Optimised Variants of Polynomial Compilation for Conditional Effects in Classical Planning

Francesco Percassi[1*], **Enrico Scala**[2], **Alfonso Emilio Gerevini**[2]

[1]School of Computing and Engineering, University of Huddersfield, United Kingdom
[2]Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Brescia, Italy
f.percassi@hud.ac.uk,enrico.scala@unibs.it,alfonso.gerevini@unibs.it

Abstract

Conditional effects are a key feature in classical planning, enabling the description of actions whose outcomes are state-dependent. It is well known that removing conditional effects in a polynomial way necessarily increases the size of a valid plan by a polynomial factor. However, preserving the exact plan size requires encoding the problem exponentially.

The paper proposes and empirically evaluates optimisations for existing polynomial compilations. These optimisations aim to make the resulting compilations more suitable for planners while limiting the increase in plan size, which is inevitable if we want to keep the compilation polynomial. Specifically, the paper introduces a polynomial compilation technique that expands conditional effects when their number is below a certain threshold and sequentialises them otherwise. Additionally, the paper demonstrates that even straightforward optimisations can have a notable impact.

Introduction

Automated planning involves generating sequences of actions to achieve predefined goals, starting from an initial state and based on a specified model. Over time, various planning formalisms have been developed to represent planning problems concisely and improve modelling capabilities. This study focuses on classical planning with conditional effects (Pednault 1989), a feature allowing actions to have state-dependent outcomes. Intuitively, a conditional effect is an action's effect that occurs if an additional condition is satisfied when the action is applied. Listing 1 provides a simple example of a PDDL action using conditional effects. This action allows the movement of a rover from one location to another. Additionally, if the rover carries a tool, it will be contextually moved.

Listing 1: PDDL action with a conditional effect.

```
(:action move_rover
 :parameters (?r - rover ?l1 ?l2 - loc ?t - tool)
 :precondition (at ?r ?l1)
 :effect (and (not (at ?r ?l1)) (at ?r ?l2)
   (when
      (and (carrying ?r ?t) (at ?t ?l1))
      (and (not (at ?t ?l1)) (at ?t ?l2)))))
```

Conditional effects have proven to be an expressive language feature with significant modelling capabilities. Indeed, they have been widely employed in various works to encode planning problems beyond the classical paradigm. These studies include planning subject to soft and hard state-trajectory constraints (Wright, Mattmüller, and Nebel 2018; Percassi and Gerevini 2019; Bonassi et al. 2021, 2023), planning under uncertainty (Palacios and Geffner 2009; Grastien and Scala 2017), deriving finite-state controllers (Bonet, Palacios, and Geffner 2009) and matrix multiplication as planning (Speck et al. 2023). From a theoretical perspective, conditional effects exhibit interesting properties, as they are related to state-dependent action costs and numeric planning (Mattmüller et al. 2018; Gigante and Scala 2023).

Notably, a wide range of well-performing planners rely on forward state-space heuristic search (e.g., Helmert 2006; Richter and Westphal 2010; Katz and Hoffmann 2014), with some notable exceptions (e.g., Gerevini, Saetti, and Serina 2003; Wehrle and Rintanen 2007; Lipovetzky and Geffner 2017; Speck, Mattmüller, and Nebel 2020). For this reason, extensive efforts have been undertaken to enable existing heuristics to support conditional effects (Keyder, Hoffmann, and Haslum 2012; Röger, Pommerening, and Helmert 2014; Domshlak, Hoffmann, and Katz 2015).

An appealing method for handling conditional effects is *compilation*. In this approach, the problem with conditional effects is transformed into an equivalent one without them, enabling the usage of several planning systems designed for simpler formalisms. However, there is a drawback to consider. According to Nebel (2000), the polynomial compilation of conditional effects leads to a polynomial increase in the plan size, negatively impacting planning systems performance, especially those based on heuristic search. As a side-effect of this finding, Nebel (2000) provides an upper-bound polynomial compilation. Conversely, it is possible to exactly preserve the plan size when transitioning from one formalism to another, albeit resulting in an exponential increase in the resulting problem (Gazen and Knoblock 1997). Between these two extremes, Gerevini, Percassi, and Scala (2024) recently introduced a polynomial compilation, COCOA, which leverages the interferences among conditional effects to reduce the plan size increase.

[*]Corresponding author.

In this article, we propose and empirically evaluate some optimisations of existing polynomial compilations of conditional effects to enhance their practical usage. As a main contribution, we introduce a new hybrid polynomial compilation method, leveraging the strengths of both COCOA and the exponential scheme while addressing their drawbacks. Furthermore, we outline some straightforward optimisations for COCOA to illustrate how specific design choices can significantly impact the efficiency of solving the compiled problem. Lastly, we propose a practical variant of Nebel's compilation to establish a more robust baseline.

Preliminaries

This section provides an overview of the syntax and semantics of classical planning with conditional effects (CEs). Furthermore, it presents the existing polynomial compilations.

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. If $\mathit{cond}(e) = \emptyset$, e is called *simple effect*. If an action a has only simple effects, it is also termed *simple action*. A *plan* π for a planning problem Π is a sequence $\langle a_1, \dots, a_n\rangle$ 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., $\mathit{atoms}(\{p, \neg q\}) = \{p, q\}$).

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, i.e., 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 $f \in L^+$ ($f' \in L^-$) we have that f (f') 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 $f \in L^-$ and $s \models C$. In the following, we assume that each action has no conflicting CEs.

A plan $\pi = \langle a_1, \dots, a_n\rangle$ is applicable in a state s_0 if a_1 is applicable in s_0 and, for $i \in \{1, \dots, n\}$, 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.

Nebel's Compilation (NCOMP)

The polynomial compilation proposed by Nebel (2000), hereafter referred to as NCOMP, prevents the exponential blow-up of the encoding by sequencing the execution of the CEs of an action instead of evaluating them in parallel. Before delving into details, it is important to recall that NCOMP expects that the effects of CEs have singleton structure, i.e., $\langle C, \{l\}\rangle$, where l is f or $\neg f$ and $f \in F$. Moreover, given an action a, $\kappa_a = |\mathit{post}(a)|$ denotes its number of CEs.

