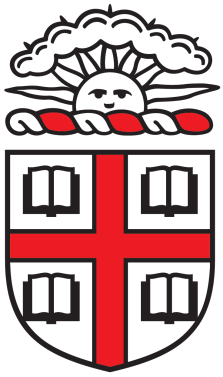
BROWN UNIVERSITY



UNDERGRADUATE HONORS THESIS

You Only Need What's in Scope: Generating Task-Specific Abstractions for Efficient AI Planning

---

Author:
Nishanth KUMAR

Advisor and First Reader:
Dr. Stefanie TELLEX
Second Reader:
Dr. George KONIDARIS

A thesis submitted in partial fulfillment of the requirements for the degree of Bachelors of Science with Honors in Computer Engineering

in the

School of Engineering, Brown University

September 10, 2021

Author's Note

Much of this work was originally documented in Kumar et al. (2020), a version of which is in-submission to an archival venue. Additionally, this work was performed in collaboration with Michael Fishman, Natasha Danas and Professors Michael Littman, Stefanie Tellex and George Konidaris within Brown University's Department of Computer Science.

BROWN UNIVERSITY

Abstract

You Only Need What's in Scope: Generating Task-Specific Abstractions for Efficient AI Planning

by Nishanth KUMAR

Many real-world planning domains of interest are *open-scope*: they contain enough information to support planning to one of many possible goals, but much of the state-action space is irrelevant to computing an optimal plan for any particular goal. Planning to most goals directly within such domains is often intractable because of their unnecessarily large state and action spaces. However, planning within an abstraction of the domain, where irrelevant states and actions have been removed, is often much more feasible and efficient. We propose *task scoping*: a method that exploits knowledge of the initial state, goal conditions, and transition system to automatically and efficiently prune provably irrelevant fluents and actions from a planning problem. We empirically evaluate task scoping on a variety of domains and demonstrate that its use as a pre-planning step can reduce the state-action space by orders of magnitude, accelerating planning for both state-of-the-art classical and numeric planners. When applied to a complex Minecraft domain, task scoping speeds up a state-of-the-art planner by more than $30\times$ (including scoping time).

Contents

Dedicated to the faculty, staff, and students of BigAI, for changing my life in many wonderful ways.

Chapter 1

Introduction

Modern planning systems have successfully solved a variety of individual and important tasks, such as logistics (Refanidis, Bassiliades, and Vlahavas, 2001) and chemical synthesis (Segler, Preuss, and Waller, 2018). However, while such tasks have immediate practical importance and economic value, planning must be made much more efficient if it is to form the core computational process of a generally intelligent agent because such agents are expected to perform a wide variety of tasks within large, rich worlds. In particular, generally intelligent agents must maintain world models that are *open-scope*: rich enough to describe any task the agent may be asked to solve, thus necessarily including large amounts of information irrelevant to any *individual* task (Konidaris, 2019). For instance, an agent that may be asked to solve any task in Minecraft must possess a model containing information necessary to craft various weapons to kill enemies, build shelter, and obtain and cook food items; however when confronted with the specific, immediate goal of crafting a bed, information about shelter and food is simply irrelevant. When confronted with the different goal of cooking a stew, a completely different set of information is rendered irrelevant.

Recent work (Vallati and Chrpa, 2019; Silver et al., 2021) has shown that many state-of-the-art planning engines—even the renowned Fast Downward Planning System (Helmert, 2006), which attempts to prune the problem's state-action space before search—suffer significant reductions in performance when irrelevant objects, fluents or actions are included in domain descriptions. For example, the state-of-the-art ENHSP-2020 planner (Scala et al., 2016) takes over 57 minutes to find an optimal plan for a Minecraft domain pictured in Figure 1.1, in large part because it must search through approximately 10^{135} states; simply deleting model components irrelevant to this specific task shrinks the state space by over 100 orders of magnitude, allowing it to be solved in just over 3.5 minutes. Generally-intelligent agents with rich, open-scope world models will therefore likely require a means of ignoring the vast majority of information in those models on a task-specific basis.

We characterize open-scope domains, identify and characterize different types of task-irrelevance that occurs within such domains and propose a novel algorithm, *task scoping*, that exploits knowledge of a problem's initial condition, goal condition, and transition dynamics to prune particular types of irrelevant fluents and actions without compromising the existence of optimal plans. This pruning process operates at the level of the problem definition instead of the concrete state-action space, making first pruning, then planning often substantially faster than planning in the original space. We prove that the resulting abstraction is sound and complete: all valid plans in the scoped domain are valid in the original and pass through the same abstract states, and all optimal plans in the original domain can be translated into an optimal plan in the scoped domain. Additionally, we show that task scoping can have better worst-case computational complexity than planning for problems with particular features and structure.

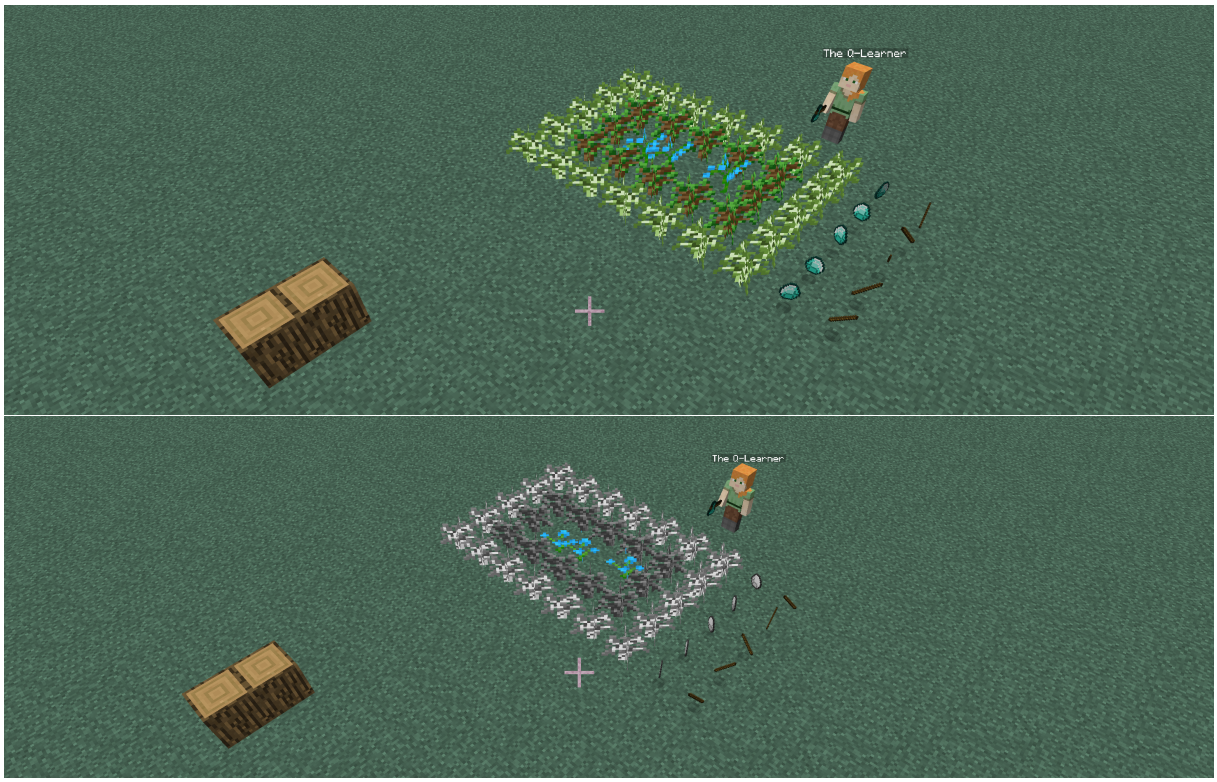


FIGURE 1.1: An open-scope Minecraft environment before (top) and after (bottom) scoping removes irrelevant objects (visualized as greyed out)

We argue that many planning domains of interest possess such characteristics, and thus, scoping followed by planning will often be more computationally efficient than planning directly. Task scoping can thus be viewed as a planner-agnostic pre-processing step that computes a relevance-based abstraction.

We empirically demonstrate task scoping's efficacy on several domains from past iterations of the International Planning Competition (IPC) and on novel domains that we developed. Our results demonstrate that task scoping can substantially reduce the state-action spaces of various planning tasks specified in PDDL 2.1 or SAS+ (Helmert, 2006), and that the resulting abstraction varies appropriately depending on the task. Moreover, using task scoping as a pre-processing step often significantly reduces the time taken by the state-of-the-art ENHSP-2020 and Fast Downward planners to solve these large problems, especially for our open-scope Minecraft domain.

