Efficient Search by Tentatively Pruning Objects from Planning Tasks

Anita de Mello Koch[1], **Naman Shah**[2], **Cameron Allen**[3] **George Konidaris** [1]

[1]Brown University
[2]Allen Institute for AI
[3]UC Berkeley
{anita de mello koch,gdk}@brown.edu, namanshah@asu.edu, camallen@berkeley.edu

Abstract

Task planners excel at deep, multi-step plans but struggle when problems contain excessive objects. This occurs when a single domain model covers many possible tasks—derived from large language models, learned world models, or hand-authored general-purpose domains. In such settings a general-purpose agent must represent all objects across any possible task, far exceeding those required for any individual plan. We introduce tentative object pruning, an approach that leverages this property by constructing multiple simplified tasks with reduced object sets and searching them in parallel until a valid, satisficing solution is found. Our method uses type information and initial-state object relationships to preserve plan validity while substantially reducing the search space. We evaluate on three benchmark suites designed to stress planners with large object counts: expanded IPC benchmarks where object counts are systematically increased beyond standard IPC problems, Hard-to-Ground Blocksworld, and Mineplanner. Our approach maintains completeness while improving coverage by 33% and 39% for Fast Downward and Powerlifted respectively.

1 Introduction

Task planners have proven remarkably effective at solving long-horizon problems, generating plans that sequence hundreds of actions (Ghallab, Nau, and Traverso 2004). This success has led to their adoption in diverse domains including robotics, reinforcement learning and autonomous decision-making (Partalas, Vrakas, and Vlahavas 2008; Jiang et al. 2019; Guo et al. 2023; Schwarting, Alonso-Mora, and Rus 2018). Many state-of-the-art planners rely on grounding—instantiating all symbolic entities with literal objects—both for search and for computing the heuristics that guide it. However, grounding becomes increasingly challenging as object counts grow (Corrêa et al. 2021). Excessive objects inflate the number of grounded actions and dramatically expand the search space, creating a scalability bottleneck even for modern planners.

Historically, benchmarks such as those in the International Planning Competition (IPC) model only the objects essential for achieving the task objective. This deliberate minimalism enables scaling to increasing plan lengths—or search *depth*—while maintaining narrow *breadth* in the search tree. However, the growing adoption of automatically generated planning problems introduces new challenges. Problems produced by large language models, learned world models, or hand-authored domains covering many possible tasks contain far more objects than any single task requires—a single domain model must remain valid across all tasks it may be asked to solve, retaining objects that are irrelevant to any individual plan. These excess objects inflate grounding overhead and induce large branching factors in the search space, degrading search efficiency and increasing memory requirements. While lifted planners have been proposed to mitigate grounding cost (Corrêa and De Giacomo 2024; Höller and Behnke 2022), they must still reason over the full object set during planning and therefore suffer from similar bottlenecks as object counts grow. Yet in all these settings, valid plans typically require far fewer objects than the domain contains.

Planning would be substantially easier if the planner knew which objects to include during search and which can be safely ignored. We propose to address this challenge through tentative object pruning: repeatedly constructing simplified tasks over reduced object sets, retaining the original goal while reducing the branching factor. This approach acts as a preprocessing step, generating simplified planning instances that can be solved by any existing planner in parallel until a valid solution is found.

We make the following contributions. Tentative object pruning is introduced as a method for generating simplified planning tasks, leveraging type information and initial state relationships to preserve plan validity. Two pruning strategies—cost-based and random—offer different trade-offs between search efficiency and applicability to highly connected domains. The parallelizable architecture enables existing planners to leverage multi-core processors to evaluate simplified tasks concurrently, scaling to problems with large object counts. Finally, a theoretical analysis establishes that when the minimal sufficient object set is small relative to the full domain, expected search cost is exponentially lower than planning over all objects and the method is complete.

Experiments across expanded IPC benchmarks, Hard-to-Ground Blocksworld, and Mineplanner—benchmark suites specifically designed to stress planners with large object counts—show that object pruning is an effective mechanism

for scaling modern planners to large, object-dense planning problems, improving coverage by 33% and 39% for Fast Downward and Powerlifted respectively.

2 Background

We formalize classical planning in the STRIPS fragment of PDDL (Ghallab et al. 1998) and establish notation used throughout.

Definition 1 (Planning Task)**.** *A* planning task *is a tuple* $\mathcal{P} = \langle \mathcal{O}, S, s_0, S_G, A, f \rangle$ *where* $\mathcal{O}$ *is a finite set of typed objects,* S *is the state space,* $s_0 \in S$ *is the initial state,* S_G *is the goal condition,* A *is a finite set of action schemas, and* $f\colon S \times A \to S$ *is the state-transition function. The components* S*,* s_0*,* S_G*, and* f *are defined precisely below in terms of ground atoms derived from* $\mathcal{O}$*,* A*, and the predicate schemas* Φ*.*

Let $\mathcal{T}$ denote a finite set of *types*. Each object $o \in \mathcal{O}$ is assigned a type $\tau(o) \in \mathcal{T}$, which constrains the schema parameters that o may instantiate.

