Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning
KR in Planning and Scheduling

A Planning Compilation to Reason about Goal Achievement at Planning Time

Alberto Pozanco , Marianela Morales , Daniel Borrajo , Manuela Veloso
J.P. Morgan AI Research
{alberto.pozancolancho,marianela.moraleselena}@jpmorgan.com, {name.surname}@jpmorgan.com

Abstract

Identifying the specific actions that achieve goals when solving a planning task might be beneficial for various planning applications. Traditionally, this identification occurs post-search, as some actions may temporarily achieve goals that are later undone and re-achieved by other actions. In this paper, we propose a compilation that extends the original planning task with commit actions that enforce the persistence of specific goals once achieved, allowing planners to identify permanent goal achievement during planning. Experimental results indicate that solving the reformulated tasks does not incur on any additional overhead both when performing optimal and suboptimal planning, while providing useful information for some downstream tasks.

1 Introduction

Automated Planning involves determining a sequence of actions, or a plan, to achieve a set of goals from an initial state (Ghallab, Nau, and Traverso 2004). Identifying the specific actions that achieve goals when solving a planning task might be beneficial for various planning applications. For example, it can be valuable for attributing goal achievement to agents in centralized multi-agent planning (Pozanco and Borrajo 2022); or for analyzing goal achievement distribution throughout the plan. However, this identification can only be done at the end of planning by analyzing the returned plan, since some actions may temporarily achieve goals that are later undone and re-achieved by other actions.

Consider the SOKOBAN task illustrated in Figure 1, where two agents are responsible for pushing stones to their designated goal locations, marked in green. One possible plan to accomplish this task involves the orange agent pushing a stone three times to the right, followed by the blue agent pushing the other stone once downward. In this plan, the first stone temporarily occupies a goal location before being moved to its final destination. Through post-processing, we can easily identify that it is not the initial action of the orange agent, but rather the final push executed by the blue agent, that successfully achieves the specific goal. However, there is currently no standard mechanism to determine at planning time, whether a goal proposition that becomes true will maintain its truth value until the plan's completion. This limitation hinders the community from developing planning solutions that not only assess goal achievement but also reason about and optimize it. In the SOKOBAN example, possessing this capability would enable us to create plans that prioritize specific goal achievement distributions over others. Examples include preferring more *fair* distributions of goal to agents; or plans that minimize the number of actions between the achievement of consecutive goals.

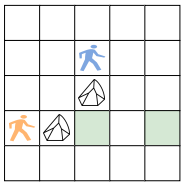

Figure 1: SOKOBAN task where two agents are responsible for pushing stones to their goal locations (in green).

In this paper, we introduce a compilation (Nebel 2000) that extends the original planning task by incorporating commit actions. These actions enable the planner to ensure the persistence of specific goals once they are achieved, thus committing to them within the search sub-tree. This approach allows the planner to determine, during the planning phase, when an action permanently achieves a goal. In the previous SOKOBAN example, the agents would have two options: either perform the standard push or execute a push-commit action, which guarantees that the stone will remain in its goal location for the remainder of the planning episode.

Experimental results across a comprehensive benchmark demonstrate that solving the reformulated tasks incurs in no additional overhead, whether in optimal or suboptimal planning. This highlights that our compilation enables reasoning about goal achievement during the planning phase without any added overhead.

2 Background

We formally define a planning task as follows:

Definition 1. *A* STRIPS ***planning task*** *is a tuple* $\mathcal{P} = \langle F, A, I, G \rangle$*, where* F *is a set of fluents,* A *is a set of actions,* $I \subseteq F$ *is an initial state, and* $G \subseteq F$ *is a goal specification.*