Chapter 2

Background

2.1 Planning Problems

Any deterministic, single-agent planning problem consists of some set of states, some set of actions the agent can take, some transition dynamics that describe how actions affect states, an initial state that is true at the start of the problem, and some goal conditions that the agent must make true to solve the problem. Such planning problems are often expressed in PDDL (Fox and Long, 2003), which uses an object-oriented format consisting of separate 'domain' and 'problem' files. The domain file defines types of objects that exist within the problem, their properties and how an agent can influence these through actions at an abstract level (similar to defining a class within an object-oriented programming language). The problem file instantiates some specific objects and uses these to specifically define the initial state and goal conditions (similar to instantiating a class within an object-oriented programming language).

Instead of maintaining the distinction between domain and problem files, we choose to operate on *grounded* problems, where the domain file is instantiated - or *grounded* - to the specific objects defined in the problem file. This yields grounded state-variables — also known as *fluents* — that can take on values from some domain, as well as grounded actions that depend on and affect specific grounded fluents.

As a running example of such a planning problem, consider a version of the continuous playroom domain from Chentanez, Barto, and Singh (2005) and pictured in Figure 2.1. An agent controls 3 effectors (an eye, a hand, and a marker) to interact with 7 objects (a thermostat, a light switch, a red button, a green button, a ball, a bell, and a monkey). The domain can be discretized into a grid where the agent can take an action to move its effectors in the cardinal directions. To interact with the thermostat, light switch or buttons, the agent's eye and hand effectors must be at the same grid cell as the relevant object. The thermostat can be set to 1 of 5 different temperatures that do not impact other dynamics of the problem whatsoever. The light switch can be turned on and off to toggle the playroom's lights, and, when the lights are on, the green button can be pushed to turn on music, while the red button can be pushed to turn off music. Once the music is on, regardless of the state of the lights, the agent can move its eye and hand effectors to the ball and its marker effector to the bell to throw the ball at the bell and frighten the monkey. While this is the domain's original goal, it is possible to specify other goals such as turning the music on, or turning the lights off.

Within this problem, $\texttt{x} - \texttt{position}(\texttt{hand1}) \in \{1.0,1.01,1.02,....,10.0\}$ and $\texttt{lights} - \texttt{on}(\texttt{light} - \texttt{switch1}) \in \{\text{True, False}\}$ are examples of grounded-state variables. $\texttt{move} - \texttt{hand} - \texttt{right}(\texttt{hand1})$ and $\texttt{turn} - \texttt{off} - \texttt{light} - \texttt{switchlight} - \texttt{switch1}$ are examples of actions that have corresponding pre-conditions in which they are applicable (such as $\texttt{lights} - \texttt{on}(\texttt{light} - \texttt{switch1}) == \texttt{True}$ for the second action) and effects (such as $\texttt{lights} - \texttt{on}(\texttt{light} - \texttt{switch1}) = \texttt{False}$). The initial state might specify that

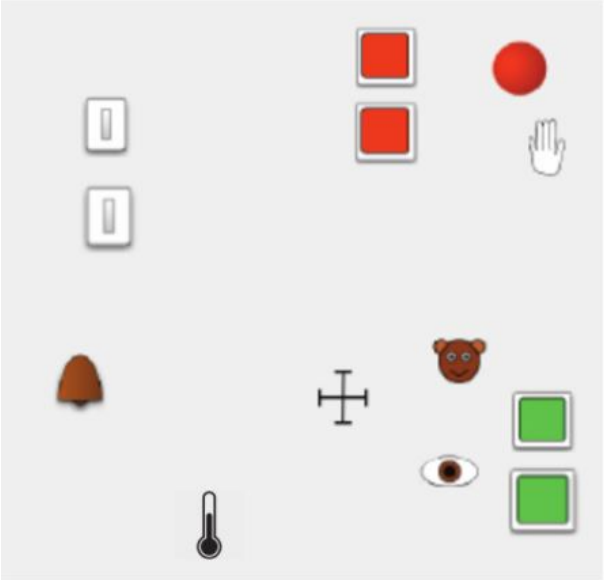


FIGURE 2.1: A visualization of our running example: the Modified Continuous Playroom Domain

$\texttt{x} - \texttt{position}(\texttt{hand1}) = 1.0$ and that $\texttt{lights} - \texttt{on}(\texttt{light} - \texttt{switch1}) = \texttt{True}$, while the goal condition might be $\texttt{lights} - \texttt{on}(\texttt{light} - \texttt{switch1}) == \texttt{True}$.

2.1.1 Planning Formalism

More formally, we assume a grounded planning formalism equivalent to PDDL 2.1 level 2 (Fox and Long, 2003) without conditional effects. This formalism subsumes SAS+ problems without axioms (Helmert, 2006). We define a planning problem as a tuple $PP = (S[\mathcal{J}], \mathcal{A}, \mathcal{T}, s_0, G)$, where $S[\mathcal{J}]$ is the state space, $\mathcal{J}$ is a set of grounded fluents[1] where each element is either numeric or propositional, $\mathcal{A}$ is a set of actions, $\mathcal{T}$ represents the transition dynamics, s_0 is an initial state, and G is a goal condition.

- $S = \prod_{j \in \mathcal{J}} S[j]$, where $S[j]$ is the set of values that fluent j can take. We consider a specific state $s \in S$ as a vector $\langle s_1, ..., s_{|\mathcal{J}|} \rangle$ with a value assigned to each fluent (i.e, a rational number to every numeric fluent, and a truth value to every propositional fluent). If $J \subseteq \mathcal{J}$, then $S[J] := \prod_{j \in J} S[j]$ refers to the projection abstraction (discussed in Section 2.2) including only the fluents mentioned in J, meaning that all *other* (non J) fluents' values are ignored.
- $\mathcal{T}$ is a set of (*pre-Condition; Action; Effect*) (or *CAE*) triples, where
  - *pre-Condition* is a predicate over $S[J]$ for some $J \subseteq \mathcal{J}$. We assume these preconditions are specified such that they are mutually exclusive for any pair of CAEs with the same action $a \in \mathcal{A}$. This choice ensures that only *one* CAE triple's effects can occur at any time step,
  - *Action* $\in \mathcal{A}$,
  - *Effect* is a list of deterministic functions $\langle F : S[j] \to S[j] \rangle_{j \in \mathcal{J}}$ describing how actions change the values of each of the J fluents.
- s_0 is an initial state.

[1] All fluents considered in this work are grounded, so we will henceforth simply use the term 'fluents'.

- G is a conjunction of predicates over S.

An action is applicable in a state s where the pre-condition of the corresponding CAE triple is satisfied. Applying an action in s results in a new state s' where the effect of the corresponding CAE triple has been applied to s. Applying a sequence of successively-applicable actions $< a_0, a_1, ..., a_n >$ from the initial state s_0 results in a final state s_{n+1}. If the goal condition G is true in this final state, then the action sequence (or *trace*) is a *valid plan*. Furthermore, we assume each action has some non-negative cost. Thus, we can define an *optimal plan* as a valid plan with the minimum cost.

2.2 Abstractions for Planning

The size of a planning problem can be measured by the state-action space, which is simply the product of the domains of all grounded state-variables and the number of abstract or grounded actions. Given this, an abstraction can be understood as a function that transforms a planning problem into another problem of a smaller size in order to solve it more easily (Hoffmann et al., 2006). There are many different types of abstractions (Sacerdoti, 1974; Seipp and Helmert, 2018; Culberson and Schaeffer, 1998), each with different advantages, disadvantages and intended purposes. In this work, we will consider a relatively simple kind of abstraction termed a *projection abstraction* (Seipp and Helmert, 2018; Culberson and Schaeffer, 1998) whereby the abstract problem is formed by simply ignoring specific fluents and actions from the original problem. This is analagous to the more-familiar notion of projection from geometry. In our case, we will attempt to discover a projection abstraction that preserves only the relevant fluents and actions for a large, open-scope planning problem.

Chapter 3

Task Scoping

3.1 Open-Scope Planning Problems

We begin this chapter by attempting to characterize open-scope planning problems and why they are difficult for existing methods to solve. A precise, formal definition of open-scope planning problems is both difficult to provide and unnecessary to the following sections. Instead, we sketch the characteristics of such problems below.

We refer to a planning problem as *open-scope* if:

1. It is possible to specify many different goals, each involving different grounded fluents. Note that the ability to specify different goals by specifying different value assignments to the same set of fluents does not meet this condition.
2. For any particular goal and some initial state, there exists a projection (or finer) abstraction that preserves all optimal plans (i.e, the abstraction is *sound* and *optimality-complete* as described in Section 3.3).

Our running example continuous playroom domain features both of the above-described characteristics. There are a variety of goals involving different fluents that can be specified; these include frightening the monkey, turning the music on or off, turning the lights on or off, setting the temperature to a specific value or moving any of the agents effectors to some particular grid locations. Furthermore, for any particular goal there are a variety of initial states under which a planner can use the right projection abstraction to effectively ignore specific subsets of fluents and actions. For instance, the thermostat can be projected away when attempting to plan to frighten the monkey, the lightswitch can be projected away when attempting to turn on the music *if* the lights are already on in the initial state, etc.

