbb{G}_a$
end while
return *atoms*$(\bigcup_{v_e \in R} \{l \mid l \in cond(e)\})$

(approximated) MFPS. Formally, the selected vertex maximises the ratio between the sum of the ingoing ($in(v_e)$) and outgoing ($out(v_e)$) edges of the vertex and the number of literals in the condition of the effect that it represents ($cond(e)$).

Hereinafter, we will refer to the pipeline of τ_{AP}^{CP} and τ_{SP}^{AP} as COCOA (COmpiling COnditional effects Away).

Related Work

Planners support CEs through compilation (as in our proposal) or natively with a custom search. In the first case, two compilations have been proposed. The first method, referred to as GKCOMP, was introduced by Gazen and Knoblock (1997). It replaces each action with a set of actions, one for each possible subset of CEs. These actions ensure that only the associated CEs are activated. While relatively straightforward to implement, this method exhibits worst-case exponential behaviour due to the powerset size of CEs.

The second compilation was proposed by Nebel (2000), and here it will be indicated with NCOMP. This approach breaks down each action a with CEs into a set of actions that must be executed in a sequence determined by the state in which a is applied. This sequence comprises two phases. The first one is the *evaluation phase* that evaluates which CEs have been activated. In detail, for each $e \in post(a)$, the execution of exactly one of the following actions is enforced: one for when e is activated, and another one for when it is not. These actions do not immediately affect the state, and this task is delegated to a subsequent sequence of *copying actions* that modify the state based on which CEs were activated during the evaluation phase. While NCOMP overcomes the exponential worst-case complexity of GKCOMP, it increases the length of valid plans polynomially. Nebel (2000) uses this as a theoretical tool to study under which condition CEs can be compiled away.

Similar to NCOMP, our compilation approach (COCOA) breaks down each action into a set of simpler actions. However, it achieves this by analysing the inherent structure of the CEs. This analysis enables COCOA to generate a more concise encoding. Firstly, COCOA simultaneously evaluates and updates the affected atoms. In contrast, NCOMP decouples these phases, which significantly impacts the plan's size. Secondly, utilising the interference graph, COCOA produces simple actions ordered upfront. Therefore, unlike NCOMP, the actual permutation of these actions is not a choice of the planner. Moreover, COCOA adds only the necessary atoms to eliminate cycles, and the inclusion of atoms for enforcing a total order among actions has a more limited impact. Naturally, COCOA introduces an overhead during pre-processing. Indeed, we have noted that achieving optimal acyclicity for the CEs would require solving an NP-COMPLETE problem (Karp 1972). Our greedy algorithm approximates the optimal solution by employing a heuristic that selects the most promising arc for removal. This approach bears some similarity to the work by Lin and Jou (2000). We expect this reasoning to pay off against NCOMP's approach and will test it experimentally.

The concept of sequencing compiled actions to make more efficient compilation is not entirely new and it is often used as a part of the compilation outcomes (e.g., Ceriani and Gerevini 2015; Torres and Baier 2015); Hoffmann et al. (2006) already suggested to sequence the NCOMP actions.

State-of-the-art planners are mostly based on state-space heuristic search, and so, significant effort has been devoted to devising novel heuristic estimates sensitive to CEs. In optimal planning, efforts have been devoted to extend the LM-cut heuristic (Helmert and Domshlak 2009). Keyder, Hoffmann, and Haslum (2012) and Röger, Pommerening, and Helmert (2014) both contribute to this research direction. More general in scope, Haslum (2013) proposes an incremental compilation approach to solve optimal delete-free relaxation for problems with CEs. This approach first tries to solve a relaxed problem compiled through a simplified transformation inspired by NCOMP, and in case of failure, it performs an incremental exponential compilation of a subset of CEs, relaxing the remaining ones. This procedure degenerates, in the worst case, into the GKCOMP compilation.