A state $s \subseteq F$ is a set of fluents that are true at a given time. A state $s \subseteq F$ is a goal state iff $G \subseteq s$. Each ac-

tion $a \in A$ is described by its name $\text{NAME}(a)$, a set of positive and negative preconditions $\text{PRE}^+(a)$ and $\text{PRE}^-(a)$, add effects $\text{ADD}(a)$, delete effects $\text{DEL}(a)$, and cost $c(a)$. An action a is applicable in a state s iff $\text{PRE}^+(a) \subseteq s$ and $\text{PRE}^-(a) \cap s = \emptyset$. We define the result of applying an action in a state as $\gamma(s,a) = (s \setminus \text{DEL}(a)) \cup \text{ADD}(a)$. We assume $\text{DEL}(a) \cap \text{ADD}(a) = \emptyset$. A sequence of actions $\pi = (a_1, \ldots, a_n)$ is applicable in a state s_0 if there are states $(s_1 \ldots, s_n)$ such that a_i is applicable in s_{i-1} and $s_i = \gamma(s_{i-1}, a_i)$. The resulting state after applying a sequence of actions is $\Gamma(s,\pi) = s_n$, and $c(\pi) = \sum_i^n c(a_i)$ denotes the cost of π. A state s is reachable from state s' iff there exists an applicable action sequence π such that $s \subseteq \Gamma(s', \pi)$. The solution to a planning task $\mathcal{P}$ is a plan, i.e., a sequence of actions π such that $G \subseteq \Gamma(I, \pi)$. We denote as $\Pi(\mathcal{P})$ the set of all solution plans to planning task $\mathcal{P}$. Also, given a plan π, we denote its alternatives, i.e., all the other sequence of actions that can solve $\mathcal{P}$ as $\Pi^\pi = \Pi(\mathcal{P}) \setminus \pi$. A plan with minimal cost is optimal.

3 Planning with Commit Actions

The aim of our compilation is to extend the original task with commit actions that allow the planner to enforce the persistence of specific goals once they are achieved. In particular, we will only focus on committing to those goals that are not already true in the initial state, and that can be achieved through actions in A. We make this restriction because it is not possible to tell apriori whether the goals that are true in I will need to be eventually undone to then be re-achieved by an action, or no action will need to ever falsify the goal in order to achieve G. We define pending goals as:

Definition 2. *Let $\mathcal{P} = \langle F, A, I, G\rangle$ be a planning task, the subset of **pending goals** $\bar{G} \subseteq G$ is formally defined as:*

$$\bar{G} = \{g_i \in G \setminus I \mid \exists a \in A, g_i \in \text{ADD}(a)\}$$

To ensure that goals in $\bar{G}$ remain achieved throughout the planning process, we extend the original set of propositions F with a new set of commit propositions that we denote F'. Each goal $g_i \in \bar{G}$ is associated with a commit version, which tracks whether g_i has been achieved and should remain true. Formally, we define $F' = \bigcup_{g_i \in \bar{G}} \{g_i\text{-commit}\}$.

We also extend the original actions to ensure the persistence of the commit propositions once they become true. To do that, we differentiate four different groups within A based on their interaction with the goals.

(1) Actions that add and do not delete goals: $A^G = \{a \in A \mid \bar{G} \cap \text{ADD}(a) \neq \emptyset \wedge \bar{G} \cap \text{DEL}(a) = \emptyset\}$. Since actions can achieve multiple goal propositions, for each action $a^G \in A^G$, we define C_{a^G} as a set containing all possible combinations of commit goals in $\text{ADD}(a^G) \cap \bar{G}$. This set will have 2^n subsets, where n is the number of goals in $\text{ADD}(a^G) \cap \bar{G}$. For instance, if $\text{ADD}(a^G) \cap \bar{G} = \{x, y\}$, then $C_{a^G} = \{\{\}, \{x\}, \{y\}, \{x, y\}\}$. Next, we introduce a new set of commit actions A^{C}: for each action $a^G \in A^G$, we introduce $|C_{a^G}|$ new actions. Each commit action $a_i^{\mathsf{C}} \in A^{\mathsf{C}}$, with $i \in C_{a^G}$, is defined as:

- $\text{NAME}(a_i^{\mathsf{C}}) = \text{NAME}(a^G)\text{-commit-}i$
- $\text{PRE}^+(a_i^{\mathsf{C}}) = \text{PRE}^+(a^G)$,
- $\text{PRE}^-(a_i^{\mathsf{C}}) = \text{PRE}^-(a^G) \cup \{\cup_{j \in i} g_j\text{-commit}\}$
- $\text{DEL}(a_i^{\mathsf{C}}) = \text{DEL}(a^G)$, $\text{ADD}(a_i^{\mathsf{C}}) = \text{ADD}(a^G) \cup \{\cup_{j \in i} g_j\text{-commit}\}$
- $c(a_i^{\mathsf{C}}) = c(a^G)$

(2) Actions that do not add but delete goals: $A^{\neg G} = \{a \in A \mid \bar{G} \cap \text{ADD}(a) = \emptyset \wedge \bar{G} \cap \text{DEL}(a) \neq \emptyset\}$. We must ensure these actions do not delete a goal that is already commit. For each $a^{\neg G} \in A^{\neg G}$, we substitute it with an updated forcecommit action $a^{\neg \mathsf{C}}$, which is added to a new set $A^{\neg \mathsf{C}}$ and defined as follows:

- $\text{NAME}(a^{\neg \mathsf{C}}) = \text{NAME}(a^{\neg G})\text{-forcecommit}$
- $\text{PRE}^+(a^{\neg \mathsf{C}}) = \text{PRE}^+(a^{\neg G})$
- $\text{PRE}^-(a^{\neg \mathsf{C}}) = \text{PRE}^-(a^{\neg G}) \cup \{g_j\text{-commit} \mid g_j \in \text{DEL}(a^{\neg G}) \cap \bar{G}\}$
- $\text{DEL}(a^{\neg \mathsf{C}}) = \text{DEL}(a^{\neg G})$, $\text{ADD}(a^{\neg \mathsf{C}}) = \text{ADD}(a^{\neg G})$
- $c(a^{\neg \mathsf{C}}) = c(a^{\neg G})$

(3) Actions that add and delete goals: $A^{G^\star} = \{a \in A \mid \bar{G} \cap \text{ADD}(a) \neq \emptyset \wedge \bar{G} \cap \text{DEL}(a) \neq \emptyset\}$. The behavior of the reformulated actions is defined as a combination of the previous two type of actions, and therefore they: (i) add commit propositions to achieve the goals in $\bar{G}$; and (ii) ensure that these actions do not delete a goal that is already commit. For each $a^{G^\star} \in A^{G^\star}$, we define $C_{a^{G^\star}}$ as the set of all the possible combinations of commit goals in $\text{ADD}(a^{G^\star}) \cap \bar{G}$. Next, we introduce a new set of simultaneous actions A^{S}: for each $a^{G^\star} \in A^{G^\star}$, we introduce $|C_{a^{G^\star}}|$ new actions. Each action $a_j^{\mathsf{S}} \in A^{\mathsf{S}}$, with $j \in C_{a^{G^\star}}$, is defined as:

- $\text{NAME}(a_j^{\mathsf{S}}) = \text{NAME}(a^{G^\star})\text{-simultaneous-}j$
- $\text{PRE}^+(a_j^{\mathsf{S}}) = \text{PRE}^+(a^{G^\star})$
- $\text{PRE}^-(a_j^{\mathsf{S}}) = \text{PRE}^-(a^{G^\star}) \cup \{\cup_{i \in j} g_i\text{-commit}\} \cup \{g_i\text{-commit} \mid g_i \in \text{DEL}(a^{G^\star}) \cap \bar{G}\}$
- $\text{DEL}(a_j^{\mathsf{S}}) = \text{DEL}(a^{G^\star})$, $\text{ADD}(a_j^{\mathsf{S}}) = \text{ADD}(a^{G^\star}) \cup \{\cup_{i \in j} g_i\text{-commit}\}$,
- $c(a_j^{\mathsf{S}}) = c(a^{G^\star})$

(4) Actions that neither add nor delete goals: $A^L = \{a \in A \mid \bar{G} \cap \text{ADD}(a) = \emptyset \wedge \bar{G} \cap \text{DEL}(a) = \emptyset\}$. We keep these actions unchanged.