Most existing methods attempt to perform some heuristic-guided search on the problem. Such approaches struggle to find optimal plans for these open-scope problems because of characteristic 2: the large number of irrelevant fluents and actions results in a massive branching factor during search. Furthermore, given the nature of characteristic 2, these fluents and actions can be safely ignored during planning, resulting in a massively-reduced branching factor for search. This gives us some motivation to discover such an abstraction in a computationally more efficient manner than heuristic-guided search, which is what we will do in the following sections.

3.2 Reduced Planning Problems

Let PP be a planning problem with fluents $\mathcal{J}$. Suppose some subset of these fluents $J^c \subseteq \mathcal{J}$ are *deleted*; there is now a unique *reduced planning problem* $PP_r[J]$ induced by this subset

$J \subseteq \mathcal{J}$ where $J \cup J^c = \mathcal{J}$. $PP_r[J]$ is a projection abstraction. We denote the states, actions, transition dynamics, initial state and goal conditions for the reduced problem as $PP_r[J] = (S[J], \mathcal{A}[J], \mathcal{T}[J], s_0[J], G[J])$.

The initial and goal conditions remain the same except that clauses mentioning fluents in J^c are deleted since they are no longer meaningful.[1] $A[J]$ is the set of actions used by $T[J]$.

Returning to our running example, suppose we decide to ignore the `temperature(thermo1)` fluent. $S[J]$ would now simply be formed by all the remaining fluents. $s_0[J], G[J]$ would be formed by simply deleting all assignments and conditions that mention this particular fluent (since the concept of temperature does not exist anymore).

The transition dynamics $\mathcal{T}[J]$ — and consequently, the actions $\mathcal{A}[J]$ — are not as straightforward to derive. They are induced by these J fluents as described below.

3.2.1 Induced Transition Dynamics.

Two CAE triples $x, x' \in \mathcal{T}$ are *equivalent with respect to* J, denoted $x \simeq_J x'$, if, for each fluent in J, x and x' have the same effect: $x.\text{effects}[J] = x'.\text{effects}[J]$. and $x.\text{action} = x'.\text{action}$.

In the RPP induced by J, we do not distinguish CAEs with identical effects on J. We obtain the transition dynamics on the RPP induced by J, denoted $\mathcal{T}[J]$, by discarding all CAEs with no effect on J, partitioning the remaining set of CAEs according to $\simeq_J$, and then creating from each part class X a new *quotient CAE* $\overline{x}$:

$$\overline{x} = \left(\bigvee_{x' \in X} x.\text{precondition}, \bigcup_{x' \in X} \{x.\text{action}\}, x.\text{effects}[J] \right).$$

When discussing the task scoping algorithm in Section 3.4, we will also refer to $\overline{x}$.sideeffects—the set of fluents *not* in J that may be affected when executing $\overline{x}$:

$$\overline{x}.\text{sideeffects} = \bigcup_{x' \in X, j \in J^c} \{x'.\text{effects}[j]\}).$$

Note that the side effects are *not* included in the returned RPP, they are simply a book-keeping tool used in the task scoping algorithm. When writing out a quotient CAE, the side effects may be included as the fourth component.

To illustrate this, consider two different CAE triples within our running example: one that moves the agent's `(agent1)` hand north and one that does the same while also flicking the thermostat `(thermo1)` if it is in the way.

- `((hand-y agent1) != (thermostat-y thermo1); move_north_hand; (hand-y agent1)++)`
- `((hand-y agent1) == (thermostat-y thermo1); move_and_flick; (hand-y agent1)++ and (temperature)++)`

Suppose that we want to compute the induced CAEs for these two CAEs and `(hand-y agent1)` is within the subset J while (temperature) is not. Since these two triples differ *only* in their effects on variables in J^c, they would be merged to produce a single triple with a *side effect* on temperature.[2]

[1] If $J^c = \emptyset$, the RPP is the same as the original planning problem.

[2] '(x) ++' is short for (increase (x) 1) (Fox and Long, 2003).

- `(True; move_north; (hand-y agent1)++; temperature)`

The precondition of this CAE triple is simply *True*, as a result of having taken the disjunction of the previous triples' preconditions. Both triples affect the agent's hand in the same way, regardless of whether it is at the thermostat.

Pseudocode for ReduceTransitions, a procedure that computes these induced transition dynamics, is given below.

Algorithm 1 Full pseudocode for the ReduceTransitions algorithm

procedure REDUCETRANSITIONS($\mathcal{T}$, J)
  $U \leftarrow$ Partition of $\mathcal{T}$ based on $\simeq_J$
  Discard from U each part that does not effect $S[J]$
  $\mathcal{T}[J] \leftarrow \{\}$
  for $X \in U$ **do**
    $\overline{x} \leftarrow \left(\bigvee x.\text{prec}, \bigcup_{x' \in X} \{x.\text{action}\}, x.\text{effects}[J], \bigcup_{x' \in X} \text{vars}(x'.\text{effects}[J^c]) \right)$
    $\mathcal{T}[J]$.insert($\overline{x}$)
  return $\mathcal{T}[J]$

3.3 Sound and optimality-Complete RPPs.

Given the above defnitions, it is possible to construct an RPP by deleting *any* combination of fluents from *any* planning problem. Intuitively, not all such RPPs will be useful abstractions for planning. In light of this, we wish to construct RPPs that are *sound* and *optimality-complete*.

Definition 3.3.1. An RPP is *optimality-complete* if an optimal solution to the original planning problem can be obtained as a solution to the reduced planning problem, or the original planning problem has no solutions.

Definition 3.3.2. An RPP is *sound* if each plan in the RPP gives the same trace with respect to the abstract state space (i.e, the J fluents) when executed in the original planning problem.

We call a sound and optimality-complete RPP an SC-RPP.

3.3.1 Types of Irrelevance

To find sufficient conditions for an RPP to be sound and optimality-complete, we first define what it means for a fluent to be irrelevant with respect to an optimal plan and distinguish between a few types of irrelevance.

Definition 3.3.3. A fluent is *irrelevant* for a specific planning problem if projecting it away results in a reduced planning problem whose optimal plans are also optimal for the original problem.

Definition 3.3.4. A fluent is *trivially irrelevant* if it is irrelevant for any initial state.

Definition 3.3.5. A fluent is *conditionally irrelevant* if it is irrelevant for the specified initial state, but may be relevant for a different initial state.

In our running example, the room's temperature is a trivially irrelevant fluent because the optimal plan is the same regardless of the value of the `(temperature)` for any initial state. Additionally, the lights and music are conditionally irrelevant to the goal of frightening the monkey, because they can be projected away without changing the optimal plan (but only if the music is already on in the initial state).

Definition 3.3.6. A *causally-linked* fluent maintains its original value throughout all optimal plans, and arbitrarily changing its value while executing a valid trace may increase the cost of optimal plans. This term is adapted from `UCPOP` (Penberthy and Weld, 1992).

Definition 3.3.7. A fluent is *causally-masked* if there is a set of causally-linked fluents such that, as long as each of these causally-linked fluents maintains its initial value throughout the trace, the causally-masked fluent can vary arbitrarily without impacting the quality of optimal plans.

By *vary arbitrarily*, we mean changed without the agent taking an action. In our running continuous playroom example, the the status of the lights is causally-masked by the music being on, which is itself causally-linked.

3.3.2 Scoped RPPs

Given the above definitions of irrelevance, we can now define and describe a *scoped* RPP. We will show that scoped RPP's are SC-RPP's under certain conditions, and introduce an algorithm to efficiently compute such RPP's from the definition of a planning problem.

Definition 3.3.8. For a given RPP, translate the preconditions in $\mathcal{T}[J]$ to conjunctive normal form (CNF), and let Φ be the set of clauses appearing in any of these preconditions. If, for each $\phi \in \Phi$, either:

1. ϕ is defined over $S[J]$, or
2. ϕ is true in s_0, and none of the fluents mentioned in ϕ are in the side effects of $\mathcal{T}[J]$,

then the RPP is *scoped*.

Suppose we are given a scoped RPP and the $J \subseteq \mathcal{J}$ fluents used to induce it. $\mathcal{J}$ can be partitioned into 3 distinct, non-overlapping sets:

- $J = J_{rel}$, the fluents satisfying (1) above. We call these fluents *relevant*.
- J_{CL}, the fluents mentioned in any ϕ satisfying (2) above. These are *causally linked*.
- J_{irrel}, all other fluents. These fluents are either trivially irrelevant or causally-masked.

