Task Scoping: Generating Task-Specific Simplifications of Open-Scope Planning Problems

Michael Fishman*, Nishanth Kumar*,[1] Cameron Allen,[2] Natasha Danas,[2] Michael Littman,[2] Stefanie Tellex,[2] George Konidaris[2]

[1]MIT CSAIL, [2]Brown University Department of Computer Science
michael@fishman.ai, njk@csail.mit.edu

Abstract

A general-purpose agent must learn an open-scope world model: one rich enough to tackle any of the wide range of tasks it may be asked to solve over its operational lifetime. This stands in contrast with typical planning approaches, where the scope of a model is limited to a specific family of tasks that share significant structure. Unfortunately, planning to solve any specific task within an open-scope model is computationally intractable—even for state-of-the-art methods—due to the many states and actions that are necessarily present in the model but irrelevant to that problem. We propose task scoping: a method that exploits knowledge of the initial state, goal conditions, and transition system to automatically and efficiently remove provably irrelevant variables and actions from grounded planning problems. Our approach leverages causal link analysis and backwards reachability over state variables (rather than states) along with operator merging (when effects on relevant variables are identical). Using task scoping as a pre-planning step can shrink the search space by orders of magnitude and dramatically decrease planning time. We empirically demonstrate that these improvements occur across a variety of open-scope domains, including Minecraft, where our approach reduces search time by a factor of 75 for a state-of-the-art numeric planner, even after including the time required for task scoping itself.

1 Introduction

Modern AI planning is extremely general-purpose—the promise of domain-independent planners is that a single program can be used to solve planning tasks arising from many specific applications. This promise has largely been realized: given an appropriately specified model of a planning problem, modern planners can quickly tackle anything from game playing (Korf 1985a,b) to transportation logistics (Refanidis et al. 2001) to chemical synthesis (Matloob and Soutchanski 2016). But while the achievements of domain-independent planners are impressive, one pervasive assumption significantly limits their generality: namely, that the domain model is always well-matched to the task. Each of the applications discussed above used a problem encoding that was carefully designed by human experts. While each domain supports multiple problems, these problems all share significant structure; a planner cannot solve logistics problems with a chemical synthesis model. The model has limited *scope*; it is only intended to be compatible with a specific, restricted set of possible tasks.

*These authors contributed equally.

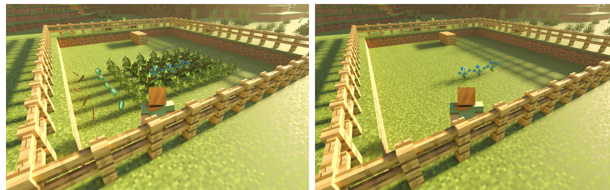

Figure 1: (Left) An open-scope Minecraft environment. All of the objects are occasionally important; however, for the specific task of crafting a bed, the planning agent can ignore most of them. (Right) Task scoping removes irrelevant objects and actions prior to planning, reducing planning time by an order of magnitude.

By contrast, general-purpose agents, particularly those that learn world models, cannot assume there will be a human expert on hand to provide them with a carefully specified model for any problem they might face. Instead, they must acquire and maintain an *open-scope* model—one rich enough to describe any planning task they may encounter during their deployment. Such an open-scope model will necessarily contain large amounts of information irrelevant to any individual task (Konidaris 2019). For instance, an agent that learns a model for planning in Minecraft (e.g. using the approach of James, Rosman, and Konidaris (2022)) must be capable of expressing a range of tasks with that model, such as obtaining resources, building shelter, crafting weapons, cooking food, and fighting off enemies—to name a few. However, when confronted with the specific, immediate task of crafting a bed, information about other tasks like cooking food is simply irrelevant.

Unfortunately, that generality comes at a cost: when learned models contain large amounts of irrelevant information, the search space grows exponentially and planning quickly becomes intractable. Recent work (Vallati and Chrpa 2019; Silver et al. 2021) has shown that many state-of-the-art planning engines suffer significant reductions in performance when irrelevant objects, state variables, or operators are included in domain descriptions. For example, the state-of-the-art numeric planner ENHSP (Scala et al. 2020) fails to find an optimal plan to construct a bed from

the objects in the Minecraft domain of Figure 1. Fast Downward (Helmert 2006) fails to even translate a non-numeric version of the same problem. With irrelevant objects and actions removed, both planners can solve the task within a few minutes.

To help general-purpose agents overcome these challenges, we introduce *task scoping*, a method for reasoning about which information can safely be removed from the agent's model. We identify three types of task-scoping simplifications that agents can use, after grounding but prior to planning, to remove irrelevant actions and variables. First, agents need only consider actions as relevant if they modify goal variables or preconditions of other relevant actions. Second, agents can identify actions that have identical effects on relevant variables, and merge them. Third, agents can also ignore variables that already match relevant preconditions and goal clauses, unless a relevant action can modify them.

We prove that task scoping preserves all optimal plans and empirically demonstrate that it leads to substantial reductions in search space size and planning time. Applying task scoping to the Minecraft problem of Figure 1 allows us to solve the previously intractable problem in under 3 minutes. We also observe significant improvements on a variety of other numeric and classical planning domains. Most importantly, the entire process is automatic, and compatible with off-the-shelf planners, enabling agents to derive their own task-specific simplifications for planning with open-scope models, all without requiring any additional domain knowledge from human experts.

2 Background

We adopt the planning formalism of Scala et al. (2016), but with multivariate enum variables replacing binary variables. We define a planning domain (or model) in terms of the following quantities:

- A finite set of variables $\mathcal{V} = \mathcal{V}_e \cup \mathcal{V}_n$, composed of enums $\mathcal{V}_e$ and numerics $\mathcal{V}_n$. The domain of a variable, $\mathcal{D}(v)$, is finite for enums and the set of reals, $\mathbb{R}$, for numerics.
- A factored state space $\mathcal{S}$, wherein each state s assigns a value $d_i \in \mathcal{D}(v_i)$ to every variable v_i in $\mathcal{V}$.
- A set of grounded operators (i.e. actions) $\mathcal{O}$ where each operator o consists of a cost $c(o) \geq 0$, as well as a precondition $pre(o)$ and an effect $\mathit{eff}(o)$, defined below.

A precondition is a (possibly nested) conjunction of enum and numeric conditions and their negations.[1] An enum condition is an equality relation $v_e = c_e$ between an enum variable v_e and a constant c_e in the domain $\mathcal{D}(v_e)$. When an enum variable is binary, we will write v_e as a shorthand for $v_e = \text{TRUE}$. A numeric condition is a binary relation $\{=, \geq, >\}$ between two numeric expressions, each of which can be either a real-valued constant, a numeric variable, or a binary function $\{+, -, \times, \div\}$ of numeric expressions.

An effect is a list of enum and numeric effects, each to a distinct variable v_i in $\mathcal{V}$. An enum effect is an assignment $v_e := c_e$, where the value c_e is a constant in the variable's domain $\mathcal{D}(v_e)$. A numeric effect is an update to a variable v_n, via an operator $\{:= ,+=,-=\}$ and a corresponding numeric expression e_i, as defined above.

An operator o is applicable in state s if s implies $pre(o)$. Executing o incurs cost $c(o)$ and causes the variable updates described in $\mathit{eff}(o)$, resulting in a new state s'. We reference the variables appearing in $pre(o)$ and $\mathit{eff}(o)$ using the notation $vars(pre(o))$ and $vars(\mathit{eff}(o))$. If an effect assignment to a variable v depends on the value of another variable u, then u is considered a precondition variable of the operator, even if it does not appear in the precondition.