The new planning task is defined as follows:

Definition 3. *Given a planning task $\mathcal{P} = \langle F, A, I, G\rangle$, a **commit planning task** is defined as a tuple $\mathcal{P}_{\mathsf{c}} = \langle F_{\mathsf{c}}, A_{\mathsf{c}}, I, G_{\mathsf{c}}\rangle$ where,*

- $F_{\mathsf{c}} = F \cup F'$,
- $A_{\mathsf{c}} = A^{\mathsf{C}} \cup A^{\neg \mathsf{C}} \cup A^{\mathsf{S}} \cup A^L$
- $G_{\mathsf{c}} = \{G \setminus \bar{G}\} \cup_{g_i \in \bar{G}} \{g_i\text{-commit}\}$

Theorem 1. *If $\mathcal{P}$ is solvable, $\mathcal{P}_{\mathsf{c}}$ is also solvable.*

Proof. Let π be a solution for $\mathcal{P}$. We need to show that there exists a plan π' that solves $\mathcal{P}_{\mathsf{c}}$ comprised by the same sequence of actions as π, only varying their version, i.e. we include actions that allow us to have commit and forcecommit versions. For each action $a \in \pi$, there could be these cases:

Case 1. $a \in A^L$, then π' will use a.

Case 2. $a \in A^{\neg G}$ is replaced by its corresponding forcecommit action $a' \in A^{\neg \mathsf{C}}$. They differ only in their preconditions related to commit goals. If such action appears

in π, it means there will be another action $a'' \in \pi$ that will achieve the goal that is being temporary deleted.

Case 3. $a \in A^G$. We differentiate two sub-cases: (i) a is not the last action in the plan achieving the goals in $\text{ADD}(a) \cap \bar{G}$: this action is replaced by the original non-commit action that we keep in A^{C}; and (ii) a is the last action in the plan achieving the goals in $\text{ADD}(a) \cap \bar{G}$: this action is replaced by its corresponding $a_i^{\mathsf{C}} \in A^{\mathsf{C}}$ where $i \in C_{a^G}$.

Case 4. $a \in A^{G^\star}$. We differentiate two sub-cases as Case 3 and combine the preconditions related to commit goals as Case 2.

For every $g_i \in G$, either $g_i \in I$, and is correctly achieved in $\mathcal{P}_{\mathsf{c}}$ by the same original actions in A_{c}; or $g_i \in \bar{G}$ corresponds to a commit version g_i-commit $\in G_{\mathsf{c}}$ and is correctly achieved by commit actions in A^{C}. □

Theorem 2. *Any plan π' that solves $\mathcal{P}_{\mathsf{c}}$ is a plan for the original problem $\mathcal{P}$.*

Proof. Let π' be a solution for $\mathcal{P}_{\mathsf{c}}$. We need to show that there is a plan π that can be mapped from π' which is a solution for $\mathcal{P}$. Each action in π' falls into one of four cases: (i) $a \in A^L$: we leave them as is in π; (ii) $a \in A^{\mathsf{C}}$: they are substituted in π by their counterparts in A^G; (iii) $a \in A^{\neg\mathsf{C}}$: they are substituted in π by their counterparts in $A^{\neg G}$, which are equivalent to the original actions in $\mathcal{P}$, regarding their effects. Their preconditions in $\mathcal{P}_{\mathsf{c}}$ include commit attributes, but this only restricts action applicability in $\mathcal{P}_{\mathsf{c}}$, not in $\mathcal{P}$; (iv) $a \in A^{\mathsf{S}}$: they are substituted in π by their counterparts in $A^{G^\star}$. Their preconditions in $\mathcal{P}_{\mathsf{c}}$ only restricts action applicability in $\mathcal{P}_{\mathsf{c}}$ as in (iii). Lastly, for all $g'_i \in G_{\mathsf{c}}$, we have that either $g'_i \in G$ or g'_i is the commit version of a $g_i \in G$. Then G is correctly achieved in $\mathcal{P}$. □

From Theorem 1 and 2, along with the preservation of costs from the original actions to the commit ones, we can show that the optimality of plans solving $\mathcal{P}$ is maintained in the commit task $\mathcal{P}_{\mathsf{c}}$.