By the above definitions, the preconditions of the CAE triples within $\mathcal{T}[J]$ only mention fluents within $S[J_{rel}] \cup S[J_{CL}]$. For any precondition C of a CAE triple in a scoped RPP, we can decompose C as the conjunction of a clause defined over $S[J_{rel}]$ and a clause defined over $S[J_{CL}]$:

$$C = C[J_{rel}] \wedge C[J_{CL}],$$

where $s_0[J_{CL}] \implies C[J_{CL}]$.

3.3.3 Theoretical Results

Given the definition for scoped RPP's, we can show that such RPP's are SC-RPP's if the goal fluents are contained in $J_{rel} \cup J_{CL}$.

Theorem 1. A scoped RPP is sound: an initial state and a sequence of $\overline{x}$ from a scoped RPP can be lifted to a sequence of x from the original PP, and these two sequences lead to the same sequence of states in $S[J_{rel}]$.

Proof. True by construction. □

Lemma 1. Given a scoped RPP, there is no CAE triple in $\mathcal{T}[J]$ that affects both J_{rel} and J_{CL}.

Proof. Suppose J_{rel} induces an SC-RPP, and $x \in \mathcal{T}$ affects both $j \in J_{rel}$ and $j' \in J_{CL}$. Then, $\overline{x} \in \mathcal{T}[J]$ affects j and has a side effect on j', contradicting Condition 2 from the definition of J_{CL} in an SC-RPP. □

Theorem 2. A scoped RPP is optimality-complete if it contains each goal fluent in $J_{rel} \cup J_{CL}$.

Proof. Suppose we have a scoped RPP. We will map each concrete trace to an abstract trace of equal length that passes through the same states when projected onto J_{rel}, and conclude that the RPP is complete. Let τ be a trace in the concrete PP, with $\tau.a = (a_0, a_1, \ldots, a_n)$ be a sequence of actions that can be executed when starting from the initial state s_0 and $\tau.s = (s_0, s_1, \ldots, s_{n+1})$ the the sequence of states obtained by taking $\tau.a$ from s_0. Let $\tau.x = (x_0, x_1, \ldots, x_n)$ be the sequence of CAE triples corresponding to taking a_i from s_i for each i.

Let the i^{th} state or action be indicated with a subscript, eg. $\tau_i.a$. Let *non-affecting action* refer to any action that has no effects on $S[J_{rel}]$.

Let $\tau_i.x[J_{rel}]$ be the abstract CAE triple induced by $\tau_i.x$ on J_{rel}. Let $\overline{\tau}$ be the trace obtained by replacing each non-affecting action in τ with a no-op.

We will show that, for each i, either $\tau_i.x$ is non-affecting or that the effects of $\tau_i.x$ and $\overline{\tau}_i.x$ on the J_{rel} fluents are the same. We will then conclude that $\tau.s = \overline{\tau}.s$

Suppose now that $\tau_i.x$ is the first CAE triple in τ that affects any of the $[J_{rel}]$ fluents.
Since we only took non-affecting actions prior to $\tau_i.x$, we know that $\tau_i.s[J_{rel}] = \overline{\tau}_i.s[J_{rel}]$. We also know that $\overline{\tau}_i.x$.precondition can be written as $C[J_{rel}] \wedge C[J_{CL}]$ by definition of our abstracted transition dynamics.
By Lemma 1, only non-affecting actions can affect fluents J_{CL}, so replacing non-affecting actions with no-ops to produce $\overline{\tau}$ guarantees that the state $\overline{\tau}_i[s]$ has the same values for the J_{CL} fluents as the initial state $s_0 = \bar{s}_0$, and so $\overline{\tau}_i[s][J_{CL}] = s_0[J_{CL}]$, so $C[J_{CL}]$ is true.

Thus, by Property 2 of an SC-RPP, $\overline{\tau}_i.x$ is executable from both $\tau_i.s$ and $\overline{\tau}_i.s$ and thus the action $\tau_i.a$ must have the same effect on J_{rel}. Therefore, $\tau_{i+1}.s[J_{rel}] = \overline{\tau}_{i+1}.s[J_{rel}]$.

We have just shown that if we assume $\tau_i.a$ is the first affecting action, then all actions upto $\tau_i.a$ can be replaced by no-ops and $\tau_{i+1}.s[J_{rel}] = \overline{\tau}_{i+1}.s[J_{rel}]$. Given this, suppose now that the next affecting action is $\tau_j.a$, where $j > i$. We can apply the exact same argument to show that $\tau_{j+1}.s[J_{rel}] = \overline{\tau}_{j+1}.s[J_{rel}]$. We can continue to do this for every subsequent affecting action after j. □

3.4 The Task Scoping Algorithm

We can now define an algorithm that produces a scoped RPP given a planning problem and thereby provably supports optimal planning. Algorithm 2 contains the full pseudo-code for our task scoping algorithm. Intuitively, the algorithm begins by assuming that the only relevant fluents are the goal fluents that are not causally linked. It then calls ReduceTransitions to create

an RPP that contains *only* these fluents. If this RPP is not a scoped RPP, then there must exist at least one fluent mentioned in the preconditions of the reduced CAE triples that is not causally linked. The algorithm adds all such fluents to the set of relevant fluents (J_{rel}) and continues this process until a scoped RPP is returned. Note that while ReduceTransitions distinguishes between CAEs with different effects on the same fluent, the details of the effect are ignored by ReduceTransitions and Scope Task.

Algorithm 2 Task Scoping. Note that g is used as a 'dummy' goal fluent, similar to `UCPOP`, to simplify the main loop.

1: **procedure** SCOPE TASK($S[\mathcal{J}], \mathcal{A}, \mathcal{T}, s_0, G$)
2: $\quad\mathcal{J} \leftarrow \mathcal{J} \cup \{g\}$
3: $\quad\mathcal{T} \leftarrow \mathcal{T} \cup \{(G, \text{doGoal}, g \leftarrow \text{True})\}$
4: $\quad J_{rel} \leftarrow \{g\}$
5: $\quad T[J_{rel}] \leftarrow$ REDUCETRANSITIONS($\mathcal{T}$, J_{rel})
6: $\quad$**while** $(S[J_{rel}], \mathcal{A}, T[J_{rel}], s_0, G)$ is not a scoped RPP **do**
7: $\quad\quad J_{aff} \leftarrow J_{rel} \bigcup_{\overline{x} \in T[J_{rel}]} \text{vars}(\overline{x}.\text{sideeffects})$
8: $\quad\quad$**for** $\overline{x} \in T[J_{rel}]$ **do**
9: $\quad\quad\quad$**for** $\phi \in \text{CNF}(\overline{x}.\text{precondition})$ **do**
10: $\quad\quad\quad\quad$**if** $(s_0 \implies \phi) \vee (\text{vars}(\phi) \cap J_{aff} \neq \{\})$ **then**
11: $\quad\quad\quad\quad\quad J_{rel} \leftarrow J_{rel} \cup \{\text{vars}(\phi)\}$
12: $\quad\quad T[J_{rel}] \leftarrow$ REDUCETRANSITIONS($\mathcal{T}$, J_{rel})
13: $\quad$**return** $(S[J_{rel}], \mathcal{A}, T[J_{rel}], s_0, G)$

Theorem 3. The task scoping algorithm returns an SC-RPP.

Proof. The algorithm begins by setting $J_{rel} = \{g\}$ and thereafter never deletes a fluent from J_{rel}. Therefore, $g \in J_{rel}$.

The algorithm adds at least one fluent to J_{rel} each iteration, or terminates. J_{rel} is always a subset of $\mathcal{J}$.

Finally, scoping *cannot* terminate without returning a *scoped* RPP due to the explicit check in line 6. By the previous theorems, a scoped RPP with goal fluents in $J_{rel} \cup J_{CL}$ is sound and optimality-complete. □

3.5 Computational Complexity

For a given planning problem, suppose the transition dynamics $\mathcal{T}$ is composed of t CAE triples, and that there are n_j fluents. The worst-case complexity of ReduceTransitions $O(t)$. Given this, and by inspecting the loop in Algorithm 1, a bound for the worst-case computational complexity of task scoping is:

$$O(n_j \cdot (t + t \cdot O(\text{convert to CNF}) \cdot |\text{clauses in CNF}| \cdot (O(\text{check } s_0 \implies \phi))) + O(t))$$

Now, the worst-case complexity of converting a clause to CNF is known to be exponential. However, in practice, this conversion does not significantly impact the algorithm's runtime since most preconditions for CAE triples in planning domains contain few clauses. The overall complexity of task scoping is thus dominated by the $n_j \times t$ term.