Definition 2 (Predicate Schema and Ground Atom)**.** *A* predicate schema $\phi \in \Phi$ *has the form* $\phi(v_1{:}\tau_1, \ldots, v_m{:}\tau_m)$*, where each* v_i *is a parameter with required type* $\tau_i \in \mathcal{T}$*. A* ground atom $\phi(o_1, \ldots, o_m)$ *is obtained by substituting every parameter* v_i *with an object* $o_i \in \mathcal{O}$ *satisfying* $\tau(o_i) = \tau_i$*. We denote by* $\mathcal{F}$ *the set of all ground atoms derivable from* Φ *and* $\mathcal{O}$*.*

A *state* $s \subseteq \mathcal{F}$ is a set of ground atoms that hold simultaneously. The initial state $s_0 \subseteq \mathcal{F}$ specifies the atoms that hold before any action is applied. A state s is a *goal state* if and only if $S_G \subseteq s$.

Definition 3 (Action Schema and Ground Action)**.** *An* action schema $a \in A$ *is specified by a tuple of typed parameters* $(v_1{:}\tau_1, \ldots, v_\alpha{:}\tau_\alpha)$ *of arity* α*, a set of preconditions* $\mathrm{pre}(a) \subseteq \Phi$*, and effects* $\mathrm{eff}(a) = \mathrm{eff}^+(a) \cup \mathrm{eff}^-(a)$ *partitioned into add effects and delete effects. A* ground action $a[\sigma]$ *is produced by a type-consistent substitution* $\sigma\colon \{v_1, \ldots, v_\alpha\} \to \mathcal{O}$*, instantiating all parameter occurrences in* $\mathrm{pre}(a)$ *and* $\mathrm{eff}(a)$ *with objects from* $\mathcal{O}$*.*

A ground action a is *applicable* in state s if and only if $\mathrm{pre}(a) \subseteq s$. Applying an applicable action a in state s yields the successor state

$$f(s, a) = \big(s \setminus \mathrm{eff}^-(a)\big) \cup \mathrm{eff}^+(a). \tag{1}$$

Definition 4 (Plan)**.** *A* plan *for task* $\mathcal{P}$ *is a sequence of ground actions* $\pi = \langle a_1, \ldots, a_\ell \rangle$ *such that each* a_{i+1} *is applicable in* $s_i = f(s_{i-1}, a_i)$ *with* s_0 *the initial state, and the final state* s_ℓ *satisfies* $S_G \subseteq s_\ell$*. A planning task is* solvable *if at least one such plan exists.*

Grounding. Many state-of-the-art planners *ground* a task by enumerating all type-consistent substitutions for each action schema to produce the full set of ground actions. For a schema of arity α, this yields $O(|\mathcal{O}|^\alpha)$ ground instances, making the total number of ground actions polynomial in $|\mathcal{O}|$ but exponential in α (Corrêa et al. 2021). As object counts grow, grounding becomes a computational bottleneck and simultaneously inflates the branching factor during search.

We characterize planning task complexity along two dimensions. *Problem depth* is the length of a shortest plan from s_0 to any goal state. *Problem breadth* is the maximum number of applicable ground actions in any reachable state, which for grounded planners grows as $O(|\mathcal{O}|^\alpha)$, directly coupling object count to search difficulty.

3 Related Works

Several approaches have been proposed to address scalability challenges in planning as problem size grows.

Lifted Planning Lifted planners operate over first-order representations, bypassing full grounding and enabling scalability in domains with large object sets. Recent work has demonstrated the feasibility of symbolic reasoning without instantiation (Corrêa and De Giacomo 2024). Höller and Behnke (2022) leverage propositional logic with SAT solvers to avoid combinatorial explosion during grounding. Powerlifted (Corrêa et al. 2023) uses database-style representations and conjunctive queries for scalable lifted search. While these methods mitigate grounding costs, they must still search over the full object set during planning. As our experiments show, lifted planners face similar scalability bottlenecks as grounded planners when object counts grow large, even without grounding overhead.

Heuristic Methods Heuristics guide planning by estimating goal distance, effectively reducing the branching factor explored during search. Traditional heuristics such as h_{max}, h_{add} (Bonet and Geffner 2001) and Fast Forward (Hoffmann and Nebel 2001) are computed over ground representations, requiring full instantiation before search begins (Corrêa et al. 2021; Helmert and Domshlak 2009). McDermott (1999) proposed regression-match graphs, which estimate goal distance by regressing goal literals through action schemas and heuristically binding free variables to whichever objects make the most goal conjuncts simultaneously true—acknowledging that when several objects can satisfy the same objectives in some plan, the choice of which object to leverage affects search difficulty. This matching heuristic commits to a single object binding per goal at each search step, with no mechanism to reconsider it, and is consequently incomplete. Lifted heuristics avoid grounding but must still reason over all available objects to evaluate applicability (Wichlacz, Höller, and Hoffmann 2022; Ridder and Fox 2014). As object counts grow, both successor generation and heuristic evaluation become bottlenecks (Ståhlberg 2023).