Corollary 3. *If a plan π optimally solves $\mathcal{P}$, then there exists a plan π' that optimally solves $\mathcal{P}_{\mathsf{c}}$.*

Proof. Let π be an optimal solution for $\mathcal{P}$. We need to show that there is a plan π' that can be mapped from π which is an optimal solution for $\mathcal{P}_{\mathsf{c}}$.

From Theorem 1, given a solution π for $\mathcal{P}$, then there exists a solution π' for $\mathcal{P}_{\mathsf{c}}$. Since the cost of every action a' in A_{c} is preserved from the cost of each action $a \in A$, then we have that $c(\pi) = c(\pi')$. Let us now observe that if π is optimal, then π' is also optimal. Note that π being optimal means that for all alternative plans $\pi^{\mathsf{a}} \in \Pi^{\pi}$, we have that $c(\pi) \leq c(\pi^{\mathsf{a}})$. Therefore, in order to show that π' is optimal, we need to show that it does not exist π'' that solves $\mathcal{P}_{\mathsf{c}}$ and $c(\pi'') < c(\pi')$.

By contradiction, let us suppose that there exists π'' that solves $\mathcal{P}_{\mathsf{c}}$ such that $\pi'' \neq \pi'$ and π'' is an optimal plan, i.e., $c(\pi'') < c(\pi')$. By Theorem 2, there exists π''' that can be mapped from π'' and is a solution for $\mathcal{P}$. Since the costs are preserved, we have $c(\pi''') = c(\pi'')$. And giving $c(\pi''') = c(\pi'') < c(\pi') = c(\pi)$, then we have that π is not optimal. This is a contradiction from the assumption that π' is not optimal. Thus, π' is an optimal solution for $\mathcal{P}_{\mathsf{c}}$. □

Let us illustrate the compilation by considering a planning task $\mathcal{P} = \langle F, A, I, G\rangle$ where $F = \{x, y\}$, $A = \{a_1, a_2\}$, $I = \{\}$, and $G = \{x, y\}$. Actions are defined as follows: $(a_1)\text{PRE}^+ = \text{PRE}^- = \{\}, \text{DEL} = \{\}, \text{ADD} = \{x\}$; and $(a_2)\text{PRE}^+ = \{x\}, \text{PRE}^- = \{\}, \text{DEL} = \{x\}, \text{ADD} = \{y\}$. The optimal plan for $\mathcal{P}$ is $\pi = (a_1, a_2, a_1)$. Note that x becomes true after the first execution of a_1, but only permanently after its last execution, as a_2 must temporarily delete it to achieve y. Given that $a_1 \in A^G$, our compilation keeps the original action a_1, and adds a commit version (a_1-commit-x) in A^{C} such that $\text{PRE}^+ = \{\}, \text{PRE}^- = \{x\text{-commit}\}, \text{DEL} = \{\}, \text{ADD} = \{x, x\text{-commit}\}$. On the other hand, given that $a_2 \in A^{G^\star}$, the following simultaneous actions are created in A^{S}: (a_2-simultaneous-$\emptyset$) $\text{PRE}^+ = \{x\}, \text{PRE}^- = \{x\text{-commit}\}$, $\text{DEL} = \{x\}, \text{ADD} = \{y\}$; and ($a_2$-simultaneous-y) $\text{PRE}^+ = \{x\}, \text{PRE}^- = \{x\text{-commit}, y\text{-commit}\}$, $\text{DEL} = \{x\}, \text{ADD} = \{y, y\text{-commit}\}$. This gives us the following commit planning task, $\mathcal{P}_{\mathsf{c}} = \langle F_{\mathsf{c}}, A_{\mathsf{c}}, I, G_{\mathsf{c}}\rangle$ where $A_{\mathsf{c}} = \{a_1\text{-commit-}\emptyset, a_1\text{-commit-x}, a_2\text{-simultaneous-}\emptyset, a_2\text{-simultaneous-y}\}$ and $G_{\mathsf{c}} = \{x\text{-commit}, y\text{-commit}\}$. Then, there exists an optimal plan in $\mathcal{P}_{\mathsf{c}}$ that can be mapped from π, $\pi_c = (a_1\text{-commit-}\emptyset, a_2\text{-simultaneous-y}, a_1\text{-commit-x})$. In π_c, unlike in π, we can determine during the search if an action permanently achieves a goal without post-processing.