Crucially, note that the time complexity of the task scoping algorithm does *not* depend on the concrete state-action space of the original planning problem. Rather, it depends only on the *number* of fluents. Thus, task scoping can substantially decrease the search space for a planner if the problem has irrelevant fluents that can take on many values. Consider our running example and suppose there are 10^{10} temperature values that the thermostat could be set to instead of 5. Task scoping's runtime is unaffected by this change, yet, by returning an SC-RPP that excludes the temperature factor, it would reduce the domain's state space by a multiplicative factor of 10^{10} and *also* enable the deletion of actions that affect only this irrelevant fluent. Conversely, task scoping might not be as useful if the temperature were a boolean, since it would only reduce the search space by a factor of 2.

Furthermore, though task scoping's worst-case complexity depends on the number (t) of CAE triples that specify the transition dynamics, in practice, it is not actually the *number* of CAE triples in the transition dynamics that affects the algorithm's performance, but rather their connectivity: whether the triples affect factors mentioned in other triples' preconditions. The more connected the relevant factors, the more iterations of ReduceTransitions must be run. For example, if instead of being scared of bells, the monkey could only be frightened by a `clap` action that can always be executed (that is, it is not connected to any other CAE triple, and directly affects the goal factor), then task scoping would terminate after just 1 iteration.

Thus, intuitively, task scoping is particularly advantageous over direct state-action space search for planning problems that possess many irrelevant fluents and actions, but have relatively few fluents in comparison to the size of the concrete state space, and whose transition dynamics are composed of CAE triples that do not have too many effects on factors mentioned in preconditions of *other* CAE triples (the transition dynamics are not too interconnected). We contend that many open-scope planning problems possess these characteristics and are thus amenable to task scoping.

Chapter 4

Implementation and Experimental Evaluation

4.1 Implementation Details

We implemented the task scoping algorithm (whose pseudocode is detailed in Algorithm 2) in Python and used the Z3 Theorem Prover's (De Moura and Bjørner, 2008) Python API to efficiently check implications (necessary for line 10 of the pseudocode). In order to run our algorithm on planning problems expressed in PDDL 2.1 (Fox and Long, 2003), we implemented a naive method to parse PDDL domain and problem files, ground all fluents and actions and then create CAE triples to pass into task scoping. Furthermore, we implemented a simple parser that reads the output of task scoping and deletes all irrelevant fluents, operators and objects from the original PDDL domain and problem files. A visual illustration of our pipeline is shown in Figure 4.1.

While this implementation is relatively efficient for planning problems with numeric fluents, it is rather inefficient for fully propositional problems due to two main reasons. Firstly, naive grounding of all fluents and actions is rather slow because propositional domains tend to have many objects, which result in many grounded fluents and actions. Secondly, the large number of grounded fluents and actions results in a significantly large worst-case time-complexity for task scoping (as discussed in Section 3.5), which makes our approach not much more efficient than direct search on the state-action space.

To enable task scoping to be efficient on such purely propositional planning problems, we implemented a pipeline combining task scoping with Fast Downward's (FD) (Helmert, 2006) translator. As part of this pipeline, we first run FD's translator on the PDDL domain and problem files to produce an SAS+ file. This file contains enum-valued fluents that are inferred from the propositional fluents of the original problem. Thus, this SAS+ planning task often contains fewer fluents, which makes it more conducive to task scoping. We parse the initial state, CAE triples and goal state for this SAS+ task and pass these as input to our implementation of task scoping. We then use the output of task scoping to prune fluents and actions from the original SAS+ planning task before planning on this reduced model. A visual illustration of this modified pipeline is shown in Fig.4.2.

We implemented the task scoping algorithm along with PDDL 2.1 and SAS+ (Helmert, 2006) domain parsers capable of extracting the information necessary for task scoping from planning problems written in either of these formats.

All experiments were conducted on an Intel Core i7-8550U CPU with 2 quad-core 4GHz processors, and 16GB of RAM.

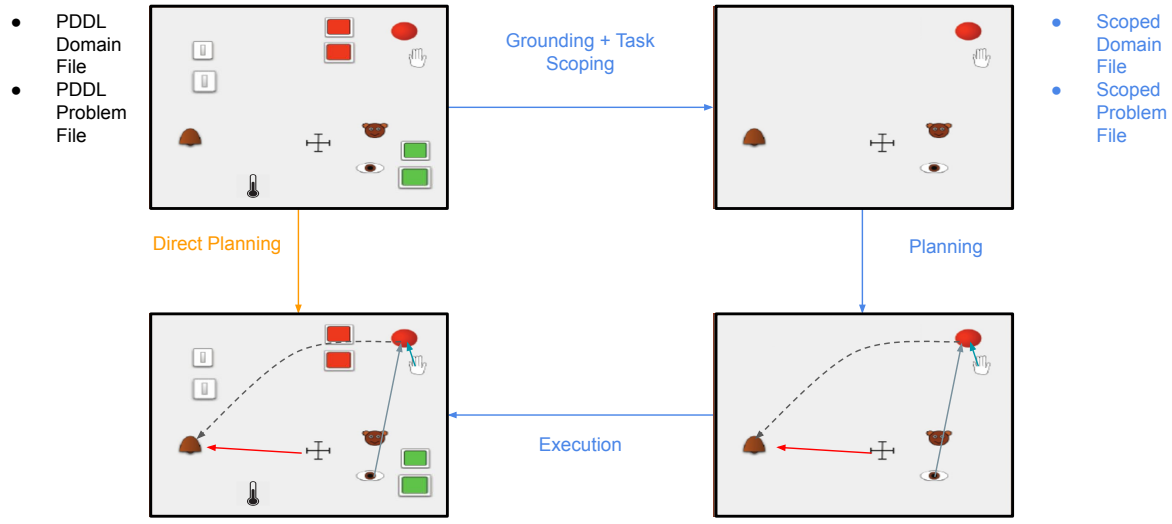


FIGURE 4.1: A visualization of our pipeline (shown in blue) vs. the traditional, direct planning method (shown in orange). We compare the end-to-end wall-clock times taken for these two different processes in each of our experiments

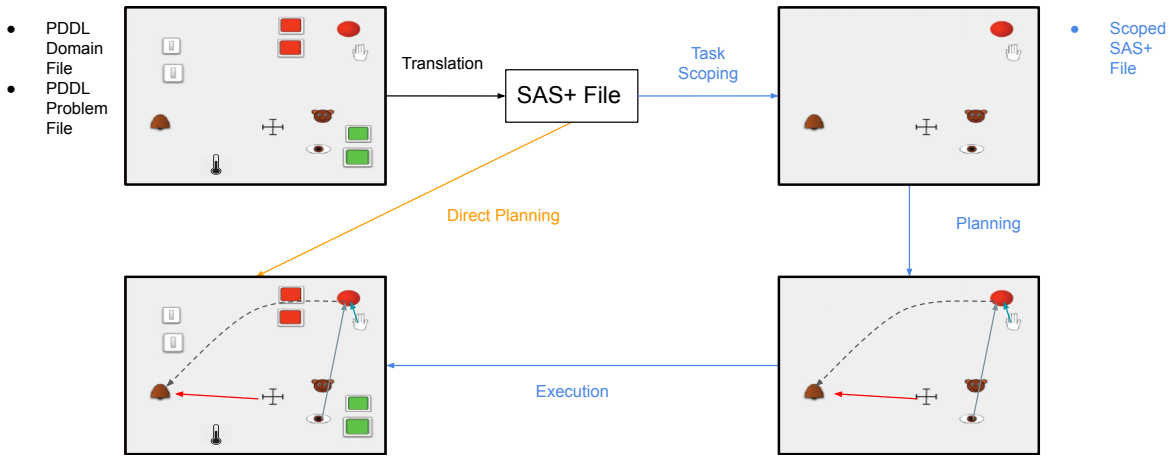


FIGURE 4.2: A visualization of our pipeline (shown in blue) vs. the traditional, direct planning method (shown in orange) levering FD's translator for purely propositional domains. We compare the end-to-end wall-clock times taken for these two different processes in each of our experiments

4.2 Experiments

4.2.1 IPC Domains with Fast Downward

Perhaps the best-known pruning algorithm for classical planning problems is the translator component of the Fast Downward planning system (FD) (Helmert, 2006). To empirically compare task scoping's utility against this existing approach, we selected 5 benchmark domains (Logistics, DriverLog, Satellite, Zenotravel and Gripper) from the optimal track of several previous iterations of the International Planning Competition (IPC) (Long and Fox, 2003; Vallati et al., 2015; Gerevini et al., 2009; Long et al., 2000); 4 of these domains are open-scope (Logistics, DriverLog, Satellite and Zenotravel). Since these domains do not generally contain enough irrelevance (especially conditional irrelevance) for pruning techniques to make a significant difference to search time (Hoffmann et al., 2006), we modified 3 problem files from the 4 open-scope domains to contain some states and actions that are either trivially or conditionally irrelevant. We translated each problem to SAS+ using FD's translator, ran task scoping to prune this SAS+ file, then ran the FD planner in the `seq-opt-lmcut` configuration on each of these problems. We inspected the original and scoped SAS+ files to compare the relative sizes of their state-action spaces and measured the end-to-end wall-clock time for FD to plan with and without task scoping. Results are in Table 4.1.