Given an action a_i, NCOMP generates a set of simple actions which are structured to be executed as follows:[1]

$$\langle a_i^{\mathit{pre}}\rangle \oplus \overbrace{\mathit{eval\text{-}phase}}^{\kappa_{a_i}\text{ actions}} \oplus \langle a_i^e\rangle \oplus \overbrace{\mathit{exec\text{-}phase}}^{\kappa'\in\{0,\dots,\kappa_{a_i}\}\text{ actions}} \oplus \langle a^c\rangle. \quad (1)$$

Referring to the previous sequence, a_i^{pre} initialises the execution of the action and can be applied if the preconditions of a_i are satisfied. The *eval-phase* sequence, which stands for *evaluation phase*, includes κ_{a_i} actions, one for each CE of a_i, aiming to evaluate which CE have been activated when a_i^{pre} is applied. Specifically, for each activated CE, the compilation tracks which atoms must be modified and how (added or deleted). Once all CEs have been evaluated, a_i^e is executed, initiating the *exec-phase* (*executive phase*). During this phase, the original problem atoms are added or deleted to the state based on the outcome of the evaluation phase. Specifically, for each atom $f \in F$, there are three possible actions: a_+^f (to add f), a_-^f (to delete f), and $a_\#^f$ (to lead to a dead-end when some conflicting CEs are activated together). It is worth noting that *exec-phase* encompasses a variable number of actions ranging in $\kappa' \in \{0, \dots, \kappa_{a_i}\}$, whose length depends on which atoms are affected by the entailed CEs. If no CEs are activated during the evaluation phase, then $\kappa' = 0$, while if all atoms are involved, then $\kappa' = \kappa_{a_i}$. Once all necessary state modifications have been carried out, the execution of the action is concluded with a^c.

COCOA Compilation

COCOA also sequences the execution of the CEs. However, the key difference with NCOMP lies in the fact that COCOA looks at the structure of the CEs for generating a more compact encoding and inducing shorter sequences of actions in which the evaluation and executive phases are blended.

Before illustrating the compilation, let us introduce some key concepts. A CE e *interferes* with another one e' (written $e \rhd e'$) iff $\mathit{atoms}(\mathit{eff}(e)) \cap \mathit{atoms}(\mathit{cond}(e')) \neq \emptyset$. Given an action a, the *effect interference graph* is defined as $\mathbb{G}_a = \langle V, E\rangle$ where $V = \{v_e \mid e \in \mathit{post}(a)\}$ and $E = \{(v_e, v_{e'}) \mid e, e' \in \mathit{post}(a), e \neq e', \text{ and } e \rhd e'\}$.

Given an action a, COCOA constructs the effect interference graph $\mathbb{G}_a$, and by reversing its edges, obtains the transpose graph $\mathbb{G}_a^T$. If $\mathbb{G}_a^T$ is acyclic, any of its topological orderings will produce an interference-free sequence of effects. This sequence can be used to generate pairs of mutually exclusive actions: one executed when the corresponding CE is activated, which modifies the state, and the other executed

[1]We use $\oplus$ for representing the chaining of sequences of actions, e.g., $\langle a_1\rangle \oplus \langle a_2, a_3\rangle = \langle a_1, a_2, a_3\rangle$

when it is not, acting transparently. The interference-free nature of this sequence preserves the parallel semantics of CEs. This contrasts with the NCOMP approach, which achieved this by decoupling the evaluation and execution of CEs.

However, if $\mathbb{G}_a^T$ contains cycles, a preprocessing step is needed to make it acyclic. This involves identifying the smallest set of atoms such that their removal yields an acyclic graph. This set is denoted as the *minimum feedback propositional set* of a (MFPS(a)), and it is computed using a sub-optimal approach. All atoms in MFPS(a) are added to the compiled problem as *twin atoms*. Next, the action a is divided into two sequential actions. The first one, a^{setup}, copies all atoms from MFPS(a) to their respective twins. The second one, i.e., a^{run}, includes all the original CEs but with any atom from MFPS(a) appearing in the condition of a CE replaced by its respective twin atom. This separation prevents interference between CEs.

These two steps of the compilation are handled modularly in COCOA. Specifically, given a planning problem with *cyclic* CEs Π, the module τ_{AP}^{CP} produces a problem with *acyclic* CEs, denoted as $\Pi' = \tau_{AP}^{CP}(\Pi)$. Then, after the problem has been made acyclic through τ_{AP}^{CP}, it is passed to the τ_{SP}^{AP} module. Given a planning problem *with* acyclic CEs Π' as input, τ_{SP}^{AP} produces one *without* CEs, denoted as $\Pi'' = \tau_{SP}^{AP}(\Pi')$. The composition of these two modules, denoted by τ, constitutes the COCOA compilation process. Below, the two modules are formalised.

Module τ_{SP}^{AP} Given $\Pi = \langle F, A, I, G\rangle$, τ_{SP}^{AP} produces the problem $\tau_{SP}^{AP}(\Pi) = \langle F', A', I, G'\rangle$ where:

$$F' = F \cup \bigcup_{a\in A} \{d_{i,a} \mid i \in \{0,\dots,\kappa_a\}\} \cup \{pause\}$$
$$A' = A^{start} \cup A^{end} \cup A^{\text{CE}+} \cup A^{\text{CE}-}$$
$$A^{start} = \bigcup_{a\in A} \{a^{start} = \langle pre(a) \cup \{\neg pause\}, \{\langle \emptyset, \{d_{0,a}, pause\}\rangle\}\rangle\}$$
$$A^{end} = \bigcup_{a\in A} \{a^{end} = \langle \{d_{\kappa_a,a}\}, \{\langle \emptyset, \{\neg d_{\kappa_a,a}, \neg pause\}\rangle\}\rangle\}$$
$$A^{\text{CE}+} = \bigcup_{a\in A} A_a^{\text{CE}+} \quad A^{\text{CE}-} = \bigcup_{a\in A} A_a^{\text{CE}-} \quad G' = G \cup \{\neg pause\}.$$

Referring to $A^{\text{CE}+}$ and $A^{\text{CE}-}$, consider an action $a \in A$ and let $\langle v_{e_1}, \dots, v_{e_{\kappa_a}}\rangle$ be a topological ordering of the κ_a vertices of $\mathbb{G}_a^T$ (where v_{e_i} denotes a CE e_i of a). $A_a^{\text{CE}+}$ has an action $a_{e_i}^{+}$ for each vertex v_{e_i} of $\mathbb{G}_a^T$ such that

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

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

$$a_{e_i}^{l} = \langle \{d_{i-1,a}, \neg l\}, \{\langle \emptyset, \{d_{i,a}, \neg d_{i-1,a}\}\rangle\}\rangle.$$

Module τ_{AP}^{CP} Given $\Pi = \langle F, A, I, G\rangle$, τ_{AP}^{CP} produces $\tau_{AP}^{CP}(\Pi) = \langle F', A', I, G'\rangle$ where:

$$F' = F \cup \bigcup_{a\in A} \{t_p \mid p \in \text{MFPS}(a)\} \cup \bigcup_{a\in A} \{run_a\} \cup \{set\}$$
$$A' = \{a^{setup} \mid a \in A\} \cup \{a^{run} \mid a \in A\}$$
$$G' = G \cup \{\neg set\} \cup \{\neg run_a \mid a \in A\}$$
$$pre(a^{setup}) = pre(a) \cup \{\neg set\}$$
$$eff(a^{setup}) = \bigcup_{p\in \text{MFPS}(a)} \{\langle \{p\}, \{t_p\}\rangle, \langle \{\neg p\}, \{\neg t_p\}\rangle\} \cup \{\langle \emptyset, \{run_a, set\}\rangle\}$$
$$pre(a^{run}) = \{run_a\}$$
$$eff(a^{run}) = \bigcup_{e\in post(a)} \{\langle \mathcal{S}(cond(e), a), eff(e)\rangle\} \cup \{\langle \emptyset, \{\neg run_a, \neg set\}\rangle\}$$

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.