Landmark-based heuristics (Richter and Westphal 2010) and abstraction techniques such as merge-and-shrink (Helmert et al. 2014) compress the state space while preserving relevant structure, reducing the number of nodes expanded during search (Helmert and Domshlak 2009). However, these methods rely on preprocessing steps that scale poorly with object counts and their effectiveness degrades when the number of potential landmarks or abstract states becomes too large (Büchner et al. 2024; Sievers 2018).

Relevance Analysis and Pruning To reduce grounding overhead, many planners use relevance analysis during pre-

processing to eliminate objects, ground atoms, and actions unlikely to contribute to solutions. These methods are solution preserving: they remove only facts, objects or actions that are provably incapable of contributing to any plan, so the retained task admits a solution whenever the original task does. An early precursor to relevance analysis (Nebel, Dimopoulos, and Koehler 1997) is not solution preserving, selecting facts to retain by backchaining from the goal while ignoring action conflicts which may result in discarding facts that may be required by a valid plan. Fast Downward (Helmert 2006) performs goal relevance pruning by discarding unreachable facts and actions before grounding. Lang and Toussaint (2009) use causal graphs to filter out objects not involved in any causal paths to the goal. Fishman et al. (2023) introduce task scoping which simplifies open-scope problems by removing irrelevant variables and actions through backward reachability analysis.

While these methods reduce the problem size they are necessarily conservative—eliminating only objects that provably cannot contribute to any solution. In domains with large object sets where many objects of the same type exist, relevance analysis must preserve all interchangeable objects as all these objects appear in a valid plan. This limits scalability when most objects of relevant types are unnecessary for individual plans.

3.1 Partial Grounding

A related line of work reduces grounding costs by pruning which facts or operators are instantiated during grounding. Areces et al. (2014) reduce grounding blowup structurally, by automatically splitting high-arity action schemas into multiple lower-arity schemas, directly shrinking the number of ground actions. Gnad et al. (2019) instead compute a priority ordering over operators using a learned relational model trained to predict whether an operator is included in an optimal plan, and instantiate operators in that order until a resource budget is exhausted or a delete-relaxed goal-reachability test succeeds; this requires training data obtained by solving small instances with symbolic search, and a separate model per action schema. This process runs in a loop, creating new groundings until a solution is found. Most recently, Canonaco, Pozanco, and Borrajo (2026) use large language models to analyze the domain and problem files, heuristically pruning objects, actions and predicates prior to grounding without a learned relational model or training data. These methods operate over facts or operators and so do not fully leverage the properties of substitutable objects, which object pruning is designed to leverage.

These prior works operate over facts or operators whereas object pruning treats object selection as a repeated, independently-sampled process: many candidate object sets are constructed and solved in parallel. Fact- and operator-level relevance is largely structural—a fact either can or cannot support a causal path to the goal—and can often be resolved by deterministic filtering or a single learned priority. Object-level relevance, by contrast, is frequently a question of *which* of several interchangeable objects a plan will use, not *whether* objects of that type are needed at all. This is precisely the setting where independent, parallel sampling over candidate subsets, rather than a single growing or one-shot projection, pays off—and it is the problem setting object pruning is designed to address.

4 Pruning Object Sets for Simplified Tasks

Consider a planning task $\mathcal{P} = \langle \mathcal{O}, S, s_0, S_G, A, f \rangle$ in which every valid plan uses only a subset $\mathcal{O}^* \subset \mathcal{O}$ of the available objects. Planning problems produced by automated methods (Guan et al. 2023; Asai and Fukunaga 2018; Mahdavi et al. 2024; Hill et al. 2023; Konidaris, Kaelbling, and Lozano-Perez 2018) routinely exhibit $|\mathcal{O}| \gg |\mathcal{O}^*|$: the domain provides far more objects than any single task requires. Because grounding yields $O(|\mathcal{O}|^\alpha)$ actions per schema of arity α (Definition 3), these excess objects inflate both grounding cost and the branching factor during search, even though they appear in no solution.