These results reveal that task scoping is able to prune some of the open-scope problems more finely than FD's translator alone, and that scoping can reduce FD's search time by more than the time taken by scoping itself. Task scoping reduces the size of the state-action space significantly more than FD's translator alone for the 4 domains that we added irrelevance to. By inspecting the SAS+ files, we observed that this difference in reduction was mostly because FD's translator was unable to prune any conditionally irrelevant states or actions. Additionally, for almost all problems within these 4 domains, running task scoping was able to significantly speed up FD's search time. This is especially evident in the Satellite domain, where the total time taken to find a plan was between $2\times$ and $6\times$ faster with task scoping than without. We also observed that task scoping was unable to prune any irrelevance in Gripper, which is unsurprising because it is a very simple domain that features neither trivial nor conditional irrelevance. However, the total planning time was approximately the same both with task scoping and without.

4.2.2 Numeric Domains with ENHSP-2020

Continuous Playroom Domain

To study how task scoping's performance and utility scale with the size of the state-action space within a simple open-scope numeric planning domain, we implemented a version of our running example in PDDL 2.1 (without the trivially irrelevant `thermostat` object or `(temperature)` factor). We created progressively larger problems by increasing the number of conditionally irrelevant light switches and buttons. For each of these problems, we ran ENHSP-2020 equipped with its default heuristic both with and without pre-processing using task scoping, and measured the end-to-end wall-clock time to produce a plan (that is, we compared the time to parse and plan for ENHSP with the time taken to parse, ground, scope, save the scoped PDDL domain and problem files, then parse and plan with ENHSP). The results, shown in Figure 4.3, clearly demonstrate that performing task scoping as a pre-processing step can significantly speed up planning by reducing the size of the state-action space. By simply removing the irrelevant switches, buttons, and actions, the state-action space can often be reduced by a large multiplicative factor, as shown in Table 4.1.

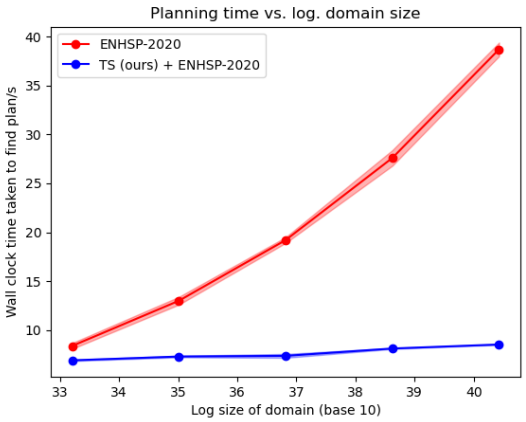


FIGURE 4.3: Planning time using ENHSP-2020 with and without task scoping, as the number of irrelevant fluents are increased in the Multi-Switch Continuous Playroom Domain. Shading indicates standard deviation across 5 independent runs.

Minecraft Domain

To examine task scoping's utility on a novel open-scope domain of interest, we created the *Build-a-Bed Domain* containing simplified dynamics and several tasks from Minecraft and pictured in Figure 1.1. The domain's main objective is to construct a blue bed—a relatively important task in the videogame—in a simplified setting. The domain features a number of interactive items—such as diamonds, sticks and various plants—strewn about the map, that the agent can destroy, place elsewhere or use to craft different items. Thus, the domain supports a large variety of potential tasks.

We wrote PDDL 2.1 files to express the Build-a-Bed domain and 3 specific tasks within it: (1) dye 3 wool blocks blue by plucking 3 blue flowers from a field and using these to craft blue dye, (2) mine wood using a diamond axe and use this to craft wooden planks, and (3) use the results of (1) and (2) to craft a blue bed and place it at a specific location. The agent possess 3 wool blocks and a diamond axe in the initial state, which does not vary between tasks.

Task scoping is successfully able to recognize and remove a large number of irrelevant fluents and associated actions depending on the task chosen within this domain, as shown in Table 4.1. The state space is reduced by over 10^{100} for each of these goals, and this reduction dramatically speeds up planning time for the Wool Dyeing and Bed Making tasks.

TABLE 4.1: Results for all of our experiments. All times are in seconds, and shown ± standard deviation across 5 independent runs.

problem	(state) x action size	scoped state x action size	scoping time	planning time (after scoping)	scoping + planning time	planning time (no scoping)
Logistics prob15	$(2.29 \times 10^{21}) \times 650$	$(\mathbf{1.14 \times 10^{9}}) \times \mathbf{250}$	1.55 ± 0.02	6.03 ± 0.02	$\mathbf{7.58 \pm 0.02}$	23.8 ± 0.5
Logistics prob20	$(2.29 \times 10^{21}) \times 650$	$(\mathbf{1.14 \times 10^{9}}) \times \mathbf{250}$	1.57 ± 0.07	10.8 ± 0.04	$\mathbf{12.4 \pm 0.09}$	53.5 ± 2
Logistics prob25	$(2.29 \times 10^{21}) \times 650$	$(\mathbf{1.93 \times 10^{10}}) \times \mathbf{290}$	1.70 ± 0.03	64.4 ± 1	$\mathbf{66.1 \pm 1}$	257 ± 7
DriverLog prob15	$(2.97 \times 10^{21}) \times 2592$	$(\mathbf{2.83 \times 10^{15}}) \times \mathbf{2112}$	6.71 ± 0.08	6.69 ± 0.04	13.4 ± 0.1	$\mathbf{10.6 \pm 0.09}$
DriverLog prob16	$(2.85 \times 10^{27}) \times 4890$	$(\mathbf{5.57 \times 10^{15}}) \times \mathbf{3540}$	14.7 ± 0.1	18.9 ± 0.3	$\mathbf{33.7 \pm 0.2}$	47.1 ± 0.7
DriverLog prob17	$(8.69 \times 10^{35}) \times 6170$	$(\mathbf{1.28 \times 10^{16}}) \times \mathbf{3770}$	17.2 ± 0.4	18.5 ± 0.3	$\mathbf{35.7 \pm 0.2}$	48.3 ± 0.2
Satellite prob05	$(2.10 \times 10^{12}) \times 339$	$(\mathbf{1.14 \times 10^{9}}) \times \mathbf{250}$	1.54 ± 0.02	1.04 ± 0.008	$\mathbf{2.58 \pm 0.02}$	4.19 ± 0.09
Satellite prob06	$(1.32 \times 10^{9}) \times 582$	$(\mathbf{1.09 \times 10^{7}}) \times \mathbf{362}$	1.35 ± 0.02	3.19 ± 0.02	$\mathbf{4.54 \pm 0.03}$	12.5 ± 0.06
Satellite prob07	$(3.76 \times 10^{13}) \times 983$	$(\mathbf{2.17 \times 10^{10}}) \times \mathbf{587}$	2.14 ± 0.02	103.0 ± 4	$\mathbf{105.0 \pm 4.0}$	689.0 ± 30.0
Zenotravel prob10	$(7.19 \times 10^{11}) \times 1155$	$(\mathbf{1.12 \times 10^{10}}) \times \mathbf{1095}$	4.48 ± 0.02	66.5 ± 0.6	$\mathbf{71.0 \pm 0.6}$	77.0 ± 2.0
Zenotravel prob12	$(1.55 \times 10^{16}) \times 3375$	$(\mathbf{7.47 \times 10^{11}}) \times \mathbf{3159}$	13.2 ± 0.09	91.0 ± 1.0	$\mathbf{104 \pm 2.0}$	107 ± 0.9
Zenotravel prob14	$(6.46 \times 10^{19}) \times 6700$	$(\mathbf{8.51 \times 10^{13}}) \times \mathbf{6200}$	13.0 ± 0.2	200.0 ± 3.0	$\mathbf{213.0 \pm 3.0}$	227.0 ± 4.0
Gripper prob04	$(1.43 \times 10^{7}) \times 82$	$(1.43 \times 10^{7}) \times 82$	0.449 ± 0.01	1.24 ± 0.05	1.69 ± 0.05	$\mathbf{1.45 \pm 0.06}$
Gripper prob05	$(1.80 \times 10^{8}) \times 98$	$(1.80 \times 10^{8}) \times 98$	0.589 ± 0.09	7.45 ± 0.2	$\mathbf{8.04 \pm 0.2}$	8.41 ± 1.0
Gripper prob06	$(2.15 \times 10^{9}) \times 582$	$(2.15 \times 10^{9}) \times 582$	0.500 ± 0.01	49.0 ± 3.0	$\mathbf{49.0 \pm 3.0}$	52.3 ± 4.0
Playroom1	$(16 \times 10^{32}) \times 18$	$(\mathbf{2 \times 10^{32}}) \times \mathbf{14}$	0.276 ± 0.01	6.64 ± 0.07	$\mathbf{6.91 \pm 0.07}$	8.38 ± 0.3
Playroom3	$(10.24 \times 10^{34}) \times 18$	$(\mathbf{2 \times 10^{32}}) \times \mathbf{14}$	0.491 ± 0.05	6.81 ± 0.04	$\mathbf{7.30 \pm 0.08}$	13.0 ± 0.4
Playroom5	$(65.546 \times 10^{35}) \times 18$	$(\mathbf{2 \times 10^{32}}) \times \mathbf{14}$	0.782 ± 0.02	6.58 ± 0.1	$\mathbf{7.37 \pm 0.2}$	19.2 ± 0.3
Playroom7	$(41.943 \times 10^{37}) \times 18$	$(\mathbf{2 \times 10^{32}}) \times \mathbf{14}$	1.26 ± 0.01	6.86 ± 0.05	$\mathbf{8.12 \pm 0.05}$	27.6 ± 0.8
Playroom9	$(26.844 \times 10^{39}) \times 18$	$(\mathbf{2 \times 10^{32}}) \times \mathbf{14}$	1.94 ± 0.02	6.59 ± 0.04	$\mathbf{8.53 \pm 0.04}$	38.7 ± 0.7
Minecraft Wool Dyeing	$(4.81 \times 10^{135}) \times 29$	$(\mathbf{3.53 \times 10^{25}}) \times \mathbf{11}$	3.70 ± 0.05	1.48 ± 0.03	$\mathbf{5.14 \pm 0.07}$	168 ± 5
Minecraft Plank Crafting	$(4.81 \times 10^{135}) \times 29$	$(\mathbf{3.53 \times 10^{25}}) \times \mathbf{13}$	4.74 ± 0.2	3.44 ± 0.1	$\mathbf{8.18 \pm 0.3}$	16.0 ± 0.4
Minecraft Bed Making	$(4.81 \times 10^{135}) \times 29$	$(\mathbf{1.02 \times 10^{28}}) \times \mathbf{16}$	4.21 ± 0.07	208 ± 6	$\mathbf{212 \pm 6}$	3450 ± 50