Enhancing Polynomial Compilations

This section discusses enhancements for existing polynomial compilations. We introduce a new hybrid compilation, merging the advantages of COCOA and the exponential approach by Gazen and Knoblock (1997) (hereafter referred to as GKCOMP) while mitigating their weaknesses. Additionally, we explore the underlying optimisations of COCOA focused on minimising the increase in plan size. These optimisations are examined to assess their impact from an experimental point of view later, providing a more precise depiction of the existing COCOA implementation. Regarding NCOMP, being a theoretical compilation, we discuss a practical variant to establish a stronger baseline for comparison with COCOA. This aims to better contextualise COCOA compared to NCOMP and emphasise the structural differences between the two compilations.

Hybrid Polynomial Compilation

The rationale behind the hybrid compilation is to develop a scheme where a single action is compiled differently based on its number of CEs, rather than applying the same transformation to each action in the problem.

Let a be an action, and $\kappa_a = |post(a)|$ its number of CEs. Let K be a natural number serving as a discriminating factor between GKCOMP and COCOA. Intuitively, if an action has many CEs (say $\kappa_a > K$), making GKCOMP infeasible, it will be compiled polynomially using COCOA. Conversely, if the action has a moderate number of CEs (say $\kappa_a \leq K$), it will be compiled exponentially using GKCOMP, thereby avoiding unnecessary plan lengthening. Below we formalise this compilation, but to facilitate the exposition, we introduce an alternative notation for compactly representing the compilation τ, borrowed from the one used by Nebel (2000).

Let $\Pi = \langle F, A, I, G\rangle$ be a planning problem with CEs. The compilation τ associated with COCOA can be represented as a tuple of functions $\langle \tau_{\text{F}}, \tau_{\text{A}}, \tau_{\text{I}}, \tau_{\text{G}}\rangle$ such that

$$\Pi' = \tau(\Pi) = \langle \tau_{\text{F}}(F, A), \tau_{\text{A}}(A), \tau_{\text{I}}(I), \tau_{\text{G}}(G)\rangle,$$

where each function of the tuple refers to and acts upon, each element of Π. For example, the function for manipulating the atoms F is defined by combining the atoms generated by τ_{AP}^{CP} and τ_{SP}^{AP} executed in sequence:

$$\tau_{\text{F}}(F, A) = F \cup \bigcup_{a\in A} \big(\{t_p \mid p \in \text{MFPS}(a)\} \cup \{run_a\} \cup \{d_{i,a} \mid i \in \{0,\dots,\kappa_a\}\}\big) \cup \{set, pause\}.$$

A similar methodology can be followed to define τ_{A}. As for the initial state and the goal state, we have that $\tau_{\text{I}} = I$ and $\tau_{\text{G}} = G \cup \{\neg \mathit{pause}, \neg \mathit{set}\} \cup \{\neg \mathit{run}_a \mid a \in A\}$.

Furthermore, let η represent the GKCOMP compilation. η is structured as a tuple of functions given by $\langle F, \eta_{\text{A}}(A), I, G\rangle$, where F, I and G are unchanged, while η_{A} generates a set of actions, potentially exponential in the worst-case scenario, following GKCOMP. (Intuitively, GKCOMP compiles each action by generating all combinations in which CEs can activate simultaneously.)

Now we define the hybrid compilation denoted by τ_K, incorporating elements from both COCOA and GKCOMP. Given a planning problem $\Pi = \langle F, A, I, G\rangle$, we categorise the actions based on their number of CEs, resulting in $A = A_{\leq K} \cup A_{>K}$, where $A_{\leq K} = \{a \in A \mid \kappa_a \leq K\}$ and $A_{>K} = A \setminus A_{\leq K}$. The translation τ_K comprises the following tuple of functions:

$$\Pi' = \langle \tau_{\text{F}}(F, A_{>K}), \eta'_{\text{A}}(A_{\leq K}) \cup \tau_{\text{A}}(A_{>K}), \tau_{\text{I}}(I), \tau_{\text{G}}(G)\rangle.$$

During the compilation process, the additional atoms are generated only for actions having more than K CEs ($\tau_{\text{F}}(F, A_{>K})$). Actions are polynomially compiled if they have more than K CEs ($\tau_{\text{A}}(A_{>K})$). Conversely, actions with K or fewer CEs are compiled exponentially according to $\eta'_A(A_{\leq K}) = \{\langle \mathit{pre}(a) \cup \{\neg \mathit{pause}, \neg \mathit{set}\}, \mathit{post}(a)\rangle \mid a \in \eta_A(A_{\leq K})\}$. The initial and goal states remain unchanged relative to τ.

Despite including a module for exponential action compilation, τ_K remains a polynomial scheme. This is because η_A is only applied to actions with no more than K CEs. Consequently, as the maximum number of CEs in A per single action, denoted as $n_{\max}$, increases, the size of $\Pi' = \tau_K(\Pi)$ grows polynomially in relation to $n_{\max}$. This implementation of τ_K will be called HCOCOA$_K$.

COCOA Optimisations

In this section, we outline some native optimisations within COCOA. These optimisations regard handling simple actions (actions without CEs) and simple effects (CEs where the condition is $\emptyset$). Further, we propose a new optimisation to handle mutually exclusive CEs generated by the τ^{CP}_{AP} module.

Handling Simple Actions The verbatim version of COCOA does not distinguish between simple actions and actions with CEs. Therefore, each action is compiled sequentially even if it is simple. This redundant treatment of actions can be easily avoided by restricting the set of actions to be compiled. Specifically, we partition the actions A as $A_{\text{sim}} \cup A_{\text{CE}}$, where A_{sim} contains only the simple actions of A and A_{CE} the remaining ones. Now we can redefine τ as $\tau_{\text{simple}} = \langle \tau_{\text{F}}(F, A_{\text{CE}}), \tau_{\text{A}}(A_{\text{CE}}) \cup A'_{\text{sim}}, \tau_{\text{I}}(I), \tau_{\text{G}}(G)\rangle$ where $A'_{\text{sim}} = \{\langle \mathit{pre}(a) \cup \{\neg \mathit{pause}, \neg \mathit{set}\}, \mathit{post}(a)\rangle \mid a \in A_{\text{sim}}\}$.