We define a problem instance (or task) of a given domain by adding an initial state $s_0 \in \mathcal{S}$ and a goal condition G consisting of an assignment to some or all of the variables in $\mathcal{V}$. Together, the domain and problem instance induce a grounded planning problem, in which each lifted operator in the domain is instantiated with all possible variable arguments to form a set of grounded operators.[2] A plan is a sequence of successively-applicable grounded operators $[o_1, o_2, ..., o_n]$ from initial state s_0 to final state s_n, such that s_n implies G. An optimal plan is any plan that incurs the minimum cost. Given a model, a task is *solvable* if there exists at least one plan using that model.

This formalism is compatible with both numeric and FDR planning and supports a variety of problem encodings, including PDDL 2.1, level 2 (Fox and Long 2003) and SAS+ (Bäckström 1992).[3] Our experiments investigate these two planning paradigms separately and assume numeric domains only ever contain binary enums, which is consistent with Scala et al. (2016). However, in principle our approach is general enough to handle numerics and multivariate enums at the same time, since our algorithm never looks at the domain of any variable.

Open-Scope Models

A general-purpose planning agent may be asked solve many tasks during its operational lifetime. If the agent's model contains too few variables or operators, some tasks will not be solvable. To ensure that as many potential tasks as possible are solvable, the agent's model must be open-scope: it must contain more variables and/or operators than are relevant for any one task.

Given a model M and a task t, we say that an operator is *task-relevant* (or simply *relevant*) if it appears in at least one shortest cost-optimal plan to solve t. We say that M is an *open-scope* model with respect to t if it contains operators that are not relevant (henceforth 'irrelevant'). The wider the

[1] Our implementation is more general and supports any boolean-valued expression that can be expressed using Z3 (De Moura and Bjørner 2008), but in principle, there are no restrictions on the expressions except that we can identify any associated variables and check effects for equality.

[2] In this work, we restrict our focus to scoping grounded planning problems, and leave potential scoping-related improvements to the grounding process itself for future work.

[3] For simplicity, we consider the former without conditional effects and the latter without axioms, though neither restriction is required by our approach.

range of tasks an agent may be asked to solve, the higher the likelihood that most of its operators will be irrelevant for any one of those tasks. These irrelevant operators are the ones we would like the agent to ignore with task scoping.

Example Consider the following simplified version of the Minecraft domain in Figure 1. The agent can collect food, sticks, and stone, which it needs for eating and making an axe. The task is to make an axe from scratch. We use numeric fluents for brevity, but the example does not require them.[4]

- $\mathcal{V} = \{N_{\text{FOOD}}, N_{\text{STICKS}}, N_{\text{STONE}} \in \{0, 1, ..., N\};$ $\text{HUNGRY}, \text{HAS_AXE} \in \{\text{TRUE}, \text{FALSE}\}\}$
- $\mathcal{O} = \{$
  - `hunt` :
    - $pre : \neg(N_{\text{FOOD}} = N) \wedge \neg\text{HUNGRY}$
    - $\mathit{eff} : (N_{\text{FOOD}} \mathrel{+}= 1) \wedge \text{HUNGRY}$
  - `gather` :
    - $pre : \neg(N_{\text{FOOD}} = N) \wedge \text{HUNGRY}$
    - $\mathit{eff} : (N_{\text{FOOD}} \mathrel{+}= 1)$
  - `get_stick` :
    - $pre : \neg(N_{\text{STICKS}} = N)$
    - $\mathit{eff} : (N_{\text{STICKS}} \mathrel{+}= 1)$
  - `get_stone` :
    - $pre : \neg(N_{\text{STONE}} = N)$
    - $\mathit{eff} : (N_{\text{STONE}} \mathrel{+}= 1)$
  - `eat` :
    - $pre : \neg(N_{\text{FOOD}} = 0) \wedge \text{HUNGRY}$
    - $\mathit{eff} : (N_{\text{FOOD}} \mathrel{-}= 1 \wedge \neg\text{HUNGRY})$
  - `make_axe` :
    - $pre : \neg(N_{\text{STICKS}} = 0 \vee N_{\text{STONE}} = 0 \vee \text{HAS_AXE})$
    - $\mathit{eff} : (N_{\text{STICKS}} \mathrel{-}= 1 \wedge N_{\text{STONE}} \mathrel{-}= 1 \wedge \text{HAS_AXE})$
  - `wait` :
    - $pre : \neg\text{HUNGRY}$
    - $\mathit{eff} : \text{HUNGRY}\}$
- $s_0 = (0,0,0,\text{FALSE},\text{FALSE})$
- $G = (\neg\text{HUNGRY} \wedge \text{HAS_AXE})$

All operators have unit cost. For this task, there are two optimal plans: [`get_stick`, `get_stone`, `make_axe`], and [`get_stone`, `get_stick`, `make_axe`]. The operators `hunt`, `gather`, `eat`, and `wait` are irrelevant and can be removed, since none appear in any optimal plan.

3 Task Scoping

The purpose of task scoping is to identify and remove task-irrelevant variables and operators from the agent's model. This process produces a simplification of the original planning problem aimed at making planning more tractable. However, not all such simplifications preserve optimal plans.

Definition 1. *Given a planning problem P, a* task-scoping simplification *P' of P is one that contains a subset of the variables and operators in P such that all shortest cost-optimal plans in P are still optimal plans of P'.*

In this section, we describe three types of task-scoping simplifications that remove increasing amounts of irrelevant information. We derive the simplifications using variations of Algorithm 1, and prove that each preserves optimal plans.

[4] Appendix C contains PDDL for this example with $N = 1$.

Algorithm 1: TASK SCOPING

Input: $\langle \mathcal{V}, \mathcal{O}, s_0, G \rangle$
Output: $\langle \mathcal{V}' \subseteq \mathcal{V}, \mathcal{K}' \subseteq \mathcal{V}, \mathcal{O}' \subseteq \mathcal{O} \rangle$

1: $V_0 \leftarrow \{\text{DUMMY_GOAL_VAR}\}$ ▷ relevant vars
2: $O_0 \leftarrow \{\texttt{dummy_goal_operator}(G)\}$ ▷ relevant ops
3: **repeat**
4: $\overline{O}_i \leftarrow O_{i-1} \parallel \text{MERGESAMEEFFECTS}(O_{i-1}, V_{i-1})$
5: $E_i \leftarrow \varnothing \parallel \{\text{variables} \in \mathit{eff}(O_{i-1})\}$
6: $L_i \leftarrow \varnothing \parallel s_0[\mathcal{V} \setminus E_i]$
7: $C_i \leftarrow \{\text{clauses in } pre(\overline{O}_i) \text{ not implied by } L_i\}$
8: $K_i \leftarrow \{\text{vars in clauses of } pre(\overline{O}_i)\} \setminus vars(C_i)$
9: ▷ causally linked variables
10: $V_i \leftarrow V_{i-1} \cup \{v \in vars(c) \; \forall \, c \in C_i\}$
11: $O_i \leftarrow \{o \in \mathcal{O} : \mathit{eff}(o) \cap V_i \neq \varnothing\}$
12: **until** $V_i = V_{i-1}$
13: $\mathcal{V}' \leftarrow V_n \setminus \{\text{DUMMY_GOAL_VAR}\}$
14: $\mathcal{K}' \leftarrow K_n \setminus \{\text{DUMMY_GOAL_VAR}\}$
15: $\mathcal{O}' \leftarrow O_n \setminus \{\texttt{dummy_goal_operator}(G)\}$
16: **return** $\langle \mathcal{V}', \mathcal{K}', \mathcal{O}' \rangle$

Backwards Reachability of Variables

The first and simplest task-scoping simplification encodes the notion that agents need not consider actions to be relevant unless they modify goal variables or preconditions of other relevant actions. This version of Algorithm 1 (which we call Algorithm 1-a) omits any of the colored text appearing after the '‖' symbols. It starts by considering only goal variables to be relevant, and performs backwards reachability analysis over variables, considering operators relevant if their effects contain any relevant variables, and then considering the precondition variables of any such operators to be relevant. The process repeats until no new variables are deemed relevant. The Fast Downward Planning System performs an equivalent simplification during its knowledge compilation process (Helmert 2006).