Chapter 5

Related Work

The fundamental idea of restricting search to relevant parts of a problem's state-action space is not new. In fact, this was a primary motivation for the development of AI planning approaches like UCPOP that backchain from the goal to focus search to actions that causally affect the goal. However, it was later discovered that determining relevance within propositional planning domains is often as computationally hard as planning itself (Nebel, Dimopoulos, and Koehler, 1997), and that doing so is often unhelpful to planners (Hoffmann et al., 2006). As a result, many planners focus on developing heuristics to guide forward search to relevant parts of the search space (McDermott, 1996; Nebel, Dimopoulos, and Koehler, 1997; Hoffmann et al., 2006) instead of explicitly attempting to discern relevance. Contrary to this approach, we build upon the intuition behind UCPOP to explicitly reason about the relevance of fluents and actions. Unlike previous approaches that were shown to provide little benefit, we focus on open-scope problems that contain large amounts of irrelevant variables, as might be encountered in learned or multi-task domains. Furthermore, we avoid explicitly backchaining through grounded states by performing backchaining more abstractly.

Another line of work has attempted to delete fluents from planning problems to form a hierarchy of *abstract planning problems* (Knoblock, 1992; Sacerdoti, 1974). The hierarchy is formed such that a solution to any abstract planning problem higher up in the hierarchy is a partial solution to less abstract problems that are lower down in the hierarchy. By contrast, in our approach, only one abstract planning problem is constructed, and *any* solution to this problem is a solution to the original, less-abstract problem.

Yet another line of work has attempted to delete fluents from planning problems in order to form a lossy-abstraction within which to derive a useful heuristic for the non-abstract problem. Culberson and Schaeffer (1998) introduced the notion of a *pattern* (which is currently more well-known as a *projection abstraction*) and showed that such patterns could be used to derive heuristics for complex problems. Helmert, Haslum, and Hoffmann (2008), Haslum et al. (2007) and others attempt to formulate automated methods of generating patterns to yield useful heuristics. In contrast to these approaches, our work automatically generates a pattern that provably preserve the optimal plan without the need to construct databases or derive heuristics.

A recent abstraction technique that is related to task scoping is CounterExample-Guided Abstraction Refinement (CEGAR) (Seipp and Helmert, 2018; Rovner, Sievers, and Helmert, 2019), which generates abstractions through iterative refinement. CEGAR finds an optimal solution to the abstraction and checks whether it is a solution to the original problem. If it is, it returns the solution. If not, it either refines the abstraction and tries again, or returns a planning heuristic based on the abstract solution. Task scoping differs in that it derives a sound and optimality-complete abstraction, rather than deriving heuristics based on lossy abstractions. CEGAR can

use richer families of abstractions like cartesian abstractions, while task scoping currently only uses projection abstractions.

A separate line of related work involves *pruning-during-grounding* techniques that attempt to prune irrelevant fluents and actions while performing grounding of a problem. Perhaps the most popular of these works is used by the translator of Helmert (2006)'s widely-used Fast Downward Planner. This translator performs abstract reachability analysis from the initial state to prune and merge various predicates. More recent works have built upon this work to perform more aggressive pruning and merging (Fišer, 2020; Fišer, Horčík, and Komenda, 2020). While the original translator does prune some irrelevant states and actions, we empirically observed (in Section 4.2.1) that it is unable to prune conditionally irrelevant states and actions. Additionally, this approach and its successors perform a reachability analysis from the initial state and specifically attempt to minimize the encoding size of the problem, whereas our approach also accounts for the particular goal conditions and serves a related, but different purpose (i.e, computing a projection that is sound and optimality-complete). Furthermore, these previous approaches operate strictly on propositional planning problems whereas task scoping can be applied to numeric problems as well (as demonstrated in Section 4.2.2).

Another technique that is similar in spirit to pruning-during-grounding is *lifted planning* (Corrêa et al., 2020; Ridder, 2014), where the expensive step of grounding of all state-variables and actions to particular objects is avoided. This is particularly useful in open-scope domains, where there might be *many* irrelevant objects. However, unlike task scoping these approaches perform this partial grounding during heuristic-guided forward search instead of as a preprocessing step. Overall, we view lifted planning techniques as potentially complementary to our own approach: combining the two approaches is an interesting direction for future work.

A recently-emergent line of work has attempted to leverage machine learning methods to predict an abstraction that ignores objects (Silver et al., 2021) or actions (Gnad et al., 2019) that are irrelevant to a particular goal. Such methods can discern both trivial and conditional irrelevance and drastically improve the search time for state-of-the-art planners like Fast Downward. However, they must be trained on a set of small instances of a planning problem with the same goal to learn to predict irrelevance. Additionally, they provide no guarantees that their predicted abstractions are solution-preserving, but rather incrementally add more objects or actions if planning fails. We view these methods as orthogonal to our approach and consider integrating them an interesting direction for future work.

Finally, we introduced some preliminary ideas behind task scoping in Kumar et al. (2020). However, this work was performed with a different problem setting (Factored Markov Decision Processes) in mind. Additionally, we provided no empirical evidence or theoretical guarantees to our proposed method then.

Chapter 6

Conclusion

We introduced task scoping, a novel, planner-agnostic abstraction algorithm that can recognize and delete provably task-irrelevant fluents and actions from planning problems. We proved that task scoping always preserves optimal plans and characterized the algorithm's computational complexity. Through experiments on a variety of domains, we showed that performing task scoping as a pre-computation can significantly reduce the state-action space of many problems, especially *open-scope* problems, thereby enabling a state-of-the-art planner to solve these problems more quickly.

While our current algorithm and implementation have been successful, there are some limitations to be overcome. One practical yet significant limitation is that our implementation of task scoping was written in Python, which is an interpreted language. We believe that implementing the algorithm in a compiled language like C++ or Cython would significantly speed-up our algorithm's runtime. A more significant limitation is that our algorithm is currently only able to compute projection abstractions; finer abstractions like domain abstractions (Rovner, Sievers, and Helmert, 2019) could enable more substantial pruning on many more problems of interest. Another limitation is that the relevant set often grows as the particular goal is made more long-horizon, since achieving a longer-horizon goal often requires affecting more fluents. This could perhaps be mitigated by discovering a series of sequential sub-goals for a particular goal and then running task scoping for each of these.