Avoiding Simple Effects Compilation In many domains, it is common to have actions where both CEs and simple effects coexist. In such cases, we can further shorten the compiled actions sequence. Specifically, without loss of generality, let us assume that actions are normalised in a standard form, i.e., all CEs sharing the same condition, empty or not, are grouped together; then, we can reshape $a = \langle \mathit{pre}(a), \mathit{post}(a)\rangle$ into $a = \langle \mathit{pre}(a), \mathit{post}_{\text{CE}}(a) \cup \{\langle \emptyset, \mathit{eff}_{\text{simple}}(a)\rangle\}\rangle$. Let κ_a now be defined as $\kappa_a = |\mathit{post}_{\text{CE}}(a)|$. The compilation τ^{AP}_{SP} can be easily revised with some adjustments. Specifically, the interference graph of each action should be constructed using only $\mathit{post}_{\text{CE}}(a)$, thus ignoring the simple effects. Since no CEs can interfere with simple effects, they can be safely added to the last action of the sequence associated with a, i.e., $a^{\mathit{end}} = \langle \{d_{\kappa_a,a}\}, \{\langle \emptyset, \{\neg d_{\kappa_a,a}, \neg \mathit{pause}\} \cup \mathit{eff}_{\text{simple}}(a)\rangle\}\rangle$.

Mutexes Conditional Effects The τ^{CP}_{AP} module, as currently presented, may unnecessarily lengthen plans when pipelined with τ^{AP}_{SP}. Specifically, let us consider an action a that induces a cyclic graph $\mathbb{G}^T_a$. For each atom $p \in \text{MFPS}(a)$, the action a^{setup} generated by τ^{CP}_{AP} encompasses two CEs of the form $e = \langle \{p\}, \{t_p\}\rangle$ and $e' = \langle \{\neg p\}, \{\neg t_p\}\rangle$ used to setup the twin atom t_p. When a^{setup} is compiled by the τ^{AP}_{SP} module, four actions are generated for handling p: $\{a^+_e, a^+_{e'}\}$, which are used when e and e' are entailed, and $\{a^p_e, a^{\neg p}_{e'}\}$, which are used when they are not. However, e and e' are mutually exclusive. Therefore, if a^+_e ($a^+_{e'}$) is applied, the only applicable action for e' (e) is $a^{\neg p}_{e'}$ (a^p_e). By making τ^{AP}_{SP} aware that the CEs generated by τ^{CP}_{AP} are mutually exclusive, it is possible to generate a pair of actions for handling p instead of four. Namely, $a^+_p = \langle \{p, d_{i-1,a}\}, \{\langle \emptyset, \{t_p, \neg d_{i-1,a}, d_{i,a}\}\rangle\}\rangle$ and $a^-_p = \langle \{\neg p, d_{i-1,a}\}, \{\langle \emptyset, \{\neg t_p, \neg d_{i-1,a}, d_{i,a}\}\rangle\}\rangle$. This improvement can significantly impact the plan size in highly cyclic problems that generate numerous twin atoms, halving the length of the setup sequence induced by τ^{AP}_{SP}.

In the following, we will refer to the system without the discussed enhancements as VANILLA, and we will denote the fully equipped system with all optimisations as COCOA.

Example 1 (HCOCOA$_K$ Compilation). *Consider the hybrid compilation with $K = 2$, i.e., τ_2. Let a_1 be an action having two* CEs*, i.e., $\mathit{post}(a_1) = \{\langle \{p_1\}, \{p_2\}\rangle, \langle \{p_3\}, \{\neg p_4\}\rangle\}$. Since $\kappa_{a_1} \leq K$, a_1 undergoes the* GKCOMP *compilation, generating the following four actions:*

$$a_{11} = \langle \mathit{pre}(a_1) \cup \{p_1, p_3, \neg \mathit{pause}, \neg \mathit{set}\}, \{\langle \emptyset, \{p_2, \neg p_4\}\rangle\}\rangle$$
$$a_{12} = \langle \mathit{pre}(a_1) \cup \{p_1, \neg p_3, \neg \mathit{pause}, \neg \mathit{set}\}, \{\langle \emptyset, \{p_2\}\rangle\}\rangle$$
$$a_{13} = \langle \mathit{pre}(a_1) \cup \{\neg p_1, p_3, \neg \mathit{pause}, \neg \mathit{set}\}, \{\langle \emptyset, \{\neg p_4\}\rangle\}\rangle$$
$$a_{14} = \langle \mathit{pre}(a_1) \cup \{\neg p_1, \neg p_3, \neg \mathit{pause}, \neg \mathit{set}\}, \{\langle \emptyset, \emptyset\rangle\}\rangle.$$

Let a_2 be an action having three CEs*, i.e., $\mathit{post}(a_2) = \{e_1, e_2, e_3\}$ where $e_1 = \langle \{p_1\}, \{p_2\}\rangle$, $e_2 = \langle \{p_3, \neg p_2\}, \{p_4, \neg p_1\}\rangle$ and $e_3 = \langle \{p_5\}, \{\neg p_3\}\rangle$. Since $\kappa_{a_2} > K$, a_2 undergoes the* COCOA *compilation. By using the definition of effect interference, we get that $e_1 \triangleright e_2$, $e_2 \triangleright e_1$ and $e_3 \triangleright e_2$. The first two graphs on the left of Figure 1 show $\mathbb{G}_{a_2}$ and $\mathbb{G}^T_{a_2}$. Since $\mathbb{G}^T_{a_2}$ is cyclic, it must be preprocessed by τ^{CP}_{AP} and the minimum set of atoms to be removed to make it acyclic is* $\text{MFPS}(a_2) = \{p_1\}$*. Therefore, τ^{CP}_{AP} introduces the twin atom t_{p_1} of p_1 and splits a_2 into the actions $\{a^{\mathit{setup}}_2, a^{\mathit{run}}_2\}$. The action a^{setup}_2 either adds or deletes t_{p_1} depending on the value of p_1, using the new* CEs

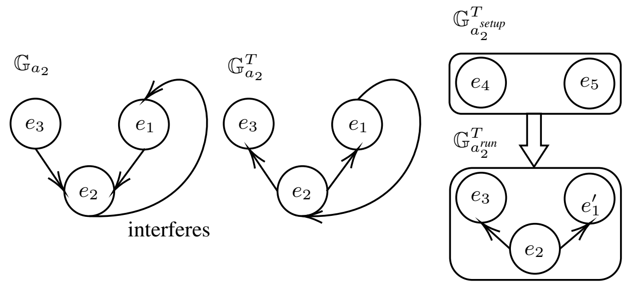

Figure 1: Overview of the compilation process of a_2. (left) $\mathbb{G}_{a_2}$, (center) $\mathbb{G}^T_{a_2}$, (right) $\mathbb{G}_{a_2^{setup}}$ and $\mathbb{G}_{a_2^{run}}$. The arrow between two vertices in $\mathbb{G}_{a_2}$ represents effect-interference. The last two graphs are connected as the two actions must be executed sequentially.

e_4 and e_5. This happens before the execution of a_2^{run}. The definition of this action is:

$$a_2^{setup} = \langle pre(a_2) \cup \{\neg set\}, \{e_4, e_5, \langle \emptyset, \{run_{a_2}, set\}\rangle\}\rangle,$$

where $e_4 = \langle\{p_1\}, \{t_{p_1}\}\rangle$ and $e_5 = \langle\{\neg p_1\}, \{\neg t_{p_1}\}\rangle$.