Another attempt worth to be mentioned is the work by Katz (2019), which extended the red-black planning framework (Domshlak, Hoffmann, and Katz 2015) to support problems with CEs. The author generalises the definition of invertibility of a variable, showing that red-black planning for tasks with DAG black causal graphs remains a tractable task even if they have CEs. This generalisation allows one to derive informed (yet non-admissible) heuristics.

Native approaches for handling CEs do not alter the problems' size but they only work within specific heuristic frameworks. This likely contributes to the limited native CEs support in modern planners. As a result, progress towards efficiently managing CEs has not kept pace with the advancements seen in classical planners. We remark that compilation approaches also serve as an inspiration for crafting relaxation-specific heuristics (Haslum 2013; Röger, Pommerening, and Helmert 2014) and, similarly, we argue that our compilation method can inspire newer relaxations/heuristics handling directly CEs, too.

Experimental Analysis

We compared our compilation (COCOA) with those provided by Gazen and Knoblock (1997) (GKCOMP) and Nebel (2000) (NCOMP). (**Source code**: https://gitlab.com/EdmondDantes/cocoa2.0.) We conducted experiments with planning problems generated from these compilations in the optimal setting because the methodologies for handling CEs in planners are less explored and established in this context. Therefore, we believe that COCOA can play a significant role in this scenario.

Specifically, we considered two classes of optimal planners. The first class comprises A* (Hart, Nilsson, and Raphael 1968) run either with $h^{\text{LM-cut}}$ (Helmert and Domshlak 2009), i.e., A$^\star(h^{\text{LM-cut}})$, or with the $h^{\max}$ heuristic (Haslum and Geffner 2000), i.e., A$^\star(h^{\max})$. The second class involves two relevant planners from the Ninth International Planning Competition (IPC-18), i.e., COMPLEMENTARY2 (Franco et al. 2018) based on symbolic PDBs (Franco et al. 2017), and METIS2 (Alkhazraji et al. 2014; Sievers and Katz 2018). The use of the first class of planners aims at measuring the impact of the compilation on search with precision, minimising any external noise stemming from the specific configuration of the system. The use of the second class of planners is motivated by the intent of assessing how much a fully equipped system can be influenced by the chosen compilation schema. Specifically, COMPLEMENTARY2 is the fastest non-portfolio planner from the IPC-18, while METIS2 is selected primarily for its ability to exploit symmetries in planning (Domshlak, Katz, and Shleyfman 2012). This capability allows METIS2 to handle Nebel's compilation more efficiently than other planners, as NCOMP introduces numerous symmetries in the compiled problem. All planners used in our experiments are based on the Fast Downward planning system (Helmert 2006). Moreover, for the sake of evaluation, we also compared all compilations against planners supporting CEs natively, denoted with PLAIN in the problem formulation. For PLAIN and for the first class of planners, we use the standard $h^{\max}$ implementation that already supports CEs and the CEs-aware variant of $h^{\text{LM-cut}}$ (Röger, Pommerening, and Helmert 2014). We also did our best to implement NCOMP as faithfully as possible to what was proposed by Nebel (2000). (Note that, ADL2STRIPS's current implementation of NCOMP only works for non-interfering CEs.)

Benchmarks. We collected domains with CEs from different sources: Fast Downward benchmark collection (https://github.com/aibasel/downward-benchmarks), problems generated by conformant-to-classical planning compilations (Palacios and Geffner 2009; Grastien and Scala 2017). Additionally, we also included domains used in the work by Röger, Pommerening, and Helmert (2014). From the collected benchmarks we excluded domains with conflicting CEs. This is the case for all domains deriving from the finite-state controller synthesis (Bonet, Palacios, and Geffner 2009). In our experiments, each test run is specified by the combination of a planner, a compilation method, and a planning problem from our benchmarks suite. With PLAIN, for instance, the compilation time is zero, whereas with COCOA, GKCOMP and NCOMP, compilation time depends on the specific instance and may not be negligible at all. Each run firstly compiles the problem and then gives the generated output to the desired off-the-shelf planner. We give a budget of 1800 seconds of runtime, 8 GB of memory for each run (compilation plus solving) and 2 GB of disk space. The experiments were run on an Intel Xeon Gold 6140M CPU with 2.30 GHz.