Despite these listed limitations, we believe that this work takes an important step towards enabling more generally-intelligent and capable AI. If an agent — such as a personal household robot — is expected to solve a wide variety of potentially long-horizon tasks via planning, then it must necessarily maintain an open-scope model that is rich enough to contain all the information that could possibly be relevant to any of these tasks. However, as we and many others have shown, planning directly within such models is extremely inefficient. In order to enable efficient planning on such open-scope domains then, the agent must be able to quickly discern irrelevance within its model to plan based only on the task-relevant parts of the model. Indeed, this capability to discern relevance is a vital aspect of human intelligence; as Andre and Russell (2002) put it: "The ability to make decisions based on only *relevant* features is a critical aspect of intelligence". We hope that our work inspires others to discover even more efficient and useful algorithms for relevance determination within open-scope models. Over the long term, we hope that such algorithms can be combined with approaches that learn open-scope models from data to develop an agent capable of autonomously learning and then planning to accomplish a wide-variety of challenging, real-world tasks.

Acknowledgements

This thesis - and most of my other research work at Brown - would not be possible without the support of many people whose names have not appeared within this document thus far. I want to acknowledge and express a heartfelt note of thanks to each of them here.

Firstly and foremostly, I would like to thank my primary co-author on this project: Michael Fishman. Michael has been instrumental to every aspect of this work, from ideation to implementation, and has taught me much about how to conduct rigorous research. Secondly, I want to thank Natasha Danas for lending a Formal Methods perspective to our work and for always supporting this project - and all of my other research endeavors - despite any challenges. Thirdly, I want to express deep gratitude to my research advisors: Professors George Konidaris, Stefanie Tellex and Michael Littman; not only have each of them provided invaluable guidance and direction for this project, but also they have provided me with wise counsel and incredible opportunities throughout my time at Brown. I also want to acknowledge and thank the many students within the BigAI initiative for many helpful discussions, and for generally fostering a welcoming, fun environment in which to do research. Finally, I want to thank my family and friends at Brown for tirelessly supporting my research endeavors.

Bibliography

Andre, David and Stuart J. Russell (2002). "State Abstraction for Programmable Reinforcement Learning Agents". In: *Eighteenth National Conference on Artificial Intelligence*. Edmonton, Alberta, Canada: American Association for Artificial Intelligence, pp. 119–125. ISBN: 0-262-51129-0. URL: http://dl.acm.org/citation.cfm?id=777092.777114.

Chentanez, Nuttapong, Andrew G Barto, and Satinder P Singh (2005). "Intrinsically motivated reinforcement learning". In: *Advances in neural information processing systems*, pp. 1281–1288.

Corrêa, Augusto B. et al. (2020). "Lifted Successor Generation Using Query Optimization Techniques". In: *Proceedings of the International Conference on Automated Planning and Scheduling* 30.1, pp. 80–89. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/6648.

Culberson, Joseph C. and Jonathan Schaeffer (1998). "Pattern Databases". In: *Computational Intelligence* 14.3, pp. 318–334. DOI: 10.1111/0824-7935.00065. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/0824-7935.00065. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/0824-7935.00065.

De Moura, Leonardo and Nikolaj Bjørner (2008). "Z3: An Efficient SMT Solver". In: *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. TACAS'08/ETAPS'08. Budapest, Hungary: Springer-Verlag, pp. 337–340. ISBN: 3-540-78799-2, 978-3-540-78799-0. URL: http://dl.acm.org/citation.cfm?id=1792734.1792766.

Fišer, Daniel (2020). "Lifted fact-alternating mutex groups and pruned grounding of classical planning problems". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 06, pp. 9835–9842.

Fišer, Daniel, Rostislav Horčík, and Antonín Komenda (2020). "Strengthening Potential Heuristics with Mutexes and Disambiguations". In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 30, pp. 124–133.

Fox, Maria and Derek Long (2003). "PDDL2. 1: An extension to PDDL for expressing temporal planning domains". In: *Journal of artificial intelligence research* 20, pp. 61–124.

Gerevini, Alfonso E et al. (2009). "Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners". In: *Artificial Intelligence* 173.5-6, pp. 619–668.

Gnad, Daniel et al. (2019). "Learning how to ground a plan–Partial grounding in classical planning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33, pp. 7602–7609.

Haslum, Patrik et al. (Jan. 2007). "Domain-Independent Construction of Pattern Database Heuristics for Cost-Optimal Planning." In: vol. 2, pp. 1007–1012.

Helmert, Malte (July 2006). "The Fast Downward Planning System". In: *J. Artif. Int. Res.* 26.1, pp. 191–246. ISSN: 1076-9757. URL: http://dl.acm.org/citation.cfm?id=1622559.1622565.

Helmert, Malte, Patrik Haslum, and Jörg Hoffmann (Jan. 2008). "Explicit-State Abstraction: A New Method for Generating Heuristic Functions." In: vol. 3, pp. 1547–1550.

Hoffmann, Jörg et al. (Jan. 2006). "Friends or Foes? An AI Planning Perspective on Abstraction and Search". In: *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS 2006), AAAI, 294-303 (2006)*.

Knoblock, Craig A. (1992). "An Analysis of ABSTRIPS". In: *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 126–135. ISBN: 155860250X.

Konidaris, George (2019). "On the necessity of abstraction". In: *Current Opinion in Behavioral Sciences* 29. SI: 29: Artificial Intelligence (2019), pp. 1 –7. ISSN: 2352-1546. DOI: https://doi.org/10.1016/j.cobeha.2018.11.005. URL: http://www.sciencedirect.com/science/article/pii/S2352154618302080.

Kumar, Nishanth et al. (2020). "Task Scoping: Building Goal-Specific Abstractions for Planning in Complex Domains". In: *arXiv preprint arXiv:2010.08869*.

Long, Derek and Maria Fox (2003). "The 3rd international planning competition: Results and analysis". In: *Journal of Artificial Intelligence Research* 20, pp. 1–59.

Long, Derek et al. (2000). "The AIPS-98 planning competition". In: *AI magazine* 21.2, pp. 13–13.

McDermott, Drew (1996). "A Heuristic Estimator for Means-Ends Analysis in Planning". In: *AIPS*.

Nebel, Bernhard, Yannis Dimopoulos, and Jana Koehler (May 1997). "Ignoring Irrelevant Facts and Operators in Plan Generation". In: vol. 1348. DOI: 10.1007/3-540-63912-8_97.

Penberthy, J. Scott and Daniel S. Weld (1992). "UCPOP: A Sound, Complete, Partial Order Planner for ADL". In: *KR*.

Refanidis, Ioannis, Nick Bassiliades, and Ioannis Vlahavas (2001). "AI Planning for Transportation Logistics". In: Citeseer.

Ridder, Bernardus (2014). "Lifted heuristics: towards more scalable planning systems". PhD thesis. King's College London (University of London).

Rovner, Alexander, Silvan Sievers, and Malte Helmert (2019). "Counterexample-Guided Abstraction Refinement for Pattern Selection in Optimal Classical Planning". In: *Proceedings of the International Conference on Automated Planning and Scheduling* 29.1, pp. 362–367. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/3499.

Sacerdoti, Earl D. (1974). "Planning in a hierarchy of abstraction spaces". In: *Artificial Intelligence* 5.2, pp. 115 –135. ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(74)90026-5. URL: http://www.sciencedirect.com/science/article/pii/0004370274900265.

Scala, Enrico et al. (2016). "Interval-Based Relaxation for General Numeric Planning". In: *Proceedings of the Twenty-Second European Conference on Artificial Intelligence*. ECAI'16. NLD: IOS Press, 655–663. ISBN: 9781614996712. DOI: 10.3233/978-1-61499-672-9-655. URL: https://doi.org/10.3233/978-1-61499-672-9-655.

Segler, Marwin HS, Mike Preuss, and Mark P Waller (2018). "Planning chemical syntheses with deep neural networks and symbolic AI". In: *Nature* 555.7698, pp. 604–610.

Seipp, Jendrik and Malte Helmert (May 2018). "Counterexample-Guided Cartesian Abstraction Refinement for Classical Planning". In: *J. Artif. Int. Res.* 62.1, 535–577. ISSN: 1076-9757. DOI: 10.1613/jair.1.11217. URL: https://doi.org/10.1613/jair.1.11217.

Silver, Tom et al. (2021). "Planning with Learned Object Importance in Large Problem Instances using Graph Neural Networks". In: *AAAI*.

Vallati, Mauro and Lukáš Chrpa (2019). "On the Robustness of Domain-Independent Planning Engines: The Impact of Poorly-Engineered Knowledge". In: *Proceedings of the 10th International Conference on Knowledge Capture*. K-CAP '19. New York, NY, USA: Association for Computing Machinery, 197–204. ISBN: 9781450370080. DOI: 10.1145/3360901.3364416. URL: https://doi.org/10.1145/3360901.3364416.

Vallati, Mauro et al. (2015). "The 2014 international planning competition: Progress and trends". In: *Ai Magazine* 36.3, pp. 90–98.