The action a_2^{run} is responsible for evaluating and executing the CE*s of a_2. It is defined as:*

$$a_2^{run} = \langle\{run_{a_2}, set\}, \{e_1', e_2, e_3, \langle\emptyset, \{\neg run_{a_2}, \neg set\}\rangle\}\rangle,$$

where e_2 and e_3 remain unchanged and e_1 is replaced by $e_1' = \langle\{t_{p_1}\}, \{p_2\}\rangle$.

Once acyclicity has been achieved, the CE*s can be compiled using τ_{SP}^{AP}. In particular, the last two graphs on the right in Figure 1 depict $\mathbb{G}^T_{a_2^{setup}}$ and $\mathbb{G}^T_{a_2^{run}}$. From these graphs the interference-free topological orderings $\langle e_4, e_5\rangle$ and $\langle e_2, e_3, e_1'\rangle$ are extracted. For brevity, we show how $\langle e_2, e_3, e_1'\rangle$ is used to generate the compiled actions for a_2^{run} (for simplicity relabelled as a):*

$$\begin{aligned}
a^{start} &= \langle\{run_a, set, \neg pause\}, \{\langle\emptyset, \{d_{0,a}, pause\}\rangle\}\rangle \\
a^+_{e_2} &= \langle\{p_3, \neg p_2, d_{0,a}\}, \{\langle\emptyset, \{p_4, \neg p_1, \neg d_{0,a}, d_{1,a}\}\rangle\}\rangle \\
a^{p_3}_{e_2} &= \langle\{\neg p_3, d_{0,a}\}, \{\langle\emptyset, \{\neg d_{0,a}, d_{1,a}\}\rangle\}\rangle \\
a^{\neg p_2}_{e_2} &= \langle\{p_2, d_{0,a}\}, \{\langle\emptyset, \{\neg d_{0,a}, d_{1,a}\}\rangle\}\rangle \\
a^+_{e_3} &= \langle\{p_5, d_{1,a}\}, \{\langle\emptyset, \{\neg p_3, \neg d_{1,a}, d_{2,a}\}\rangle\}\rangle \\
a^{p_5}_{e_3} &= \langle\{\neg p_5, d_{1,a}\}, \{\langle\emptyset, \{\neg d_{1,a}, d_{2,a}\}\rangle\}\rangle \\
a^+_{e_1'} &= \langle\{t_{p_1}, d_{2,a}\}, \{\langle\emptyset, \{p_2, \neg d_{2,a}, d_{3,a}\}\rangle\}\rangle \\
a^{t_{p_1}}_{e_1'} &= \langle\{\neg t_{p_1}, d_{2,a}\}, \{\langle\emptyset, \{\neg d_{2,a}, d_{3,a}\}\rangle\}\rangle \\
a^{end} &= \langle\{d_{3,a}\}, \{\langle\emptyset, \{\neg d_{3,a}, \neg run_a, \neg set, \neg pause\}\rangle\}\rangle.
\end{aligned}$$

According to the optimisation concerning mutexes CE*s generated by τ_{AP}^{CP}, e_4 and e_5 are compiled into two mutexes actions instead of four. Note that the simple effects of a_2^{run} have been appended to the end of the sequence in the a^{end} action.*

A Practical Implementation of NCOMP

NCOMP is a theoretical compilation provided as a side-effect for demonstrating that a problem with CEs can be compiled in polynomial time while preserving plan size polynomially (Nebel 2000, Theorems 20-22). Therefore, NCOMP should be considered a theoretical construct rather than a practical compilation tool. This section describes how NCOMP can be adapted for practical application. The resulting scheme is referred to as NCOMP*.

Size of the Encoding NCOMP uses several additional atoms that must be deleted every time a sequence associated with an action is concluded. Specifically, they are deleted both in a_i^{pre} and a^c. Since there is an action a_i^{pre} for each action in the input problem, this results in an inflation of the resulting problem. Therefore, the atoms deletion can be moved to a^c, producing a lightweight problem.

Parallelisation of the CEs Evaluation NCOMP assumes that the CEs have the form $\langle C, \{l\}\rangle$. As a result, if there is a CE $\langle C, L\rangle$, where L is not a singleton, the evaluation phase will contain one action for each literal in L. Instead, NCOMP* handles the evaluation of the CEs in parallel. Therefore, in the case of non-singleton effects, there will be only one action in *exec-phase* instead of $|L|$.

Enforcing an Ordering The action sequence outlined in Equation 1 allows for various orderings while achieving the same state outcome. Specifically, the actions within the evaluation and execution phases can be reordered. Given that the separation of evaluation and executive phases eliminates the possibility of interferences between CEs, it is possible to enforce an absolute and arbitrary ordering for the actions in the *eval-phase*. However, this is impossible for the executive phase due to its variable length.

Experimental Analysis

The experimental section is structured in two parts. The first part evaluates the impact of the proposed optimisations. In particular, the behaviour of HCOCOA$_K$ is studied as K varies. Next, we evaluate the impact of the optimisations implemented in COCOA and NCOMP. In the second part, we systematically compare the native and compilation-based approaches, considering all the discussed compilations.

Similar to the approach by Gerevini, Percassi, and Scala (2024), the focus is on the optimal setting, as in this context CEs are less supported. Moreover, we utilised the same benchmark suite, which includes domains collected from various sources: the Fast Downward benchmark collection (https://github.com/aibasel/downward-benchmarks) and problems generated by conformant-to-classical planning compilations (Palacios and Geffner 2009; Grastien and Scala 2017), identified by the prefixes T0 and CPCES. Moreover, the domains used by Röger, Pommerening, and Helmert (2014) are included.