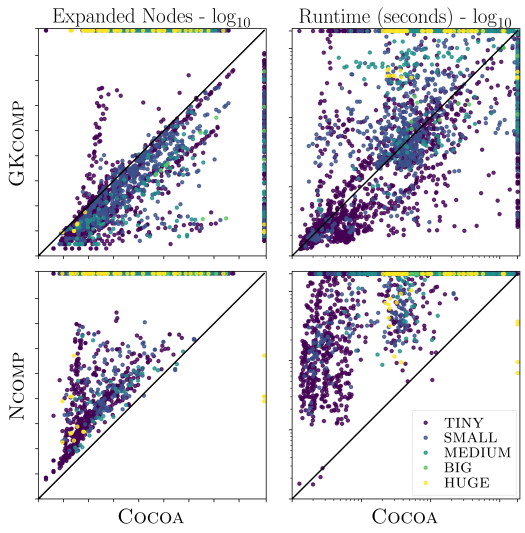

Figure 1: Expanded nodes (left) and Runtime (right) for COCOA vs GKCOMP (top) and COCOA vs NCOMP (bottom).

Results

Table 1 shows the coverage obtained by each system with a given compilation. Together with this information, we also present the coverage for the Virtual Best Solver, both considering an individual planner (LVBS, where L stands for local) and all systems (VBS). The *fail* row shows the number of problems that did not pass Fast Downward's preprocessing.

Comparison Among Compilations. Observing Table 1, it is evident that COCOA consistently achieves more coverage than GKCOMP and NCOMP, except for COMPLEMENTARY2, where GKCOMP behaves better. The overall best configuration is COCOA with METIS2.

Figure 1 shows the number of expanded nodes and runtime comparing GKCOMP vs COCOA (upper), and NCOMP vs COCOA (lower). The scatter plots encompass all planners, with points grouped based on the average number of CEs per operator, denoted as n_{avg}. The buckets are: TINY ($n_{\text{avg}} \leq 2$), SMALL ($2 < n_{\text{avg}} \leq 4$), MEDIUM ($4 < n_{\text{avg}} \leq 6$), BIG ($6 < n_{\text{avg}} \leq 8$), and HUGE ($n_{\text{avg}} > 8$). The general trend is that GKCOMP expands fewer nodes than COCOA. By a closer examination of our raw experimental data, we note that the points above the bisector are always associated with $h^{\text{LM-cut}}$. This result is not surprising, considering that GKCOMP preserves the plan length (Nebel 2000), while COCOA extends it by a polynomial factor. Additionally, despite GKCOMP leading to the expansion of fewer nodes, the run-

	A* Search										Planning System										BVS
	A*($h^{\max}$)					A*($h^{\text{LM-cut}}$)					COMPLEMENTARY2					METIS2					
	C	N	G	P	BVLS	C	N	G	P	BVLS	C	N	G	P	BVLS	C	N	G	P	BVLS	
Σ (1608)	529	132	478	**650**	650	742	212	541	**828**	892	617	101	649	**703**	825	**860**	316	687	596	971	1045
fail	0	0	0	0	—	15	0	0	339	—	0	0	0	0	—	0	0	0	341	—	—

Table 1: Coverage for all planners under different problem formulations. "C" stands for COCOA, "N" for NCOMP, "G" for GKCOMP and "P" for "PLAIN". Bold is for best in a given planning system, and the underlined bold is the best across all planning systems and problem formulations. *fail* denotes the number of problems that did not pass Fast Downward's preprocessing.