This observation suggests a natural strategy: construct a *simplified task* over a reduced object set $\mathcal{O}_P \subset \mathcal{O}$ satisfying $\mathcal{O}^* \subseteq \mathcal{O}_P$ and $|\mathcal{O}_P| \ll |\mathcal{O}|$. The resulting task has branching factor $O(|\mathcal{O}_P|^\alpha) \ll O(|\mathcal{O}|^\alpha)$, yielding a substantially easier search problem. However, $\mathcal{O}^*$ is unknown a priori: conservative relevance analysis retains all objects whose types appear in goal-directed action sequences, while aggressive pruning risks removing objects required by every valid plan. We therefore propose *tentative object pruning*: rather than solving $\mathcal{P}$ directly, we generate a sequence of candidate object subsets $\mathcal{O}_{P_1}, \mathcal{O}_{P_2}, \ldots$ and solve the corresponding simplified tasks in parallel until a plan valid in $\mathcal{P}$ is found. Each simplified task preserves the goal condition S_G while reducing the branching factor, and parallel evaluation amortizes the cost of unsuccessful candidates across available computational resources.

4.1 Defining the Object Sampling Space

Not every object in $\mathcal{O}$ can participate in a goal-directed action sequence. We first restrict attention to those that can, defining a *relevant object set* $\mathcal{O}_R \subseteq \mathcal{O}$ from which all candidate subsets are drawn. $\mathcal{O}_R$ is computed without grounding via backward reachability analysis at the *lifted* level—over predicate and action schemas rather than ground atoms. The analysis rests on two mutually recursive notions of relevance.

Definition 5 (Relevant Predicate Schema)**.** *A predicate schema $\phi \in \Phi$ is* relevant*, written $\phi \in \Phi_R$, if either* (i) *ϕ appears in S_G, or* (ii) *$\phi \in \mathrm{pre}(a)$ for some relevant action schema $a \in A_R$.*

Definition 6 (Relevant Action Schema)**.** *An action schema $a \in A$ is* relevant*, written $a \in A_R$, if $\mathrm{eff}(a) \cap \Phi_R \neq \varnothing$, i.e., at least one of its effects achieves a relevant predicate.*

Because Φ_R and A_R are mutually dependent, they are computed jointly by the fixed-point iteration in Algorithm 1: starting from the predicates in S_G, the procedure alternately expands A_R with schemas whose effects achieve a predicate in Φ_R, and Φ_R with preconditions of newly added schemas, terminating when neither set grows. The relevant object set

is then

$$\mathcal{O}_R = \big\{o \in \mathcal{O} \mid \tau(o) \text{ appears as a parameter type in } \Phi_R \text{ or } A_R\big\}. \quad (2)$$

Algorithm 1: Compute Object Sampling Space

Require: P $\leftarrow (\mathcal{O}, S, s_0, S_G, A, f)$
  Initialize $\Phi_R \leftarrow \{\phi | \phi$ appears in $S_G\}$
  Initialize $A_R \leftarrow \varnothing$
  $\delta_{\Phi_R} \leftarrow |\Phi_R|$, $\delta_{A_R} \leftarrow |A_R|$
  while $\delta_{\Phi_R} > 0$ or $\delta_{A_R} > 0$ **do**
    $\Phi_{R\text{old\_size}} \leftarrow |\Phi_R|$,$A_{R\text{old\_size}} \leftarrow |A_R|$
    $A_R \leftarrow A_R \cup \{a \in A |$ eff$(a) \cap \Phi_R \neq \varnothing\}$
    $\Phi_R \leftarrow \Phi_R \cup \{\phi | \phi \in$ pre(a) for some $a \in A_R\}$
    $\delta_{\Phi_R} \leftarrow |\Phi_R| - \Phi_{R\text{old\_size}}$, $\delta_{A_R} \leftarrow |A_R| - A_{R\text{old\_size}}$
  end while
  $\mathcal{O}_R \leftarrow \{o \in \mathcal{O} | \tau(o)$ appears as parameter type in Φ_R or $A_R\}$
  return $\mathcal{O}_R$

Because this analysis operates on schemas rather than ground atoms, it runs in time polynomial in $|\Phi|$ and $|A|$, independent of $|\mathcal{O}|$. However, $\mathcal{O}_R$ retains *all* objects whose type matches any relevant schema parameter; when many objects share the same type, $\mathcal{O}_R$ may offer little reduction over $\mathcal{O}$. The sampling strategies in Sections 4.3–4.4 address this limitation by constructing candidate subsets of $\mathcal{O}_R$.

While $\mathcal{O}_R$ restricts attention to objects whose types participate in goal-directed action sequences, it does not guarantee inclusion of objects appearing explicitly in the goal condition, which must feature in every valid plan. Therefore, we additionally identify objects that must belong to every valid plan.

Definition 7 (Required Objects)**.** *The set of* required objects *is* $\mathcal{O}_N = \{o \in \mathcal{O} \mid \exists\, \phi(o_1, \ldots, o_m) \in S_G$ *with* $o \in \{o_1, \ldots, o_m\}\}$.