Each test run in our experiment involves a planner, a compilation method, and a planning problem from the benchmark suite. Each run first compiles the problem and then sends the output to an off-the-shelf planner. If a problem is given directly to a planner without compilation, it is labelled as PLAIN. We allocated a runtime budget of 1,800 seconds (compilation plus solving), 8 GB of memory, and 2 GB of disk space. The experiments ran on an Intel Xeon Gold

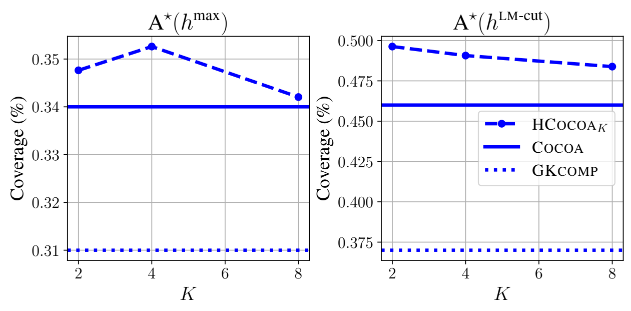

Figure 2: Percentage of problems solved by HCocoa$_K$ as K varies in $\{2, 4, 8\}$, Cocoa and GKcomp, considering A$^\star(h^{\max})$ and A$^\star(h^{\text{LM-cut}})$.

6140M CPU at 2.30 GHz. The compilation tool is available at https://gitlab.com/EdmondDantes/cocoa2.0.

Rational Behind the Planners Choice 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 $h^{\max}$ (Haslum and Geffner 2000), i.e., A$^\star(h^{\max})$. These systems are employed in the first part of the experimental evaluation, concerning the optimisation assessment, aiming to maintain a controlled setting. Notably, $h^{\max}$ natively supports CEs while, regarding $h^{\text{LM-cut}}$, we utilised the CEs-aware variant when Plain problems are addressed (Röger, Pommerening, and Helmert 2014).

In the planning systems comparison, we selected two planners to favour both the compilation-based and native approaches. For the compilation-based approach, we chose Metis2 (Domshlak, Katz, and Shleyfman 2012; Alkhazraji et al. 2014; Shleyfman et al. 2015; Sievers and Katz 2018) because it has demonstrated effectively handling compiled problems, regardless of the scheme used (Gerevini, Percassi, and Scala 2024) On the native side instead, we opted for SymK (Speck et al. 2019; Speck, Mattmüller, and Nebel 2020; Speck 2023), a top-k planner based on symbolic search that natively supports a wide range of planning features, including CEs. As recommended by the authors, we utilised the bidirectional symbolic search configuration to find a single optimal plan. All the systems considered so far are based on Fast Downward (Helmert 2006).

Enhancements Evaluation

Hybrid Cocoa This section evaluates the performance of HCocoa$_K$ by varying the parameter K to find the threshold that maximises its behaviour. We examined $K \in \{2, 4, 8\}$. Figure 2 shows the percentage of problems solved as K changes. To highlight the benefits of hybrid compilation, we compared HCocoa$_K$ with its components, Cocoa and GKcomp. In all cases, HCocoa$_K$ outperformed its individual compilations, especially when paired with A$^\star(h^{\text{LM-cut}})$. The curve's trend varies with the heuristic, but the optimal value for K seems to be between two and four.

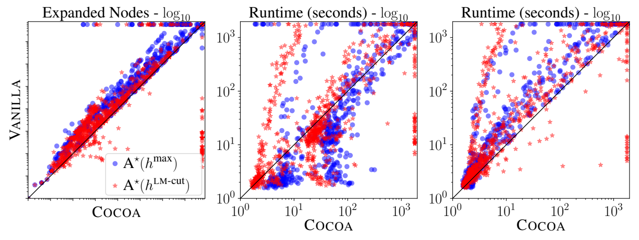

Figure 3: Comparison between Vanilla and Cocoa regarding expanded nodes and runtime, considering A$^\star(h^{\max})$ and A$^\star(h^{\text{LM-cut}})$. Points above the bisector favour Cocoa, and points on the edges correspond to unsolved instances. The first two figures refer to the default run obtained with Fast Downward, while the third one refers to the run where invariant generation has been disabled.

Cocoa In this section, we assess the impact of the optimisations in Cocoa compared to its verbatim implementation, Vanilla. The first two plots on the left side of Figure 3 compare the two compilations paired with A*, focusing on the expanded nodes and runtime. As expected, the optimisations integrated into Cocoa yield favourable results regarding expanded nodes, with most data points positioned above the bisector. However, the runtime plot reveals a mixed scenario, with Vanilla often exhibiting faster performance. Further investigation into the logs revealed that this difference originates from the invariant generation process in Fast Downward. While this process is generally useful (Alcázar and Torralba 2015), it produces too many candidates in grounded instances like the ones being considered.

To delve deeper, we disabled the invariant generation and experimented again. The runtime comparison without invariant generation is shown in the third plot on the right in Figure 3. The results indicate that, in this benchmark, invariant generation does not improve performance and may even hinder it. Notably, the system's coverage slightly decreases when computing invariants, such as in A$^\star(h^{\max})$, in which it drops from 544 to 529.

Ncomp* In this section, we evaluate the benefits of the optimised variant Ncomp* compared to its original implementation. We consider various aspects influenced by the discussed enhancements: encoding size, plan size, expanded nodes, and runtime.

Figure 4 summarises the results obtained through scatter plots and a box plot. For encoding size, we analysed all commonly compiled instances. Regarding the plan size, we focused on instances commonly solved by A$^\star(h^{\text{LM-cut}})$ to avoid redundancy.

The optimisations introduced by Ncomp* significantly impact all the considered metrics. For example, encoding size for the largest cases can decrease by as much as two orders of magnitude. Regarding plan size, evaluating CEs in parallel reduces it by an average of 35%. In the worst case, Ncomp produces a plan of around 3,000 actions, while Ncomp* plan has only 158 actions. Similar trends are observable for expanded nodes and runtime.

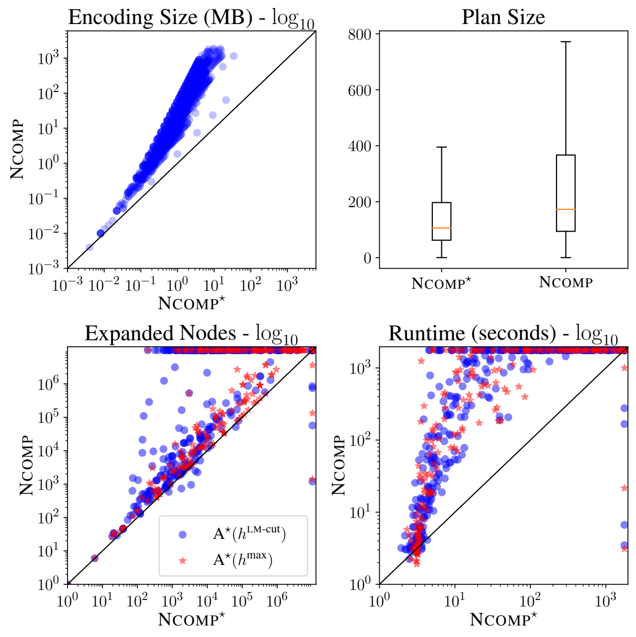

Figure 4: Comparison between NCOMP* and NCOMP regarding encoding size (top-left), plan size (top-right), expanded nodes (bottom-left), and runtime (bottom-right) for A*($h^{\max}$) and A*($h^{\text{LM-cut}}$). Points above the bisector favour NCOMP*, and points on the edges correspond to unsolved instances.