In the example of Section 2, this process would work proceed as follows:

HAS_AXE → `make_axe` → N_{STICKS} → `get_stick`
→ N_{STONE} → `get_stone`

HUNGRY → `eat` → N_{FOOD} → `hunt`
→ `gather`
→ `wait`

Eventually all variables and operators would be marked as relevant. Had ¬HUNGRY not appeared in the goal, the algorithm would not consider the bottom chains relevant, and those items would be removed. In Section 3 we will upgrade the algorithm to recognize that ¬HUNGRY is satisfied by the initial state and not modified by the top operators; therefore, the bottom chains can still be removed.

Merging Same-Effect Operators

The second task-scoping simplification reflects the idea that actions with identical effects on relevant variables are interchangeable and can therefore be merged. Algorithm 1-b extends the previous version by adding the MERGESAMEEF-

Algorithm 2: MERGESAMEEFFECTS

Input: O_i, V_i
Output: $\overline{O}_i$

1: $O_{\text{equiv}} \leftarrow$ Partition O_i based on effects on V_i and cost.
2: $\overline{O}_i \leftarrow$ merge operators in equivalence classes: {
  pre : take disjunction of preconditions & simplify
  eff : copy effects on V_i (identical)
  c : copy cost of component operators (identical)}
3: **return** $\overline{O}_i$

FECTS function on line 4, which is detailed in Algorithm 2.

The merging procedure partitions the relevant operators O_i into equivalence classes that have identical costs and effects on relevant variables V_i. Each resulting merged operator removes any effects on non-relevant variables, and its precondition is the disjunction of the original operators (simplified to remove any unnecessary clauses using Z3). As a result, line 7 of Algorithm 1 now produces potentially fewer precondition clauses C_i from which to add relevant variables in line 8. Note that the merged operators never appear in $\mathcal{O}'$, only the non-merged originals.

For example, suppose N_{FOOD} is the only relevant variable (perhaps the task is now to gather food). The operators `hunt` and `gather` both modify N_{FOOD}, so both are marked as relevant. Algorithm 1-b then calls MERGESAME-EFFECTS and determines that both operators have the same effect on N_{FOOD}, the only relevant variable, and can therefore be merged. After simplifying preconditions, the merge results in the following operator (name added for clarity):

`get_food`:
  pre : $\neg(N_{\text{FOOD}} = N)$
  eff : $(N_{\text{FOOD}} \mathrel{+}= 1)$

In this example, HUNGRY was not relevant to begin with and does not appear in the merged operator's precondition, so it would remain irrelevant. As a result, the `eat` and `wait` operators, which modify HUNGRY, never become relevant either, and the algorithm will return $\mathcal{O}' = \{\texttt{hunt}, \texttt{gather}\}$.

Causally Linked Irrelevance

The third task-scoping simplification we introduce captures the idea that agents can ignore variables that already match relevant preconditions and goal clauses, unless a relevant action modifies them. This corresponds to the concept of causal links (McAllester and Rosenblitt 1991). A clause is *causally linked* when (1) it is implied by some state (here we only consider causal links from s_0), (2) it appears in the precondition of a subsequent operator, and (3) it is not modified by any operators in between. In the Minecraft example of Section 2, this corresponded to the variable HUNGRY. Since the initial state and goal both contained the clause $\neg$HUNGRY and no relevant operator modified it, HUNGRY was *causally linked* and could safely be removed.

The full Algorithm 1 builds on the previous version by additionally identifying clauses L_i (in lines 5-6) that are causally linked with the initial state s_0 and contain no overlap with any variables mentioned in the effects of relevant operators.[5] Note that Algorithm 1 is guaranteed to terminate, since $\mathcal{V}$ is finite and $V_{i-1} \subseteq V_i \subseteq \mathcal{V}$ for every iteration.

Main Theorem

We show in this section that Algorithm 1 produces a simplification of the original planning problem that contains all optimal plans. The proof works by removing unnecessary operators from each plan, and makes use of a Merge Substitution lemma, which ensures that the resulting action sequences are still valid plans.

For any quantity X_i in the algorithm, we will use the subscript X_n to indicate the value X_i has in the final iteration of the algorithm. For example, K_n is the final set of causally linked variables. Additionally, for a set of state variables $Z \subseteq \mathcal{V}$, we use the notation $s[Z]$ to denote the partial state of s with respect to only the variables in Z.

Lemma 1 (Merge Substitution)**.** *Let o be any operator in O_n, s any state that implies $pre(o)$, and $s' \neq s$ another state. If the partial state of s' with respect to relevant and causally linked variables is the same as in state s (i.e. $s'[V_n \cup K_n] = s[V_n \cup K_n]$), then there exists another operator o', also in O_n (and possibly equal to o), such that: s' implies $pre(o')$; o and o' have the same effect on V_n; neither o nor o' has any effect on K_n; and o and o' have the same cost.*

Proof. Run MERGESAMEEFFECTS(O_n,V_n) to compute $\overline{O}_n$, and find the (potentially merged) operator $\overline{o}$ corresponding to o. Such an operator exists, because MERGESAMEEFFECTS partitions O_n.

First we will show that s' implies $pre(\overline{o})$. By construction, $pre(\overline{o})$ contains only variables in $(V_n \cup K_n)$, and the clauses containing variables in K_n are causally linked. This means that if s implies $pre(\overline{o})$, and $s'[V_n \cup K_n] = s[V_n \cup K_n]$, then s' also implies $pre(\overline{o})$. Since $pre(\overline{o})$ is just the disjunction of the preconditions of its component operators, it must be the case that s' implies the precondition of at least one such component operator $o' \in O_n$.

Since o' and o correspond to the same abstract operator $\overline{o}$, they must have the same effect on V_n and the same cost. Since both operators are in O_n, neither can affect K_n. □

Theorem 1. *Every shortest cost-optimal plan for a given task uses a subset of the operators returned by Algorithm 1.*

Proof. We will show that for any plan π containing operators not in O_n, there exists another plan π' that is shorter, has lesser or equal cost, and only uses operators in O_n.

We will go through the operators of π, from the beginning to the end, and for each operator o, add a corresponding operator to π' as follows:

1. If o affects at least one variable in V_n and can be taken from the current state of the modified plan, keep it; o is in O_n, since it modifies a variable in V_n.

[5]Additional optimizations are possible, such as by considering subsequent states or ignoring variables in C_i when their values do not affect the truth value of the causally linked clauses.

2. If o affects at least one variable in V_n and cannot be taken from the current state of the modified plan, replace it with an operator o' that has the same effects on V_n and can be taken from the current state. Such an operator is guaranteed to exist by Lemma 1.
3. If o does not affect V_n (and therefore is not in O_n), add a no-op operator to π' with the same cost as o.

Note that in case 3, the no-op operators need not exist in the domain. We could simply ignore o entirely in this case, but this would make the correspondences between the operators and states of π and π' slightly less clear, since the plans would have different lengths. We will delete the no-ops from π' after comparing the plans.

Now we provide an inductive proof over the steps in the plan to show that the following inductive assumption holds.

Inductive assumption: At each step, π and π' have the same partial state on V_n, and π''s partial state on K_n (the set of causally linked variables) is equal to the initial partial state on K_n. That is, $s_i[V_n] = s'_i[V_n]$ and $s'_i[K_n] = s_0[K_n]$.

Inductive base: Empty plans share initial state.

Inductive step: Each of the three cases outlined above preserves the inductive assumption.