Since every goal state s must satisfy $S_G \subseteq s$, no plan can avoid the objects named in S_G; hence $\mathcal{O}_N$ is included in every candidate object set. Note that $\mathcal{O}_N$ is necessary but not in general sufficient: a valid plan may require additional objects that do not appear in S_G but are needed to satisfy action preconditions along the plan.

4.2 Preserving Initial State Relationships

Removing objects from $\mathcal{O}$ also removes every ground atom in which they participate. If a candidate set $\mathcal{O}_P$ omits an object that co-occurs with a selected object in some initial-state atom, the simplified task may lack preconditions necessary for actions along a valid plan. We therefore define a closure operator that augments any candidate set with all co-occurring objects from s_0.

Definition 8 (Relevantly Linked Objects)**.** *Objects* $o, o' \in \mathcal{O}$ *are* relevantly linked *if there exists a ground atom* $\phi(o_1, \ldots, o_m) \in s_0$ *such that* $\{o, o'\} \subseteq \{o_1, \ldots, o_m\}$. *For any subset* $\mathcal{O}_i \subseteq \mathcal{O}$*, the* relevant-link closure *is*

$$\mathcal{O}_L(\mathcal{O}_i) = \big\{o \in \mathcal{O} \mid \exists\, o' \in \mathcal{O}_i \text{ s.t. } o \text{ and } o' \text{ are relevantly linked}\big\}.$$

By construction, any ground atom $\phi(o_1, \ldots, o_m) \in s_0$ that mentions at least one object in $\mathcal{O}_P$ has all of its arguments contained in $\mathcal{O}_P \cup \mathcal{O}_L(\mathcal{O}_P)$. Consequently, the simplified task built from $\mathcal{O}_P \cup \mathcal{O}_L(\mathcal{O}_P)$ preserves every initial-state atom reachable from the selected objects: if a plan π is valid in the simplified task, the same action sequence is applicable in $\mathcal{P}$ and reaches the same goal state, so π is also valid in $\mathcal{P}$.

4.3 Constructing Simplified Planning Tasks

Given the relevant object set $\mathcal{O}_R$ and the required objects $\mathcal{O}_N$ (Section 4.1), we describe two strategies for constructing candidate object sets $\mathcal{O}_P$ from which simplified tasks are built.

Algorithm 2: Cost-Based Object Pruning

Require: $\mathcal{O}_N$, $\mathcal{O}_R$, previous_sets, max_attempts
  $\mathcal{O}_P \leftarrow \mathcal{O}_N$
  attempts $\leftarrow 0$
  scope_size[type] $\leftarrow 1$ for all types
  while attempts $<$ max_attempts **do**
    attempts $\leftarrow$ attempts + 1
    $\mathcal{O}_P \leftarrow \mathcal{O}_N$
    for all type in object_types **do**
      Add scope_size[type] lowest-cost objects of type to $\mathcal{O}_P$
    end for
    $\mathcal{O}_P \leftarrow \mathcal{O}_P \cup \mathcal{O}_L(\mathcal{O}_P)$
    if $\mathcal{O}_P \notin$ previous_sets **then**
      return $\mathcal{O}_P$
    end if
    Increment scope_size for next type in round-robin order
  end while
  return $\emptyset$

Cost-Based Object Pruning Each object $o \in \mathcal{O}_R$ is assigned a *linking cost* $l(o) = |\mathcal{O}_L(\{o\})|$, measuring how many additional objects the relevant-link closure (Section 4.2) forces into any candidate set containing o. Cost-based pruning greedily selects, for each relevant type, the objects with lowest linking cost. As detailed in Algorithm 2, each candidate begins from the required set $\mathcal{O}_N$ and includes one object per type; after each unsuccessful attempt, the per-type quota is incremented in round-robin order. The resulting sequence of candidate sets grows monotonically in size, evaluating the narrowest search spaces first—analogous to iterative deepening over problem breadth rather than depth (Korf 1985). Like iterative deepening, this strategy ensures that if a small object set suffices, it will be found before larger, more expensive sets are evaluated.

Random Object Pruning As an alternative to the cost-based heuristic, candidate sets can be constructed by uniformly sampling objects of each relevant type from $\mathcal{O}_R$ (Algorithm 3). We distinguish two variants. *Safe* random pruning augments each candidate with the relevant-link closure $\mathcal{O}_L(\mathcal{O}_P)$ (Section 4.2), preserving plan validity. In highly connected domains, however, the closure may add so many objects that the candidate offers little reduction over $\mathcal{O}$. *Unsafe* random pruning omits the closure step, enabling more aggressive reduction of $|\mathcal{O}_P|$ at the expense of guaranteed validity: any plan discovered in the simplified task must be validated against the original task $\mathcal{P}$ before search terminates.