4 Evaluation

Experimental Setting. We selected all the STRIPS tasks from the optimal suite of the Fast Downward (Helmert 2006) benchmark collection[1]. This gives us 1767 tasks divided across 62 domains. We reformulated each task using our approach and solved both the original ($\mathcal{P}$) and the compiled ($\mathcal{P}_{\mathsf{c}}$) tasks using two planners. The first one, LAMAF, is a state-of-the-art planner that won the agile track of the last International Planning Competition (IPC)[2]. LAMAF runs the first iteration of the well-known LAMA planner (Richter and Westphal 2010), aiming to find a solution as soon as possible, disregarding its quality. It combines, among others: deferred heuristic evaluation, preferred operators, and multiple open lists guided by the FF (Hoffmann and Nebel 2001) and LANDMARK-SUM (Hoffmann, Porteous, and Sebastia 2004) heuristics. The second one, LMCUT, is an optimal planner that runs A^* with the admissible LMCUT heuristic. We run both planners with a 8GB memory bound and a time limit of 900s on Intel Xeon E5-2666 v3 CPUs @ 2.90GHz. We will report the following metrics, where higher numbers indicate better performance.

Coverage: number of problems solved by the planner.

SAT Score: C^*/C, where C^* is the lowest cost found for the task, and C is the plan's cost the planner gets in the task.

AGL Score: $1 - log(T)/log(900)$, where T is the planner's runtime, including grounding and search time.