- Case 1: the original operator o is applicable and does not change the plan. The resulting partial state $s'_{i+1}[V_n] = s_{i+1}[V_n]$. Since o is in O_n, it does not modify K_n.
- Case 2: this operator replacement is possible due to Lemma 1 and the inductive assumption. By Lemma 1, o and o' have the same effect on V_n, no effect on K_n, and equal cost. Furthermore, o' is in O_n.
- Case 3: o does not modify V_n, so $s'_{i+1}[V_n] = s_{i+1}[V_n]$. The no-op does not affect K_n, and has equal cost to o.

The above inductive argument shows that π' is also a valid plan, and has the same cost and length as π. We now delete the no-ops from π', and afterwards, π' is shorter than π, has lesser or equal cost, and uses only operators from O_n.

For any plan π with an operator not in O_n, there exists a shorter plan of equal or lesser cost, containing only operators in O_n. Therefore, all shortest cost-optimal plans use only operators from O_n. □

Discussion

It should be clear that Algorithm 1 does not involve searching through specific states. Rather, it reasons over the variables and operators of the planning problem. Space precludes a thorough complexity analysis, but Algorithm 1's worst-case complexity is dominated by $|\mathcal{V}| \times |\mathcal{O}|$. Since $|\mathcal{V}| \times |\mathcal{O}|$ is generally much smaller than the problem's full state–action space (which may even be infinite), deriving a task-scoping simplification is thus often significantly more efficient than planning over the original problem.[6]

The simplifications in the previous sections are not intended to be an exhaustive list of task-scoping simplifications. More aggressive simplifications are clearly possible, particularly when considering causal links.

[6]This complexity expression neglects the cost of simplifying preconditions in Algorithm 1, which is reasonable in practice because most preconditions are relatively simple.

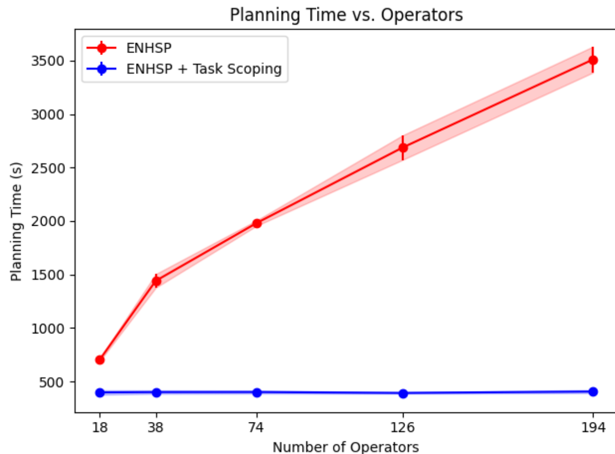

Figure 2: Results for the Multi-Switch Continuous Playroom domain. Total planning time with task scoping is essentially constant compared to the baseline, even as the size of the domain grows exponentially. (Total planning time includes scoping time; error bars show standard deviation across 10 independent trials.)

4 Experimental Evaluation

Algorithm 1 is agnostic to which particular representation is used to express the planning problem. To demonstrate its performance and utility empirically, we customized it to work with both numeric PDDL 2.1 level 2 (Fox and Long 2003), and SAS+ (Bäckström 1992; Helmert 2006). The output of the algorithm consists of relevant variables $\mathcal{V}'$, causally linked variables $\mathcal{K}'$, and relevant operators $\mathcal{O}'$. For SAS+, we simply remove operators outside of $\mathcal{O}'$ and causally linked goal conditions. For PDDL, we remove lifted operators from the domain file whenever they correspond to no grounded operators in $\mathcal{O}'$, and we remove objects from the problem file whenever they correspond to no variables in $\mathcal{V}' \cup \mathcal{K}'$.[7] Since this is a more conservative simplification, it is still a valid task-scoping simplification.

All experiments were conducted on a cluster of 2.90GHz Intel Xeon Platinum 8268 CPUs, using 2 virtual cores and 16GB of RAM per trial, and measurements are averaged across 10 independent trials.

Numeric Domains with ENHSP

We first investigate our approach's performance and utility in numeric domains. In all the following experiments, we run ENHSP (Scala et al. 2020) in optimal mode (`WAStar+hrmax`) with and without task scoping as a preprocessing step. We measure wall-clock time and nodes generated during search to produce a plan.

Multi-Switch Continuous Playroom. In order to study how our approach scales with the size of the input problem in a simplified setting, we implemented the continuous playroom domain from Chentanez, Barto, and Singh (2005)

[7]We keep objects corresponding to variables in $\mathcal{K}'$ because these objects may be used to ground operators in $\mathcal{O}'$.

Problem	Operators		State Variables		Evaluations		Scoping	Planning	Total Time (s)	
	Unscoped	Scoped	Unscoped	Scoped	Unscoped	Scoped		Scoped	Unscoped	Scoped
Playroom 1	18	**14**	22	**12**	13.0M	**11.3M**	0.3 ± 0.1	397 ± 20	703 ± 13	$\mathbf{397 \pm 20}$
Playroom 3	38	**14**	48	**12**	14.9M	**11.3M**	0.5 ± 0.0	400 ± 15	1440 ± 63	$\mathbf{401 \pm 15}$
Playroom 5	74	**14**	82	**12**	15.2M	**11.3M**	0.9 ± 0.0	400 ± 13	1981 ± 25	$\mathbf{401 \pm 13}$
Playroom 7	126	**14**	124	**12**	15.2M	**11.3M**	1.5 ± 0.1	391 ± 6	2686 ± 114	$\mathbf{393 \pm 6}$
Playroom 9	194	**14**	174	**12**	15.2M	**11.3M**	2.3 ± 0.1	403 ± 14	3510 ± 121	$\mathbf{406 \pm 14}$
Composite (depot)	1809	**1062**	592	**154**	(> 19.0M)	**488K**	26.6 ± 0.3	84.9 ± 0.9	(> 5.8K)	$\mathbf{111.5 \pm 1.1}$
Composite (driverlog)	1809	**336**	592	**263**	(> 18.6M)	**66K**	22.6 ± 0.1	4.7 ± 0.3	(> 4.3K)	$\mathbf{27.3 \pm 0.4}$
Composite (satellite)	1809	**94**	592	**157**	(> 22.1M)	**92K**	23.4 ± 0.2	3.8 ± 0.4	(> 5.3K)	$\mathbf{27.2 \pm 0.4}$
Composite (zenotravel)	1809	**37**	592	**36**	1.7M	**598**	11.7 ± 0.1	1.1 ± 0.1	363 ± 2	$\mathbf{12.7 \pm 0.1}$
Minecraft (planks)	106	**22**	268	**58**	353	**338**	4.8 ± 0.1	1.3 ± 0.2	$\mathbf{1.7 \pm 0.2}$	6.1 ± 0.3
Minecraft (wool)	106	**18**	268	**58**	1.6M	**6612**	4.1 ± 0.1	1.7 ± 0.2	432.5 ± 21.5	$\mathbf{5.8 \pm 0.2}$
Minecraft (bed)	106	**25**	268	**64**	(> 14.2M)	**1.1M**	5.0 ± 0.1	173.1 ± 8.0	(> 4.3K)	$\mathbf{178.1 \pm 8.0}$

Table 1: **Numeric planning results**. Task scoping removes irrelevant operators and state variables, which leads to significant reductions in evaluations and planning time. Boldface denotes better performance. Standard deviation is across 10 trials. $(> N)$ denotes out-of-memory.

in PDDL 2.1. In this domain, the agent can move in cardinal directions within a grid. Its goal is to pick up a ball and throw it at a bell, but this requires first pressing a number of green buttons, which in turn requires turning on a number of lights. Both the lights and buttons are strewn throughout the grid. In our experiments, we create progressively larger problems with more buttons and lights, yet ensure that all green buttons are turned on in the initial state. This renders the lights causally-linked with the initial state, so all corresponding actions that affect them are irrelevant.