Planner	PF	COCOA	NCOMP	GKCOMP	PLAIN
A*($h^{\max}$)	COCOA	—	397	89	0
	NCOMP	0	—	3	0
	GKCOMP	38	349	—	0
	PLAIN	123	520	174	—
A*($h^{\text{LM-cut}}$)	COCOA	—	534	252	40
	NCOMP	4	—	3	23
	GKCOMP	51	332	—	52
	PLAIN	128	641	341	—
COMPL2	COCOA	—	516	81	72
	NCOMP	0	—	1	15
	GKCOMP	113	549	—	103
	PLAIN	159	618	158	—
METIS2	COCOA	—	544	256	319
	NCOMP	0	—	28	74
	GKCOMP	83	399	—	138
	PLAIN	55	354	47	—

Table 2: Complementarity analysis. Given a planner, entry $\langle x, y \rangle$ denotes the problems solved by x (row) not solved by y (column). PF stands for Problem Formulation.

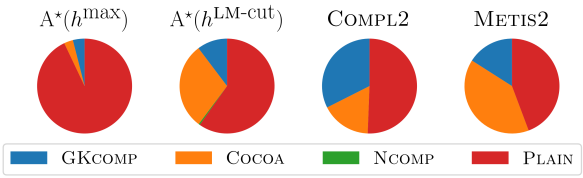

Figure 2: Percentage of times one formulation was faster than all others for each BVLS of a given planner.

time is significantly affected, especially with an increasing number of CEs (bearing in mind that GKCOMP is exponential in the number of CEs). Consequently, as the number of CEs increases, points tend to shift above the bisector, indicating a preference for COCOA. In contrast, GKCOMP tends to exhibit faster performance on smaller instances.

COCOA consistently outperforms NCOMP; this is somewhat expected since NCOMP was primarily designed for theoretical purposes, while COCOA aims to provide efficient compilation. Instances generated by NCOMP are, on average, 6.2 times larger than those generated by COCOA in terms of the number of actions (min. and max. increases of 1.9 and 19.7, respectively), and 4.1 times larger in terms of atoms (min. and max. increases of 2.9 and 8.9, respectively).

Comparison with Native Support. Table 1 shows that the native approach is generally faster. Conversely, METIS2 performs better with COCOA than with the original formulation, achieving performance slightly better than that of $h^{\text{LM-cut}}$.

To gain a deeper insight into the impact of different configurations, we investigated the relationship between compilation and native approaches. Our primary objective was to ascertain whether the systems under consideration solve distinct problem sets that do not overlap, or if one method dominates the others. We did so with a pairwise comparison among all systems, and by measuring, for each planner, which formulation enables finding a solution more quickly.

Table 2 summarises the pairwise comparison as follows. Each $\langle x, y \rangle$ entry of the table indicates the number of problems solved by x (row) that have not been solved by y (column). By examining the PLAIN column for each planner, we can evaluate the extent of complementarity between the compilation methods and the native approaches. Notably, in the case of $h^{\max}$, no complementarity is observed, indicating a clear dominance of the native approach. Conversely, for $h^{\text{LM-cut}}$ and COMPLEMENTARY2, we observe a moderate level of complementarity. A strong level of complementarity is instead observed for METIS2, in particular when comparing COCOA and PLAIN. Figure 2 shows, for each BVLS obtained for a specific planner, the percentage of times each formulation allows us to find a solution faster than all the others. A*($h^{\max}$) and A*($h^{\text{LM-cut}}$) are mainly dominated by PLAIN, and in the first case, dominance is almost total. Globally, as the complexity of the planning system grows, it can be observed that COCOA and GKCOMP provide a significant contribution to achieve better performance, especially for METIS2. NCOMP's contribution is instead negligible.

Conclusions

We have presented a polynomial technique for compiling CEs away in classical planning problems. The compilation is inspired by an early idea by Nebel (2000), but extends it substantially starting from the observation that CEs exhibits a structure which can be characterised using a directed graph. We have shown how to exploit such a structure obtaining a much more effective compilation. Specifically, our compilation removes some branching points during the search whilst preserving soundness and completeness. Moreover, in practice, the new compilation requires fewer actions and atoms. These improvements make the search much lighter. An experimental analysis demonstrates the practical advantages of the proposed technique, which, overall, unlocks the use of optimal planning engines over a much larger class of instances, exhibiting performances on par and complementary with engines supporting CEs directly during the search.