[1] https://github.com/aibasel/downward-benchmarks

[2] https://ipc2023-classical.github.io/

	Coverage		SAT Score		AGL Score	
Domain (#Tasks)	$\mathcal{P}$	$\mathcal{P}_c$	$\mathcal{P}$	$\mathcal{P}_c$	$\mathcal{P}$	$\mathcal{P}_c$
agricola18 (20)	**20**	**20**	**20.0**	**20.0**	**15.5**	**15.5**
airport (50)	**34**	**34**	**34.0**	33.9	**33.0**	32.9
barman11 (20)	**20**	**20**	**20.0**	19.9	25.8	**25.9**
barman14 (14)	**14**	**14**	**14.0**	13.9	**17.2**	17.1
blocks (35)	**35**	**35**	22.7	**35.0**	56.4	**56.8**
childsnack14 (20)	**15**	**15**	**15.0**	**15.0**	**20.0**	19.9
data-net18 (20)	**20**	**20**	**20.0**	**20.0**	**29.6**	29.5
depot (22)	20	**21**	18.8	**20.1**	21.3	**22.4**
driverlog (20)	**20**	**20**	17.8	**19.5**	26.7	**30.7**
elevators08 (30)	**30**	**30**	28.3	**28.4**	**48.1**	47.9
elevators11 (20)	**20**	**20**	18.8	**19.2**	**32.4**	32.3
floortile11 (20)	**9**	**9**	**8.7**	8.6	**6.1**	5.8
floortile14 (20)	**9**	8	**8.7**	7.1	**2.9**	2.5
freecell (80)	77	**79**	74.7	**75.4**	81.0	**83.1**
ged14 (20)	**20**	**20**	**20.0**	19.6	**32.2**	31.9
grid (5)	**5**	**5**	4.9	**5.0**	**6.5**	**6.5**
gripper (20)	**20**	**20**	**20.0**	**20.0**	**32.9**	32.3
hiking14 (20)	**20**	**20**	**20.0**	**20.0**	**24.0**	**24.0**
logistics00 (28)	**28**	**28**	27.6	**27.7**	**46.4**	46.2
logistics98 (35)	**35**	34	**34.9**	33.7	**41.0**	40.9
miconic (150)	**150**	**150**	**150.0**	**150.0**	**231.6**	231.3
movie (30)	**30**	**30**	26.2	**30.0**	54.5	**54.6**
mprime (35)	**35**	**35**	**35.0**	**35.0**	**47.2**	**47.2**
mystery (30)	**19**	**19**	18.9	**19.0**	**25.7**	25.6
nomystery11 (20)	**16**	**16**	**16.0**	**16.0**	**22.5**	22.3
openstacks08 (30)	**30**	**30**	20.6	**30.0**	**45.9**	43.9
openstacks11 (20)	**20**	**20**	13.1	**20.0**	**30.5**	29.0
openstacks14 (20)	**20**	**20**	12.2	**20.0**	**26.4**	24.5
openstacks (30)	**30**	**30**	29.8	**30.0**	**36.0**	34.8
org-syn18 (20)	**7**	6	**7.0**	6.0	**9.2**	5.3
org-syn-spl18 (20)	**18**	11	**18.0**	11.0	**17.7**	12.0
parcprinter08 (30)	**30**	21	**28.6**	21.0	**47.0**	27.4
parcprinter11 (20)	**20**	15	**19.0**	15.0	**31.5**	17.7
parking11 (20)	**20**	18	**18.9**	15.6	**20.2**	12.3
parking14 (20)	**20**	18	**19.8**	15.3	**21.0**	13.8
pathways (30)	**23**	**23**	**23.0**	22.9	**32.3**	**32.3**
pegsol08 (30)	**30**	29	26.2	**28.5**	**43.2**	34.1
pegsol11 (20)	**20**	19	18.1	**18.7**	**26.6**	19.7
pipes-notank (50)	**43**	**43**	33.1	**40.6**	47.4	**53.8**
pipes-tank (50)	**43**	37	**35.6**	35.4	**41.4**	39.6
psr-small (50)	**50**	**50**	**50.0**	49.1	**87.0**	86.0
rovers (40)	**40**	**40**	**40.0**	**40.0**	**54.1**	**54.1**
satellite (36)	**36**	**36**	33.4	**35.9**	41.7	**42.8**
scanalyzer08 (30)	**30**	**30**	**27.9**	27.4	**36.8**	34.1
scanalyzer11 (20)	**20**	**20**	**18.8**	17.9	**25.0**	23.5
sokoban08 (30)	**29**	**29**	26.4	**27.0**	**32.4**	30.4
sokoban11 (20)	**20**	**20**	**18.7**	18.5	**22.8**	21.6
spider18 (20)	**19**	16	**17.8**	15.4	**14.3**	8.8
storage (30)	19	**21**	16.9	**19.9**	26.5	**27.7**
tetris14 (17)	**17**	10	**15.3**	9.3	**13.1**	9.1
tidybot11 (20)	**18**	**18**	17.8	**17.9**	15.5	**15.7**
tidybot14 (20)	**19**	**19**	**19.0**	18.8	**13.5**	13.3
tpp (30)	**30**	**30**	**30.0**	**30.0**	**38.4**	38.0
transport08 (30)	**30**	**30**	**29.9**	29.7	**42.8**	42.6
transport11 (20)	**20**	**20**	**20.0**	19.9	**29.7**	29.5
transport14 (20)	**20**	**20**	19.9	**20.0**	**28.1**	28.0
trucks (30)	15	**17**	15.0	**17.0**	16.1	**17.8**
visitall11 (20)	**20**	**20**	**20.0**	16.4	**33.1**	32.4
visitall14 (20)	**20**	**20**	**20.0**	16.5	**29.1**	27.9
woodwork08 (30)	**30**	**30**	**29.9**	27.6	**40.9**	36.5
woodwork11 (20)	**20**	**20**	**19.9**	18.7	**26.8**	24.0
zenotravel (20)	**20**	**20**	**19.9**	19.3	**29.7**	29.6
Total (1767)	**1637**	1598	1544.3	**1554.3**	**2154.1**	2058.8

Table 1: LAMAF Coverage, SAT and AGL scores when solving the original ($\mathcal{P}$) and compiled ($\mathcal{P}_c$) tasks. Bold figures indicate best performance.