The results, shown in Figure 2, demonstrate that task scoping is able to keep planning time relatively constant compared to the baseline, even as the size of the problem grows exponentially. Table 1 provides more detail, and shows that even after accounting for the time required to perform task scoping, the scoped planner still finds plans more quickly than the unscoped baseline.

Composite IPC Domains Task scoping is intended to make *open-scope* planning tractable, but most existing benchmarks are not open-scope: they have been carefully hand-designed to exclude any task-irrelevant information. While a full investigation of how to *learn* such an open-scope model is beyond the scope of this paper, it is straightforward to construct an open-scope domain from existing benchmarks: simply combine multiple domains together and attempt to solve a problem from any one of them.

We construct such a model by combining the domain and problem files of the `Depots`, `DriverLog`, `Satellite`, and `ZenoTravel` domains from IPC 2002 (Long and Fox 2003) into a single composite domain file and 4 different composite problem files. The problem files differ only in their goal condition: each file contains a goal related to a specific sub-domain, and the variables and actions related to the other 3 domains are task-irrelevant.

The results in Table 1 indicate that task scoping can dramatically reduce the number of operators, which makes optimal open-scope planning tractable in all four tasks, whereas ENHSP typically runs out of memory without scoping.

Minecraft To examine the utility of task scoping on a novel open-scope domain of interest, we created a planning domain containing simplified dynamics and several tasks from Minecraft and pictured in Figure 1. The domain features interactive items at various locations in the map, which the agent can destroy, place elsewhere, or use to craft different items. Thus, the domain is truly open-scope: it supports a large variety of potential tasks such that most objects and actions are irrelevant to most tasks. Within this domain, we wrote PDDL 2.1 files to express 3 specific tasks: (1) craft wooden planks, (2) dye 3 wool blocks blue, and (3) craft and place a blue bed at a specific location (which requires completing both prior tasks as subgoals). The results show that task scoping is able to recognize and remove a large number of irrelevant operators depending on the task chosen within this domain, as shown in Table 1. This dramatically speeds up planning time for the wool-dyeing and bed-making tasks.[8]

Classical Planning with Fast Downward

Having investigated our approach's utility and performance in a variety of numeric domains, we now turn to propositional domains. We are interested in examining whether our approach is able to discover task-irrelevant information beyond what the translator component of the well-known Fast-Downward planning system (Helmert 2006), and whether removing such irrelevance can substantially improve planning time. To this end, we selected 4 benchmark domains (Logistics, DriverLog, Satellite, Zenotravel) 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). Since these domains do not contain any task-irrelevance on their own (Hoffmann, Sabharwal, and Domshlak 2006), we modified 3 problem files from each domain with initial states and

[8] We also created a propositional version of this domain. Fast Downward was not even able to complete translation on it; however, after removing irrelevant objects for each problem by hand, planning took just a few seconds.

Problem	Operators		Expansions		Evaluations		Translate	Scoping	Planning Time (s)		Total Time (s)	
	Unscoped	Scoped	Unscoped	Scoped	Unscoped	Scoped			Unscoped	Scoped	Unscoped	Scoped
Driverlog 15	2592	**2112**	1392	**1379**	23K	**21K**	0.5 ± 0.0	3.6 ± 0.2	4.4 ± 0.1	$\mathbf{3.1 \pm 0.1}$	$\mathbf{4.9 \pm 0.2}$	7.2 ± 0.2
Driverlog 16	4890	**3540**	3618	**3087**	87K	**60K**	0.7 ± 0.0	8.3 ± 0.3	19.8 ± 0.8	$\mathbf{8.1 \pm 0.2}$	20.5 ± 0.8	$\mathbf{17.1 \pm 0.5}$
Driverlog 17	6170	**3770**	1058	**985**	29K	**21K**	0.8 ± 0.0	9.6 ± 0.3	22.0 ± 0.9	$\mathbf{7.9 \pm 0.2}$	22.8 ± 1.0	$\mathbf{18.3 \pm 0.5}$
Logistics 15	650	**250**	6395	6395	153K	**118K**	0.3 ± 0.1	0.7 ± 0.1	11.6 ± 0.2	$\mathbf{3.3 \pm 0.1}$	11.9 ± 0.2	$\mathbf{4.2 \pm 0.1}$
Logistics 20	650	**250**	15K	**14K**	381K	**260K**	0.3 ± 0.0	0.7 ± 0.0	26.9 ± 0.3	$\mathbf{5.9 \pm 0.1}$	27.2 ± 0.3	$\mathbf{6.9 \pm 0.2}$
Logistics 25	650	**290**	68K	**67K**	1.7M	**1.3M**	0.3 ± 0.0	0.8 ± 0.0	127.7 ± 1.4	$\mathbf{34.8 \pm 0.4}$	128.0 ± 1.4	$\mathbf{35.8 \pm 0.4}$
Satellite 05	609	**339**	1034	1034	63K	**35K**	0.3 ± 0.1	0.6 ± 0.1	2.0 ± 0.0	$\mathbf{0.6 \pm 0.0}$	2.3 ± 0.1	$\mathbf{1.5 \pm 0.0}$
Satellite 06	582	**362**	5766	**4886**	312K	**166K**	0.3 ± 0.1	0.5 ± 0.1	6.3 ± 0.1	$\mathbf{1.7 \pm 0.0}$	6.6 ± 0.1	$\mathbf{2.5 \pm 0.0}$
Satellite 07	983	**587**	125K	**96K**	10.5M	**4.9M**	0.4 ± 0.1	0.9 ± 0.1	333.1 ± 1.8	$\mathbf{50.9 \pm 0.3}$	333.4 ± 1.8	$\mathbf{52.2 \pm 0.3}$
Zenotravel 10	1155	**1095**	24K	24K	676K	**655K**	0.3 ± 0.0	2.4 ± 0.1	37.7 ± 0.7	$\mathbf{33.2 \pm 0.4}$	38.1 ± 0.7	$\mathbf{35.9 \pm 0.5}$
Zenotravel 12	3375	**3159**	4766	**4735**	222K	**211K**	0.6 ± 0.0	7.4 ± 0.1	45.5 ± 0.1	$\mathbf{39.2 \pm 0.3}$	$\mathbf{46.0 \pm 0.2}$	47.2 ± 0.3
Zenotravel 14	6700	**6200**	6539	6539	599K	**588K**	1.0 ± 0.0	14.2 ± 0.2	232.4 ± 18.5	$\mathbf{193.2 \pm 8.7}$	233.4 ± 18.5	$\mathbf{208.3 \pm 8.6}$

Table 2: **Classical planning results**. Task scoping removes irrelevant operators from every domain, which leads to significant reductions in node expansions, evaluations, and planning time. Boldface denotes better performance. Standard deviation is across 10 trials.

goals set to introduce irrelevance while keeping the domain files fixed. We grounded each problem to SAS+ using FD's translator, ran task scoping on the resulting SAS+ file, then ran the FD planner with the LM-cut heuristic (Helmert and Domshlak 2009) on this problem. We report number of operators, planning time, and nodes expanded and evaluated during search both with and without task scoping.

The results (see Table 2) reveal that task scoping can abstract some of these problems beyond what FD's translator can accomplish alone and lead to a net speedup. Algorithm 1 reduces the number of operators significantly for all 4 domains. This difference was mostly because FD's translator was unable to remove any causally-linked irrelevant variables or operators, though it was able to remove the simpler types of irrelevance discussed in Section 3.