Algorithm 3: Random Object Pruning

Require: $\mathcal{O}_R, \mathcal{O}_N$, safe_flag, previous_sets, object_types, max_attempts
  $\mathcal{O}_P \leftarrow \mathcal{O}_N$
  attempts $\leftarrow 0$
  while attempts $<$ max_attempts **do**
    attempts $\leftarrow$ attempts + 1
    for all type in object_types **do**
      sample_size$\leftarrow$random_int
      $\mathcal{O}_S \leftarrow$ uniform_sample($\{\mathcal{O}_R | o \in \mathcal{O}_R$ has type(o) = object_type$\}$, sample_size)
      $\mathcal{O}_P \leftarrow \mathcal{O}_P \cup \mathcal{O}_S$
    end for
    if safe_flag is True **then**
      $\mathcal{O}_P \leftarrow \mathcal{O}_P \cup \mathcal{O}_L(\mathcal{O}_P)$
    end if
    if $O_P \notin$previous_sets **then**
      return $\mathcal{O}_P$
    end if
  end while
  return $\emptyset$

4.4 Parallel Simplified Task Evaluation

Given a candidate object set $\mathcal{O}_P \subseteq \mathcal{O}$ produced by one of the strategies above, we construct a simplified task.

Definition 9 (Simplified Task)**.** *Given a planning task $\mathcal{P} = \langle \mathcal{O}, S, s_0, S_G, A, f \rangle$ and a subset $\mathcal{O}_P \subseteq \mathcal{O}$ with $\mathcal{O}_N \subseteq \mathcal{O}_P$, the* simplified task *is*

$$\mathcal{P}^S = \langle \mathcal{O}_P,\ S',\ s'_0,\ S_G,\ A',\ f' \rangle$$

where $s'_0 = \{\phi(o_1, \ldots, o_m) \in s_0 \mid o_1, \ldots, o_m \in \mathcal{O}_P\}$ restricts the initial state to atoms whose arguments all lie in $\mathcal{O}_P$, A' retains only action schemas whose type-consistent ground instances draw all parameters from $\mathcal{O}_P$, and S' and f' are induced accordingly.

Because each simplified task $\mathcal{P}^S$ is a self-contained planning task, multiple instances can be solved concurrently. We implement this via a coordinator-worker architecture: a coordinator generates candidate object sets using Algorithm 2 or 3 and dispatches the corresponding tasks $\mathcal{P}^S$ to worker processes. Search terminates when any worker returns a plan π valid in $\mathcal{P}$, or when no further candidate sets can be generated. For safe pruning (Section 4.2), any plan found in $\mathcal{P}^S$ is guaranteed valid in $\mathcal{P}$; under unsafe pruning, workers validate discovered plans against the full task before reporting success.

The procedure is *satisficing*: the candidate set yielding an optimal plan for $\mathcal{P}$ may never be selected. It is, however, *complete*: candidate sets grow monotonically, so the sequence eventually produces $\mathcal{O}_P = \mathcal{O}$, recovering $\mathcal{P}$ itself (formalized in Theorem 2). In practice, $\mathcal{P}$ can be solved by a dedicated worker in parallel with the simplified tasks, ensuring performance no worse than baseline planning.

4.5 Theoretical Properties of Object Pruning

We analyze the expected search cost and completeness of object pruning under simplifying assumptions. Let $n = |\mathcal{O}|$ and let α denote the maximum arity of any action schema in A. We model the search induced by a task with m objects as a uniform tree of depth d (the length of a shortest plan) and branching factor $O(m^\alpha)$, so the tree contains $O(m^{\alpha d})$ nodes. Searching the original task $\mathcal{P}$ therefore requires expanding $O(n^{\alpha d})$ nodes in the worst case, whereas a simplified task $\mathcal{P}_i^S$ with $|\mathcal{O}_i| = i < n$ objects induces a tree of size $O(i^{\alpha d})$.

Let $k = |\mathcal{O}^*|$ denote the size of a smallest sufficient object set. Consider the sequence of simplified tasks $\mathcal{P}_1^S, \mathcal{P}_2^S, \ldots, \mathcal{P}_n^S$ with monotonically growing object sets $|\mathcal{O}_1| < |\mathcal{O}_2| < \cdots < |\mathcal{O}_n| = n$, as produced by cost-based pruning. Each $\mathcal{P}_i^S$ is searched independently until a plan valid in $\mathcal{P}$ is found. Let p_i denote the probability that an object subset of size i contains $\mathcal{O}^*$, so that $\mathcal{P}_i^S$ admits a valid plan.

Theorem 1 (Expected Cost of Object Pruning)**.** *The simplified task $\mathcal{P}_i^S$ induces a search tree of size at most $t_i = i^{\alpha d}$. The expected total number of node expansions before object pruning finds a valid plan is*

$$\mathbb{E}[\mathrm{OP}] = \sum_{i=1}^{n} p_i \cdot t_i = \sum_{i=1}^{n} p_i \cdot i^{\alpha d}.$$

If $k \ll n$ and p_i is sharply concentrated around $i \approx k$ (i.e., $p_i \approx 0$ for $i \gg k$), then $\mathbb{E}[\mathrm{OP}] \ll n^{\alpha d}$.