Planning Systems Comparison

This section compares the compilation-based approach with those that natively support CEs, referred to as PLAIN, across two full-fledged planning systems. For HCOCOA$_K$, we included two variants with $K \in \{2, 3\}$, as suggested by previous experiments. Notably, according to our experiments, A*($h^{\text{LM-cut}}$) achieved good coverage on the PLAIN formulation, though lower than that achieved by SYMK (828 vs. 892). Therefore, it was not included in this analysis.

Table 1 shows the coverage achieved by each system with a given problem formulation.[2] The first notable result is that the combined use of HCOCOA$_2$ with METIS2 achieves performance comparable to that of SYMK on PLAIN problems, with a coverage of 905 compared to 892.

The scatter plot in Figure 5 (left) illustrates the runtime between the two systems and highlights a significant degree of complementarity. Specifically, METIS2 $\circ$ HCOCOA$_2$ solves 105 problems that SYMK$\circ$PLAIN fails to solve, while SYMK $\circ$ PLAIN solves 98 cases that METIS2 $\circ$ HCOCOA$_2$ does not. Interestingly, despite the drawback of polynomial compilation, which is partially mitigated by the GKCOMP module, HCOCOA$_2$ exhibits slightly better runtime performance, as indicated by the data points that tend to lie above the bisector. Regarding METIS2, HCOCOA$_2$ seems faster than PLAIN as it achieves higher coverage. However, PLAIN

[2]All the plans generated on the compiled problems are validated against the original formulation.

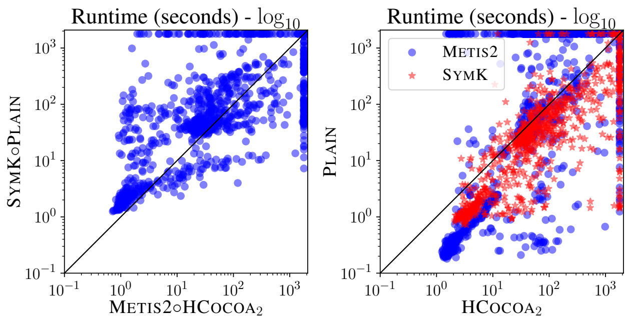

Figure 5: Runtime comparison between METIS2 $\circ$ HCOCOA$_2$ and SYMK $\circ$ PLAIN (left). Runtime comparison between HCOCOA$_2$ and SYMK for METIS2 and PLAIN (right). Points above the bisector favour the approach on the x-axis, while points on the edges indicate unsolved instances.

is hindered by Fast Downward's preprocessing issue. Observing the scatter plot in Figure 5 (right), which shows the runtime for HCOCOA$_2$ and PLAIN combined with METIS2, it appears that the native approach tends to be faster. Generally speaking, when using the same planning system, the native approach outperforms the polynomial compilation, as the latter bears the burden of lengthened plans. Indeed, the same trend is observed when comparing the performance of SYMK with PLAIN and HCOCOA$_2$.

The results obtained within the polynomial schemes align with the discussions in previous sections. Regarding hybrid compilation, the choice of K that maximises coverage varies depending on the system. Furthermore, upon closer examination of the results, it can be observed that varying K can lead to higher coverage in certain domains while reducing it in others. For instance, when using METIS2 with $K = 3$, 10 additional instances are solved in CPCES-UTS compared to $K = 2$, though coverage decreases in other domains. This lack of dominance suggests room for smarter, instance-specific strategies for selecting K.

Moreover, regardless of the planner used, HCOCOA$_K$ improves the performance of COCOA. The improvement is more pronounced in SYMK (around +12% coverage) than in METIS2 (around +5% coverage). There are also interesting observations regarding NCOMP* versus NCOMP. Specifically, both planners show notable improvement, increasing coverage by +80% for SYMK and +36% for METIS2.

However, despite the significant increase in coverage, these gains are not enough to compete with COCOA-based approaches, GKCOMP, and native methods. This finding reinforces the idea that COCOA-based approaches have characteristics that go beyond simple optimisations of NCOMP, allowing polynomial compilations to remain competitive.

Finally, Figure 6 illustrates the coverage over time for METIS2 combined with all the discussed compilations and the system SYMK $\circ$ PLAIN. The VBS (Virtual Best Solver) was achieved by combining the best of both the compiled and native approaches, specifically METIS2$\circ$HCOCOA$_2$ and SYMK$\circ$PLAIN, to demonstrate the complementarity of these