5 Related Work

The Fast Downward Planning System (Helmert 2006) performs Algorithm 1-a from Section 3 as part of its knowledge compilation process. This is backwards reachability analysis on what the authors call the *achievability* causal graph. The MERGESAMEEFFECTS extension can be interpreted as computing abstract operators with fewer preconditions, making the causal graph sparser. The causal links extension (lines 5–7 of Algorithm 1) also makes the causal graph sparser by ignoring satisfied clauses of preconditions. The sparser causal graph means that the backwards reachability analysis terminates sooner, with fewer variables marked as potentially relevant.

Another popular method for reducing the size of the search space is the discovery of forward and backward invariants (a.k.a mutex constraints) (Bonet and Geffner 2001; Edelkamp and Helmert 1999; Chen, Xing, and Zhang 2007; Alcázar and Torralba 2015). Removing such invariants removes unreachable states or dead-end states, and their associated operators, from the planning problem and has been shown to dramatically improve search (Helmert 2006). However, removing invariants essentially amounts to removing states and operators that cannot be part of any plan. By contrast, our approach removes states that are very much reachable from both the initial and goal states; removing them does not preserve all plans, but rather all *optimal* plans.

Some recent work removes operators and corresponding states (Fišer, Torralba, and Shleyfman 2019; Horčík and Fišer 2021) that may be part of plans, but can still be safely ignored to preserve at least one optimal plan. This research is based on the central idea that some operators, or transitions (Haslum, Helmert, and Jonsson 2013; Torralba and Kissmann 2015), in plans may be strictly dominated by others, and thus can be safely removed. Our work can be seen as focusing on efficiently removing a subset of such dominated operators and states. Importantly, these existing approaches depend on problems having a finite state space—they often rely on "factorizing" a problem into smaller problems (Horčík and Fišer 2021; Torralba and Hoffmann 2015) and performing potentially expensive operations like symmetry-checking or constraint-satisfaction over these smaller problems. By contrast, our approach can handle infinite state spaces (as long as the number of variables and operators is finite), and Algorithm 1's complexity does not scale with the size of the state space, but rather with the number of (grounded) variables and operators.

Yet another line of work involves using abstractions to derive heuristics to guide search within the concrete problem (Culberson and Schaeffer 1998; Nebel, Dimopoulos, and Koehler 1997; Katz and Domshlak 2010). Some of these approaches can use richer families of abstractions than Algorithm 1 (for example, Cartesian abstractions). However, such approaches do not directly remove irrelevance from planning tasks, since the resulting abstractions do not necessarily preserve any valid plans. Some of this research (Rovner, Sievers, and Helmert 2019; Seipp and Helmert 2018) performs an iterative abstraction refinement similar to our approach, but interleaves planning and abstraction refinement whereas Algorithm 1 does not need to perform planning to refine its simplification.

6 Conclusion

Task scoping enables existing domain-independent planners to generalize to a much broader class of *open-scope* planning problems. By carefully removing irrelevant variables and actions from consideration, our algorithm allows planners to overcome the exponential cost of planning with large amounts of irrelevant information. This reduction in prob-

lem complexity leads to substantial improvements in planning time, even after accounting for the time spent deriving such simplifications. Moreover, planners that use these simplifications do not suffer any penalty in terms of plan quality, as all optimal plans are guaranteed to be preserved under our algorithm. This work builds on the already impressive legacy of domain-independent planners as general-purpose problem solvers, and represents an important step on the path to realizing truly general decision-making agents.

References

Alcázar, V.; and Torralba, A. 2015. A Reminder about the Importance of Computing and Exploiting Invariants in Planning. In *ICAPS*, 2–6. ISBN 9781577357315.

Bäckström, C. 1992. Equivalence and Tractability Results for SAS+ Planning. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*.

Bonet, B.; and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence*, 129(1): 5–33.

Chen, Y.; Xing, Z.; and Zhang, W. 2007. Long-Distance Mutual Exclusion for Propositional Planning. In *IJCAI*, 1840–1845.

Chentanez, N.; Barto, A. G.; and Singh, S. P. 2005. Intrinsically motivated reinforcement learning. In *NIPS*, 1281–1288.

Culberson, J. C.; and Schaeffer, J. 1998. Pattern Databases. *Computational Intelligence*, 14(3): 318–334.

De Moura, L.; and Bjørner, N. 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, 337–340. Berlin, Heidelberg: Springer-Verlag. ISBN 3-540-78799-2, 978-3-540-78799-0.

Edelkamp, S.; and Helmert, M. 1999. Exhibiting Knowledge in Planning Problems to Minimize State Encoding Length. In *ECP*, 135–147.

Fišer, D.; Torralba, Á.; and Shleyfman, A. 2019. Operator Mutexes and Symmetries for Simplifying Planning Tasks. In *AAAI*, 7586–7593.

Fox, M.; and Long, D. 2003. PDDL2. 1: An extension to PDDL for expressing temporal planning domains. *Journal of artificial intelligence research*, 20: 61–124.

Gerevini, A. E.; Haslum, P.; Long, D.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, 173(5-6): 619–668.

Haslum, P.; Helmert, M.; and Jonsson, A. 2013. Safe, Strong, and Tractable Relevance Analysis for Planning. In *ICAPS*, 317–321.

Helmert, M. 2006. The Fast Downward Planning System. *J. Artif. Int. Res.*, 26(1): 191–246.

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

Hoffmann, J.; Sabharwal, A.; and Domshlak, C. 2006. Friends or Foes? An AI Planning Perspective on Abstraction and Search. In *ICAPS*, 294–303.

Horčík, R.; and Fišer, D. 2021. Endomorphisms of Classical Planning Tasks. In *AAAI*, 11835–11843.

James, S.; Rosman, B.; and Konidaris, G. 2022. Autonomous Learning of Object-Centric Abstractions for High-Level Planning. In *International Conference on Learning Representations*.

Katz, M.; and Domshlak, C. 2010. Implicit Abstraction Heuristics. *J. Artif. Intell. Res.*, 39: 51–126.

Konidaris, G. 2019. On the necessity of abstraction. *Current Opinion in Behavioral Sciences*, 29: 1 – 7. SI: 29: Artificial Intelligence (2019).

Korf, R. E. 1985a. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1): 97–109.

Korf, R. E. 1985b. Macro-operators: A weak method for learning. *Artificial Intelligence*, 26(1): 35–77.

Long, D.; and Fox, M. 2003. The 3rd international planning competition: Results and analysis. *J. Artif. Intell. Res.*, 20: 1–59.

Long, D.; Kautz, H.; Selman, B.; Bonet, B.; Geffner, H.; Koehler, J.; Brenner, M.; Hoffmann, J.; Rittinger, F.; Anderson, C. R.; et al. 2000. The AIPS-98 planning competition. *AI magazine*, 21(2): 13–13.

Matloob, R.; and Soutchanski, M. 2016. Exploring Organic Synthesis with State-of-the-Art Planning Techniques. In *Proceedings of Scheduling and Planning Applications woRKshop (SPARK)*.

McAllester, D. A.; and Rosenblitt, D. 1991. Systematic Nonlinear Planning. In *AAAI Conference on Artificial Intelligence*.

Nebel, B.; Dimopoulos, Y.; and Koehler, J. 1997. Ignoring Irrelevant Facts and Operators in Plan Generation. In *ECP*, 338–350.

Refanidis, I.; Bassiliades, N.; Vlahavas, I.; and Greece, T. 2001. Ai Planning For Transportation Logistics. *Proceedings 17th International Logistics Conference*.

Rovner, A.; Sievers, S.; and Helmert, M. 2019. Counterexample-Guided Abstraction Refinement for Pattern Selection in Optimal Classical Planning. In *ICAPS*, 362–367.

Scala, E.; Haslum, P.; Thiebaux, S.; and Ramirez, M. 2016. Interval-Based Relaxation for General Numeric Planning. In *ECAI*, 655–663.