Proof. Suppose for some constant $\gamma > 1$ the probabilities satisfy

$$\sum_{i=1}^{\gamma k} p_i \approx 1 \quad \text{and} \quad \sum_{i=\gamma k+1}^{n} p_i \leq \epsilon \ll 1.$$

Splitting the expected cost at index γk:

$$\mathbb{E}[\mathrm{OP}] = \sum_{i=1}^{\gamma k} p_i \cdot i^{\alpha d} + \sum_{i=\gamma k+1}^{n} p_i \cdot i^{\alpha d}.$$

For the first sum, $i \leq \gamma k$ gives

$$\sum_{i=1}^{\gamma k} p_i \cdot i^{\alpha d} \leq (\gamma k)^{\alpha d} \cdot \sum_{i=1}^{\gamma k} p_i \leq (\gamma k)^{\alpha d}.$$

For the second sum, $i^{\alpha d} \leq n^{\alpha d}$ gives

$$\sum_{i=\gamma k+1}^{n} p_i \cdot i^{\alpha d} \leq n^{\alpha d} \cdot \epsilon.$$

Combining: $\mathbb{E}[\text{OP}] \leq (\gamma k)^{\alpha d} + \epsilon \cdot n^{\alpha d}$. Since $k \ll n$ and $\epsilon \ll 1$, we obtain $\mathbb{E}[\text{OP}] \ll n^{\alpha d}$. □

Theorem 2 (Completeness of Object Pruning)**.** *If $\mathcal{P}$ is solvable, then the object pruning procedure finds a plan valid in $\mathcal{P}$.*

Proof. Every candidate set satisfies $\mathcal{O}_N \subseteq \mathcal{O}_{P_i}$, and the sets are constructed so that $|\mathcal{O}_{P_1}| < |\mathcal{O}_{P_2}| < \cdots$. In the limit, $\mathcal{O}_{P_n} = \mathcal{O}$, so the simplified task $\mathcal{P}_n^S$ coincides with the original task $\mathcal{P}$. Because $\mathcal{P}$ is solvable, the procedure finds a valid plan no later than iteration n. □

5 Results

5.1 Experiment Setup

We evaluate object pruning across three benchmark suites. The first expands the IPC benchmarks to isolate the effect of object count on planning performance. Given a problem with object set $\mathcal{O}$ and initial state s_0, we create an expanded problem by duplicating $\mathcal{O}$ into $\mathcal{O}'$ and replicating all ground predicates from s_0, substituting objects from $\mathcal{O}'$. For example, if $\phi(o_a, o_b) \in s_0$, then $\phi(o_a{}', o_b{}')$ is added to the expanded initial state. The goal condition remains unchanged so the solution length is not lengthened by this process. We apply this expansion to the smallest problems from selected IPC domains, focusing on domains where FD's relevance analysis does not fully eliminate the expanded objects during preprocessing.

The second benchmark is HTG Blocksworld from the Hard-to-Ground benchmark suite (Corrêa and Seipp 2022; Lauer et al. 2021) which increases the block count—with some problems containing thousands of objects—while maintaining the standard Blocksworld structure. We focus on HTG Blocksworld specifically because its large block counts stress planners through object scale rather than plan complexity, isolating the bottleneck that object pruning targets. Other HTG domains stress different bottlenecks such as large action arities, which are outside the scope of this work.

The third benchmark is Mineplanner (Hill et al. 2023), which algorithmically generates PDDL problems from Minecraft game instances, modeling the object counts of a real environment where object sets arise naturally from domain modeling rather than careful design. The benchmark contains three difficulty levels, with each difficulty adding additional objects, with the easiest problem still containing over 400 objects.

All experiments use Fast Downward (FD) (Helmert 2006) configured with lazy greedy best-first search using the FF heuristic (Hoffmann and Nebel 2001) with preferred operators — a standard satisficing configuration — and Powerlifted (Corrêa et al. 2023) using the Yannakakis generator. Object pruning uses identical planner configurations to the baselines: simplified tasks are solved by the same FD or Powerlifted instance, so any properties of the search configuration affect both baseline and object pruning runs equally. For expanded IPC we evaluate cost-based and safe random object set pruning, HTG Blocksworld evaluates cost-based object pruning and Mineplanner evaluates unsafe random object pruning.

Table 1: Per-domain coverage for Fast Downward (FD) with and without object pruning on expanded IPC and HTG Blocksworld benchmarks. The total number of problems is denoted n, Δ_{cost} and Δ_{random} reports the absolute coverage gain over the FD baseline for cost-based and random pruning respectively.