	Coverage		SAT Score		AGL Score	
Domain (#Tasks)	$\mathcal{P}$	$\mathcal{P}_c$	$\mathcal{P}$	$\mathcal{P}_c$	$\mathcal{P}$	$\mathcal{P}_c$
airport (50)	**28**	**28**	**28.0**	**28.0**	33.3	**33.4**
blocks (35)	**28**	**28**	**28.0**	**28.0**	35.8	**36.6**
elevators11 (20)	**18**	**18**	**18.0**	**18.0**	**10.7**	10.3
floortile14 (20)	**6**	5	**6.0**	5.0	**1.9**	1.6
gripper (20)	**7**	5	**7.0**	5.0	**7.1**	5.5
logistics00 (28)	**20**	**20**	**20.0**	**20.0**	**23.7**	23.2
openstacks08 (30)	**21**	20	**21.0**	20.0	**17.7**	16.0
org-syn-spl18 (20)	**15**	11	**15.0**	11.0	**13.4**	10.3
pegsol11 (20)	**18**	11	**18.0**	11.0	**15.8**	3.3
rovers (40)	**7**	**7**	**7.0**	**7.0**	**9.9**	9.8
sokoban11 (20)	**20**	**20**	**20.0**	**20.0**	**18.3**	17.0
tetris14 (17)	**6**	3	**6.0**	3.0	**3.3**	2.2
trucks (30)	**10**	**10**	**10.0**	**10.0**	**9.0**	8.8
woodwork11 (20)	**12**	11	**12.0**	11.0	**10.3**	8.8
Total (1767)	**917**	887	**917.0**	887.0	**1006.2**	943.2

Table 2: LMCUT Coverage, SAT and AGL scores when solving the original ($\mathcal{P}$) and compiled ($\mathcal{P}_c$) tasks.

Results. Tables 1 and 2 show the Coverage, SAT and AGL scores when solving the original ($\mathcal{P}$) and the compiled ($\mathcal{P}_c$) tasks with LAMAF and LMCUT, respectively. Due to space constraints, we present only a subset of the results for LMCUT. Domains with problems where goal propositions cannot be falsified once achieved are highlighted in gray. These tasks do not require reformulation into commit planning tasks, but we aimed to verify the general applicability of our compilation across different problem structures. As observed, both planners achieve slightly better overall results when solving $\mathcal{P}$, attaining higher scores across most metrics compared to $\mathcal{P}_c$. The performance gap varies by domain and planner, yet $\mathcal{P}_c$ tasks can always be solved comparably to $\mathcal{P}$, both optimally and suboptimally. This holds true even in domains such as OPENSTACKS or ROVERS, where our commit compilation is unnecessary due to the permanence of goals. This emphasizes the non-intrusive nature of our compilation, providing the advantage of reasoning about goal achievement during the planning phase without compromising search efficiency.

5 Conclusions and Future Work

We introduced a compilation that extends the original planning task with commit actions that allow planners to enforce the persistence of specific goals once achieved, thus committing to them in the search sub-tree. This new task structure allows us to determine at planning time when an action permanently achieves a goal. This opens up research opportunities for reasoning and optimizing goal achievement distribution, potentially benefiting various planning applications.

Certain domains, such as BLOCKS, consistently benefit from our reformulation, indicating that allowing planners to greedily commit to achieving specific goals may enhance search performance in certain situations. In other domains such as OPENSTACKS, performance improvements appear to stem from an internal state representation that is more compatible with the specific planner being used (Vallati et al. 2015). We intend to further investigate this phenomenon and its implications as part of our future work.

Disclaimer

References

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated planning - theory and practice*. Elsevier.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

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

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *Journal of Artificial Intelligence Research* 22:215–278.

Nebel, B. 2000. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research* 12:271–315.

Pozanco, A., and Borrajo, D. 2022. Fairness in multi-agent planning. *arXiv preprint arXiv:2212.00506*.

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

Vallati, M.; Hutter, F.; Chrpa, L.; and McCluskey, T. L. 2015. On the effective configuration of planning domain models. In *International Joint Conference on Artificial Intelligence (IJCAI)*. AAAI press.