Scala, E.; Haslum, P.; Thiébaux, S.; and Ramírez, M. 2020. Subgoaling Techniques for Satisficing and Optimal Numeric Planning. *J. Artif. Intell. Res.*, 68: 691–752.

Seipp, J.; and Helmert, M. 2018. Counterexample-Guided Cartesian Abstraction Refinement for Classical Planning. *J. Artif. Int. Res.*, 62(1): 535–577.

Silver, T.; Chitnis, R.; Curtis, A.; Tenenbaum, J.; Lozano-Perez, T.; and Kaelbling, L. P. 2021. Planning with Learned Object Importance in Large Problem Instances using Graph Neural Networks. In *AAAI*, 11962–11971.

Torralba, A.; and Hoffmann, J. 2015. Simulation-based admissible dominance pruning. In *IJCAI*, 1689–1695.

Torralba, Á.; and Kissmann, P. 2015. Focusing on What Really Matters: Irrelevance Pruning in Merge-and-Shrink. In *SOCS*, 122–130.

Vallati, M.; and Chrpa, L. 2019. On the Robustness of Domain-Independent Planning Engines: The Impact of Poorly-Engineered Knowledge. In *K-CAP*, 197–204.

Vallati, M.; Chrpa, L.; Grześ, M.; McCluskey, T. L.; Roberts, M.; Sanner, S.; et al. 2015. The 2014 international planning competition: Progress and trends. *AI Magazine*, 36(3): 90–98.

A Results using Fast Downward's Merge and Shrink Heuristic

The full configuration we used for the merge and shrink heuristic in Table 3 is:

```
  astar(merge_and_shrink(shrink_strategy=
shrink_bisimulation(greedy=false),
merge_strategy=merge_sccs
(order_of_sccs=topological,merge_selector=
score_based_filtering
(scoring_functions[goal_relevance,dfp,
total_order])),label_reduction=
exact(before_shrinking=true,
before_merging=false),max_states=50k,
threshold_before_merge=1))
```

B Detailed Experimental Domain Descriptions

Below, we provide detailed descriptions of our different experimental environments. The PDDL files that were actually used to run these domains in our experiments are included with the provided code submission.

Numeric Planning Domains

Multi-Switch Continuous Playroom In this domain, an agent controls 3 effectors (an eye, a hand, and a marker) to interact with 6 kinds of objects (a light switch, a red button, a green button, a ball, a bell, and a monkey). The agent exists in a grid where it can take an action to move its effectors in the cardinal directions. To interact with the light switch or buttons, the agent's eye and hand effectors must be at the same grid cell as the relevant object. 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. For this particular goal, if the green buttons are already pressed in the initial state, then all green buttons, as well as all light switches, are rendered task irrelevant.

In our experiments, we created tasks within progressively larger versions of the domain by progressively increasing the number of pressed green buttons and light-switches. The optimal plan for each of these tasks is for the agent to navigate its eye and hand effectors to the ball, and navigate the marker to the bell and then throw the ball at the bell to make the monkey scream.

Composite IPC Domain This domain was constructed by simply combining the numeric *Satellites*, *Driverlog*, *Depots*, and *Zenotravel* domains from the 2002 IPC. The combination was done straightforwardly: the corresponding sections of the different domain files (i.e. `types`, `predicates`, `functions`, `operators`) were simply appended to form one combined section in the new composite domain file. To create the composite problem files, we chose one problem file from each domain and combined the objects and initial states. However, we did not combine the goals (i.e., we created four separate files that had the same composite objects and initial state, but had the goal of the corresponding original problem file from each of the four respective domains). Below, we provide a description of the dynamics of the individual domains. The dynamics of the composite domain are simply the union of those of all the individual domains.

Satellites In this domain, the agent controls a host of satellites, each equipped with various instruments. The various instruments can be switched on or off independently. They can also be calibrated by pointing them at specific calibration targets. The instruments can also take images of specific phenomena if they are calibrated. Finally, the satellite itself can be angled to point at specific phenomena. Repositioning satellites requires power, and taking readings uses up storage space, both of which are finite. Problems involve procuring images of specific phenomena with specific instruments and pointing specific satellites in particular directions.

Driverlog In the DriverLog domain, the agent must organize drivers and trucks to transport packages to specific locations. Trucks can be loaded and unloaded with packages, drivers can disembark and board other trucks, and trucks can drive only between locations that are connected. Additionally, driving or walking between locations incurs different amounts of time. Problems involve moving certain drivers, trucks and packages to specific locations with no constraints on time.

Depots In the Depots domain, the agent must controls various vehicles and cranes that must coordinate together to transport large containers from one location to another. Containers have weights and vehicles have defined load limits that cannot be exceeded. Problems involve ensuring particular containers are left in specific locations.

Zenotravel In the ZenoTravel domain, the agent must route people and airplanes to specific cities. People can board or disembark from airplanes, and planes can fly between connected cities. When flying, planes can either choose to travel at a normal speed or 'zoom', which consumes more fuel. However, zooming can only be done when the number of passengers on the plane is smaller than a prescribed amount. Moreover, planes can be refueled up to a defined capacity. Aircrafts can also be refueled at any location. Most problems require the agent to get various people and planes to specific cities.

Minecraft In this domain, the agent controls Minecraft's central playable character and can move infinitely in either the x or y directions (though all the interactable objects necessary to complete any of the agent's tasks are located in a fairly small grid in front of the agent as pictured in Figure 1). The agent possesses a diamond axe and three "blocks" of wool in the initial state, and is standing in front of a grid of plants (white flowers, oak saplings and blue flowers). The agent can "pluck" any plant by hitting it repeatedly and then replant it at any location. There is also a set of items, namely four diamonds and seven sticks, beside the grid of plants.