Domain (no. of instances)	SymK								Metis2							
	V	C	H_2	H_3	N	N⋆	G	P	V	C	H_2	H_3	N	N⋆	G	P
Airport (50)	17	**20**	**20**	**20**	0	2	3	**20**	25	**27**	**27**	**27**	7	14	3	0
Assembly (30)	11	16	**17**	**17**	0	0	**17**	12	**4**	**4**	**4**	**4**	0	0	**4**	0
Briefcase (50)	6	7	7	6	3	2	7	**8**	6	6	6	6	6	2	7	**9**
Cald-Opt (20)	8	8	**12**	**12**	0	4	**12**	9	11	13	**16**	**16**	4	6	**16**	15
Cald-Sat (20)	2	2	**4**	**4**	0	0	**4**	2	4	4	**6**	**6**	0	0	**6**	5
Cald-Split-Opt (20)	8	**14**	**14**	**14**	0	0	**14**	11	9	**11**	**11**	10	2	4	**11**	**11**
Cald-Split-Sat (20)	2	5	**6**	**6**	0	0	**6**	**6**	2	3	3	3	0	0	**4**	**4**
Citycar-Opt (20)	10	**18**	**18**	17	0	0	17	**18**	15	13	13	9	2	8	**18**	**18**
Citycar-Sat (20)	0	**1**	**1**	**1**	0	0	**1**	**1**	1	0	0	0	0	0	1	**3**
Miconic (150)	79	141	**150**	**150**	27	43	**150**	**150**	132	142	143	143	45	77	**144**	**144**
Nurikabe-Opt (20)	10	10	10	10	0	1	10	**12**	10	10	10	10	2	4	10	**12**
Nurikabe-Sat (20)	4	5	4	4	0	0	3	**7**	4	5	5	5	0	0	5	**6**
Schedule (150)	22	14	15	21	0	5	7	**45**	26	26	23	18	17	12	13	**44**
T0-Coins (30)	10	11	15	15	5	5	**17**	**17**	0	13	12	13	12	12	**15**	**15**
T0-Comm (25)	6	7	7	7	2	0	14	**15**	0	3	4	4	3	0	**6**	**6**
T0-Grid-Dispose (15)	1	1	1	1	0	0	0	**2**	2	**3**	1	1	2	0	0	0
T0-Grid-Push (5)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T0-Sortnet-Alt (6)	**3**	**3**	**3**	**3**	1	1	1	2	1	**3**	**3**	**3**	1	1	0	1
T0-Uts (29)	5	5	5	5	3	4	4	**6**	0	5	5	5	4	3	4	**6**
Cpces-Blocks (47)	15	15	15	**16**	6	8	**16**	12	**18**	**18**	**18**	17	11	11	13	0
Cpces-Bomb (180)	113	130	160	160	8	21	102	**170**	136	161	**180**	**180**	51	69	102	89
Cpces-Coins (94)	53	56	64	66	9	18	71	**76**	66	66	69	**71**	37	34	66	60
Cpces-Disp (190)	128	141	147	152	6	19	103	**156**	167	178	**180**	176	40	63	103	94
Cpces-One-Disp (181)	91	95	96	97	18	32	78	**102**	115	**118**	**118**	110	48	53	57	54
Cpces-Uts (216)	25	25	34	**35**	12	15	**35**	33	28	28	48	58	22	22	**79**	0
Σ (1608)	629	750	825	839	100	180	692	**892**	782	860	**905**	895	316	395	687	596

Table 1: Coverage for all planners under different problem formulations. **V** stands for Vanilla, **C** for Cocoa, $\mathbf{H}_K$ for HCocoa$_K$, **N** for Ncomp, **N**⋆ for Ncomp⋆, **G** for GKcomp, and **P** for Plain. Bold indicates the best result in a given planning system, while underlined bold represents the best result across all planning systems and problem formulations.

two methods. This visualisation depicts the dominance hierarchy among the compilations, from Ncomp to HCocoa$_K$, and highlights the degree of complementarity between the top-performing planning systems.

Summary

The experimental analysis yielded the following findings in optimal planning with CEs. (i) Simple optimisations of polynomial compilations can have significant experimental impacts. (ii) Metis2 ∘ HCocoa$_2$ achieves performance comparable to SymK, a versatile planner that robustly supports CEs. (iii) The optimised variant of Ncomp is still not competitive compared to Cocoa-based approaches, indicating that leveraging the structure of CEs is crucial for achieving good performance.

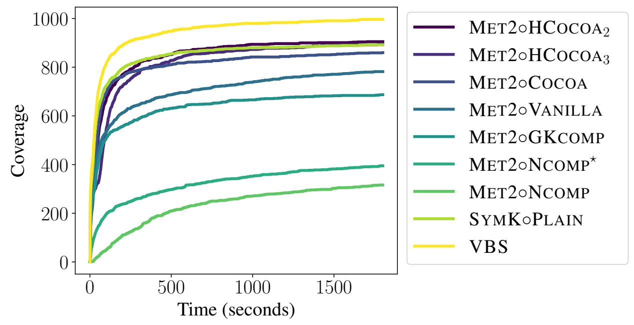

Figure 6: Coverage over time for all compilations combined with Metis2 and SymK ∘ Plain. The VBS (Virtual Best Solver) is achieved by combining Metis2 ∘ Cocoa$_2$ and SymK ∘ Plain.

Conclusions

In this article, we proposed and empirically assessed optimised variants of existing polynomial compilation for conditional effects. Specifically, we introduced a novel hybrid compilation approach that combines the strengths of Cocoa and GKcomp. This scheme selectively applies GKcomp to actions with a limited number of conditional effects, avoiding unnecessary increases in plan size. Conversely, actions with several conditional effects undergo polynomial compilation, avoiding the exponential blowup. The resulting scheme remains polynomial because an input parameter bounds the exponential module.

The hybrid compilation also outperforms its stand-alone modules and competes with a state-of-the-art planning system that natively handles conditional effects. We discuss enhancements to Cocoa, showing how minor tweaks can significantly boost the compilation, suggesting potential for even more advanced techniques. Additionally, we present a practical version of Nebel's compilation to provide clearer insights into Cocoa's unique features. In future work, we plan to extend Cocoa to handle conflicting conditional effects and explore the satisficing context experimentally.

Acknowledgments

Francesco Percassi was supported by a UKRI Future Leaders Fellowship [grant number MR/T041196/1]. Enrico Scala and Alfonso Gerevini were supported by Climate Change AI project (No. IG-2023-174), EU H2020377 project AIPlan4EU (No. 101016442), and MUR PRIN project RIPER (No. 20203FFYLK). Alfonso Gerevini was also supported by EU ICT-48 2020 project TAILOR (No. 952215).

References

Alcázar, V.; and Torralba, A. 2015. A Reminder About the Importance of Computing and Exploiting Invariants in Planning. In *Proc. of ICAPS*, volume 25, 2–6.

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. FOND Planning for Pure-Past Linear Temporal Logic Goals. In *ECAI 2023*, 279–286.

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.

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*.

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.

Gerevini, A. E.; Percassi, F.; and Scala, E. 2024. An Effective Polynomial Technique for Compiling Conditional Effects Away. In *Proc. of AAAI*, 20104–20112.

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. Cybern.*, 4(2): 100–107.

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

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*, volume 19, 162–169.

Katz, M.; and Hoffmann, J. 2014. Mercury Planner: Pushing the Limits of Partial Delete Relaxation. *IPC 2014 Planner Abstracts*, 43–47.

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

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.

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.

Shleyfman, A.; Katz, M.; Helmert, M.; Sievers, S.; and Wehrle, M. 2015. Heuristics and Symmetries in Classical Planning. In *Proc. of AAAI*, 3371–3377.

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

Speck, D. 2023. SymK–A Versatile Symbolic Search Planner. *IPC 2023 Planner Abstracts*.

Speck, D.; Geißer, F.; Mattmüller, R.; and Torralba, Á. 2019. Symbolic Planning with Axioms. In *Proc. of ICAPS*, 464–472.

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

Speck, D.; Mattmüller, R.; and Nebel, B. 2020. Symbolic Top-k Planning. In *Proc. of AAAI*, 9967–9974.

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.