Acknowledgements

We thank the reviewers for their helpful suggestions to improve the paper. This work was partially supported by the EU H2020 project AIPlan4EU (No. 101016442), the EU ICT-48 2020 project TAILOR (No. 952215), and the PRIN project RIPER (No. 20203FFYLK). Francesco Percassi was supported by a UKRI Future Leaders Fellowship [grant number MR/T041196/1].

References

Alkhazraji, Y.; Katz, M.; Matmüller, R.; Pommerening, F.; Shleyfman, A.; and Wehrle, M. 2014. Metis: Arming Fast Downward with Pruning and Incremental Computation. *IPC 2014 Planner Abstracts*, 88–92.

Bonassi, L.; De Giacomo, G.; Favorito, M.; Fuggitti, F.; Gerevini, A. E.; and Scala, E. 2023. Planning for Temporally Extended Goals in Pure-Past Linear Temporal Logic. In *Proc. of ICAPS*, 1, 61–69.

Bonassi, L.; Gerevini, A. E.; Percassi, F.; and Scala, E. 2021. On Planning with Qualitative State-Trajectory Constraints in PDDL3 by Compiling them Away. In *Proc. of ICAPS*, 46–50.

Bonet, B.; Palacios, H.; and Geffner, H. 2009. Automatic Derivation of Memoryless Policies and Finite-State Controllers Using Classical Planners. In *Proc. of ICAPS*, 34–41.

Ceriani, L.; and Gerevini, A. E. 2015. Planning with Always Preferences by Compilation into STRIPS with Action Costs. In *Proc. of SoCS*, 161–165.

Domshlak, C.; Hoffmann, J.; and Katz, M. 2015. Red-black planning: A new systematic approach to partial delete relaxation. *Artif. Intell.*, 221: 73–114.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2012. Enhanced Symmetry Breaking in Cost-Optimal Planning as Forward Search. In *Proc. of ICAPS*, 343–347.

Franco, S.; Lelis, L. H.; Barley, M.; Edelkamp, S.; Martines, M.; and Moraru, I. 2018. The Complementary2 Planner in the IPC 2018. *IPC 2018 Planner Abstracts*, 28–31.

Franco, S.; Torralba, A.; Lelis, L. H.; and Barley, M. 2017. On Creating Complementary Pattern Databases. In *Proc. of IJCAI*, 4302–4309.

Gazen, B. C.; and Knoblock, C. A. 1997. Combining the Expressivity of UCPOP with the Efficiency of Graphplan. In *Proc. of ECP*, 221–233.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning Through Stochastic Local Search and Temporal Action Graphs in LPG. *J. Artif. Intell. Res.*, 20: 239–290.

Gigante, N.; and Scala, E. 2023. On the Compilability of Bounded Numeric Planning. In *Proc. of IJCAI*, 5341–5349.

Grastien, A.; and Scala, E. 2017. Intelligent Belief State Sampling for Conformant Planning. In *Proc. of IJCAI*, 4317–4323.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. Syst. Sci. Cybern.*, 4(2): 100–107.

Haslum, P. 2013. Optimal Delete-Relaxed (and Semi-Relaxed) Planning with Conditional Effects. In *Proc. of IJCAI 2013*, 2291–2297.

Haslum, P.; and Geffner, H. 2000. Admissible Heuristics for Optimal Planning. In *Proc. of AIPS*, 140–149. AAAI.

Helmert, M. 2006. The Fast Downward Planning System. *J. Artif. Intell. Res.*, 26: 191–246.

Helmert, M.; and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In *Proc. of ICAPS*, 162–169.