Problem	Operators		Expansions		Evaluations		Translate	Scoping	Planning Time (s)		Total Time (s)	
	Unscoped	Scoped	Unscoped	Scoped	Unscoped	Scoped			Unscoped	Scoped	Unscoped	Scoped
Driverlog 15	2,592	**2,112**	527,636	**460,244**	8,796,110	**7,289,831**	0.5 ± 0.0	3.6 ± 0.2	12.4 ± 0.5	$\mathbf{9.4 \pm 0.4}$	$\mathbf{12.9 \pm 0.5}$	13.5 ± 0.5
Driverlog 16	4,890	**3,540**	38,681	**29,618**	925,264	**577,118**	0.7 ± 0.0	8.4 ± 0.4	7.6 ± 0.2	$\mathbf{4.5 \pm 0.2}$	$\mathbf{8.3 \pm 0.3}$	13.6 ± 0.6
Driverlog 17	6,170	**3,770**	7,768,684	**4,607,258**	198,623,900	**88,705,990**	0.8 ± 0.0	9.5 ± 0.3	154.0 ± 4.8	$\mathbf{50.2 \pm 1.5}$	154.9 ± 4.9	$\mathbf{60.6 \pm 1.8}$
Logistics 15	650	**250**	5,951,997	**672,736**	140,607,200	**11,646,320**	0.3 ± 0.0	0.7 ± 0.0	73.7 ± 2.9	$\mathbf{11.1 \pm 0.8}$	74.0 ± 2.9	$\mathbf{12.1 \pm 0.8}$
Logistics 20	650	**250**	289,584	**86,663**	6,941,424	**1,524,997**	0.3 ± 0.0	0.7 ± 0.0	$\mathbf{7.8 \pm 0.4}$	8.6 ± 0.5	$\mathbf{8.1 \pm 0.4}$	9.5 ± 0.6
Logistics 25	650	**290**	11,437,240	**1,944,960**	265,871,100	**35,198,000**	0.3 ± 0.0	0.8 ± 0.0	148.0 ± 5.2	$\mathbf{21.5 \pm 1.1}$	148.3 ± 5.3	$\mathbf{22.5 \pm 1.1}$
Satellite 05	609	**339**	114	114	7,001	**3,950**	0.3 ± 0.1	0.6 ± 0.1	$\mathbf{3.9 \pm 0.2}$	4.5 ± 0.2	$\mathbf{4.2 \pm 0.2}$	5.5 ± 0.3
Satellite 06	582	**362**	21	21	1,084	**684**	0.3 ± 0.1	0.5 ± 0.1	$\mathbf{1.0 \pm 0.1}$	1.4 ± 0.1	$\mathbf{1.4 \pm 0.1}$	2.2 ± 0.1
Satellite 07	983	**587**	> 21,423,400	**13,438,440**	> 362,395,000	**679,734,000**	0.4 ± 0.1	0.9 ± 0.1	$> 813.3 \pm 4.2$	$\mathbf{203.2 \pm 5.6}$	$> 838.1 \pm 18.3$	$\mathbf{204.5 \pm 5.7}$
Zenotravel 10	1,155	**1,095**	1,466,718	**1,031,280**	37,782,140	**25,677,010**	0.3 ± 0.0	2.4 ± 0.0	22.1 ± 0.4	$\mathbf{16.6 \pm 0.3}$	22.5 ± 0.4	$\mathbf{19.3 \pm 0.4}$
Zenotravel 12	3,375	**3,159**	5,306,442	**3,348,866**	231,592,500	**140,117,800**	0.6 ± 0.0	7.3 ± 0.1	125.4 ± 4.9	$\mathbf{72.3 \pm 3.6}$	125.9 ± 5.0	$\mathbf{80.2 \pm 3.7}$
Zenotravel 14	6,700	**6,200**	> 6,462,279	> 5,392,118	> 168,064,648	> 130,752,144	1.0 ± 0.0	14.3 ± 0.3	$> 644.6 \pm 11.5$	$> 661.4 \pm 17.0$	$> 644.6 \pm 11.5$	$> 676.2 \pm 17.1$

Table 3: Results for our Fast Downward experiments using the Merge and Shrink heuristic. Entries beginning with > indicate that Fast Downward did not find a plan, due to an out-of-memory error. Note that Satellite 07 could only be completed when scoped, and that Logistics 25 was over 6 times as fast when using scoping.

The agent can "pick" any of these items by moving to the same location as them, and also place them at any other location. Finally, there are two wooden blocks placed ahead of the grid of plants. Unlike the plants or items, these blocks are solid objects (like the wool blocks the agent already possesses) and will obstruct the agent's path. However, the agent can destroy these blocks with its axe, which them to be automatically picked up by the agent. As with any item, the agent can then place them anywhere, whereupon they will become solid objects again.

The items within the domain can be used to "craft" various other items. If the agent possesses three blue flowers (obtained by plucking), it can invoke an action to craft a blue dye. This dye can be applied to any of the wool blocks to turn them blue. The agent can also craft a diamond axe from three diamonds and two sticks. For every wooden block the agent possesses, it can choose to craft four wooden plank blocks. Finally, the agent can craft a blue bed from three blue-dyed woolen blocks and 3 planks. Note that all these crafted items (except for the diamond axe) are items and can be picked and placed at any location. The bed and wooden planks are solid object blocks that obstruct the agent's movement and must be destroyed with the axe to be picked up and moved.

Within this domain, we defined three different tasks: (1) dye three wool blocks blue, (2) mine wood using a diamond axe and use this to craft wooden planks, and (3) craft a blue bed and place it at a specific location. To complete (1), the agent must pluck the three blue flowers from the center of the grid of plants, craft blue dye, then apply the dye to the wool blocks. To complete (2), the agent must use its axe to break one of the two wooden blocks in the domain, then invoke an action to craft planks. To accomplish (3), the agent must accomplish (1) and (2), then use three dyed wool blocks and three wooden planks to craft a bed. For task (1), the diamond axe sticks and other flowers are irrelevant by backwards reachability (Section 3). For task (2), the diamond axe is causally-linked (Section 3), the flowers, sticks and diamonds are irrelevant by backwards reachability. For task (3), the wool blocks and the diamond axe are causally-linked, and all plants other than the blue flowers, as well as the diamonds and sticks are irrelevant by backwards reachability.

IPC Domains

Here, we describe the dynamics of each of the planning domains used for Section 5 of the main paper. Note that we did not modify the dynamics of the domain whatsoever for our experiments - we only modified specific problem instances to introduce task irrelevance as described below:

Logistics In this well-known planning domain, the agent is tasked with delivering various packages to specific destination locations. To move the packages, the agent can choose to load them into a plane and fly them between locations or load them into a truck and drive them. Trucks can drive between any locations, but airplanes can only fly between locations with airports. This problem is rather similar to the Depots domain described above in Section B.

We introduced irrelevance into problems by modifying the goal so that most packages were already at their goal locations in the initial state.

DriverLog This domain is exactly the same as that described in Section B, except that there are no time costs incurred while driving or walking between locations.

We introduced irrelevance into problems by modifying the goal so that most conditions were already satisfied in the initial state.

Satellite This domain is exactly the same as that described in Section B, except that satellites do not have power or data storage limits.

We introduced irrelevance into these problems by modifying the goal so that several of its conditions were already satisfied or almost satisfied (e.g. specific instruments were already calibrated and pointing at goal phenomena) in the initial state. We also added additional instruments and satellites but not specifying any goal conditions involving these, rendering these irrelevant as well.

ZenoTravel This domain is exactly the same as that described in Section B, except there is no limit on the number of passengers that the plane can zoom with, or on the amount of fuel a plane can hold.

We introduced irrelevance into problems by modifying the goal so that many people and planes were already at their goal locations in the initial state.

C Full PDDL for example domain

Domain File

```
(define (domain toy-example)
    (:requirements :strips)

    (:predicates
        (has-food ?ag)
        (has-sticks ?ag)
        (has-stone ?ag)
        (hungry ?ag)
        (has-axe ?ag)
    )

    (:action hunt
        :parameters (?ag)
        :precondition (and
            (not (has-food ?ag))
            (not (hungry ?ag))
        )
        :effect (and (has-food ?ag))
    )

    (:action gather
        :parameters (?ag)
        :precondition (and
            (not (has-food ?ag))
            (hungry ?ag)
        )
        :effect (and (has-food ?ag))
    )

    (:action get_stick
        :parameters (?ag)
        :precondition (and
            (not (has-sticks ?ag)))
        :effect (and (has-sticks ?ag))
    )

    (:action get_stone
        :parameters (?ag)
        :precondition (and
            (not (has-stone ?ag)))
        :effect (and (has-stone ?ag))
    )

    (:action eat
        :parameters (?ag)
        :precondition (and
            (has-food ?ag)
            (hungry ?ag))
        :effect (and
            (not (has-food ?ag))
            (not (hungry ?ag))
        )
    )

    (:action make_axe
        :parameters (?ag)
        :precondition (and
            (has-sticks ?ag)
            (has-stone ?ag)
            (not (has-axe ?ag))
        )
        :effect (and
            (not (has-sticks ?ag))
            (not (has-stone ?ag))
            (has-axe ?ag)
        )
    )

    (:action wait
        :parameters (?ag)
        :precondition (and
            (not (hungry ?ag)))
        :effect (and (hungry ?ag))
    )
)
```

Problem File

```
(define (problem example-1)
    (:domain toy-example)

    (:objects steve)

    (:init
        (not (has-food steve))
        (not (has-sticks steve))
        (not (has-stone steve))
        (not (hungry steve))
        (not (has-axe steve))
    )

    (:goal (and
            (not (hungry steve))
            (has-axe steve)
        )
    )
)
```