Hoffmann, J.; Edelkamp, S.; Thiébaux, S.; Englert, R.; dos S. Liporace, F.; and Trüg, S. 2006. Engineering Benchmarks for Planning: the Domains Used in the Deterministic Part of IPC-4. *J. Artif. Intell. Res.*, 26: 453–541.

Karp, R. M. 1972. Reducibility Among Combinatorial Problems. In *Proc. of CCC*, 85–103.

Katz, M. 2019. Red-Black Heuristics for Planning Tasks with Conditional Effects. In *Proc. of AAAI*, 7619–7626.

Keyder, E. R.; Hoffmann, J.; and Haslum, P. 2012. Semi-Relaxed Plan Heuristics. In *Proc. of AAAI*, 128–136.

Lin, H.; and Jou, J. 2000. On Computing the Minimum Feedback Vertex Set of a Directed Graph by Contraction Operations. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 19(3): 295–307.

Lipovetzky, N.; and Geffner, H. 2017. Best-First Width Search: Exploration and Exploitation in Classical Planning. In *Proc. of AAAI*, 3590–3596.

Mattmüller, R.; Geißer, F.; Wright, B.; and Nebel, B. 2018. On the Relationship Between State-Dependent Action Costs and Conditional Effects in Planning. In *Proc. of AAAI*, 6237–6245.

Nebel, B. 2000. On the Compilability and Expressive Power of Propositional Planning Formalisms. *J. Artif. Intell. Res.*, 12: 271–315.

Palacios, H.; and Geffner, H. 2009. Compiling Uncertainty Away in Conformant Planning Problems with Bounded Width. *J. Artif. Intell. Res.*, 35: 623–675.

Pednault, E. P. D. 1989. ADL: Exploring the Middle Ground Between STRIPS and the Situation Calculus. In *Proc. of KR*, 324–332.

Percassi, F.; and Gerevini, A. E. 2019. On Compiling Away PDDL3 Soft Trajectory Constraints without Using Automata. In *Proc. of ICAPS*, 320–328.

Percassi, F.; Scala, E.; and Vallati, M. 2023. A Practical Approach to Discretised PDDL+ Problems by Translation to Numeric Planning. *J. Artif. Intell. Res.*, 76: 115–162.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *J. Artif. Intell. Res.*, 39: 127–177.

Röger, G.; Pommerening, F.; and Helmert, M. 2014. Optimal Planning in the Presence of Conditional Effects: Extending LM-Cut with Context Splitting. In *Proc. of ECAI*, 765–770.

Sievers, S.; and Katz, M. 2018. Metis 2018. *IPC 2018 Planner Abstracts*, 83–84.

Speck, D.; Borukhson, D.; Mattmüller, R.; and Nebel, B. 2021. On the Compilability and Expressive Power of State-Dependent Action Costs. In *Proc. of ICAPS*, 358–366.

Speck, D.; Höft, P.; Gnad, D.; and Seipp, J. 2023. Finding Matrix Multiplication Algorithms with Classical Planning. In *Proc. of ICAPS*, 411–416.

Tarjan, R. 1972. Depth-first Search and Linear Graph Algorithms. *SIAM J. Comput.*, 1(2): 146–160.

Torres, J.; and Baier, J. A. 2015. Polynomial-Time Reformulations of LTL Temporally Extended Goals into Final-State Goals. In *Proc. of IJCAI*, 1696–1703.

Wehrle, M.; and Rintanen, J. 2007. Planning as Satisfiability with Relaxed $-Step Plans. In *Australian Conference on Artificial Intelligence*, volume 4830, 244–253.

Wright, B.; Mattmüller, R.; and Nebel, B. 2018. Compiling Away Soft Trajectory Constraints in Planning. In *Proc. of KR*, 474–483.

Younger, D. 1963. Minimum Feedback Arc Sets for a Directed Graph. *IEEE Transactions on Circuit Theory*, 10(2): 238–245.