Domain	n	FD	OP (cost)	OP (rand)	Δ_{cost}	Δ_{rand}
airport-adl	48	33	48	46.3	+15.0	+13.3
assembly	49	41	49	48.0	+8.0	+7.0
barman-opt14-strips	48	18	43	44.7	+25.0	+26.7
blocks	56	52	56	55.3	+4.0	+3.3
blocks-3op	49	29	40	44.7	+11.0	+15.7
briefcaseworld	56	40	56	55.3	+16.0	+15.3
caldera-split-opt18	56	20	56	55.7	+36.0	+35.7
citycar-opt14-adl	56	35	54	54.7	+19.0	+19.7
depot	56	22	48	47.7	+26.0	+25.7
floortile-opt14-strips	56	11	17	18.0	+6.0	+7.0
freecell	112	53	111	111.0	+58.0	+58.0
fridge	79	23	73	74.7	+50.0	+51.7
goal-2	10	3	10	10.0	+7.0	+7.0
goal-3	10	3	10	10.0	+7.0	+7.0
goal-4	10	3	10	10.0	+7.0	+7.0
goal-5	10	3	10	10.0	+7.0	+7.0
grid	40	14	30	30.0	+16.0	+16.0
gripper	56	38	55	55.3	+17.0	+17.3
hiking-opt14-strips	56	16	55	55.0	+39.0	+39.0
logistics98	56	30	46	51.0	+16.0	+21.0
mprime	56	41	55	53.3	+14.0	+12.3
no-mprime	56	41	55	53.7	+14.0	+12.7
nurikabe-opt18	56	41	53	55.0	+12.0	+14.0
pipesworld-06	56	56	54	54.7	-2.0	-1.3
settlers-opt18	69	46	68	65.0	+22.0	+19.0
spider-opt18	69	31	51	52.7	+20.0	+21.7
thoughtful-sat14-strips	72	38	63	64.0	+25.0	+26.0
trucks	56	44	51	49.3	+7.0	+5.3
tyreworld	56	52	56	55.7	+4.0	+3.7
woodworking-opt11-strips	56	36	54	52.7	+18.0	+16.7
zenotravel	56	35	55	55.0	+20.0	+20.0
TOTAL	1627	948	1492	1498.5	+544.0	+550.5

All runs share the same total memory budget determined by the available hardware. For object pruning, each of the 64 parallel worker threads is allocated a share of this total budget; the aggregate memory across all threads does not exceed what is available to the baseline planner. Each worker thread is limited to 10 minutes and 4GB of memory per simplified task, with a 30-minute overall limit per problem.

All reported runtimes include preprocessing, grounding, and search time; for object pruning this includes object set generation and all parallel search attempts. Since object pruning evaluates simplified tasks in parallel, CPU time is reported as the sum of time across all concluded worker processes up to the point a valid solution is found—equivalent to the time the system would have taken had each simplified task been evaluated sequentially. This means reported runtimes fairly reflect total computational effort expended rather than wall-clock time.

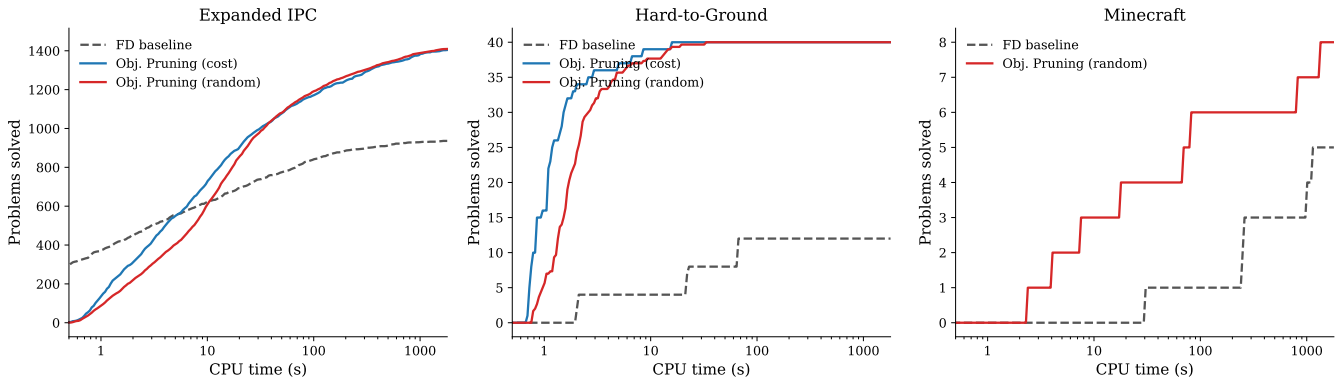

Figure 1: Coverage as a function of cumulative CPU time for Fast Downward across expanded IPC, HTG Blocksworld, and Mineplanner benchmarks. CPU time is the sum across all worker processes up to the point a valid solution is found. CPU-time is reported using a logarithmic scale.

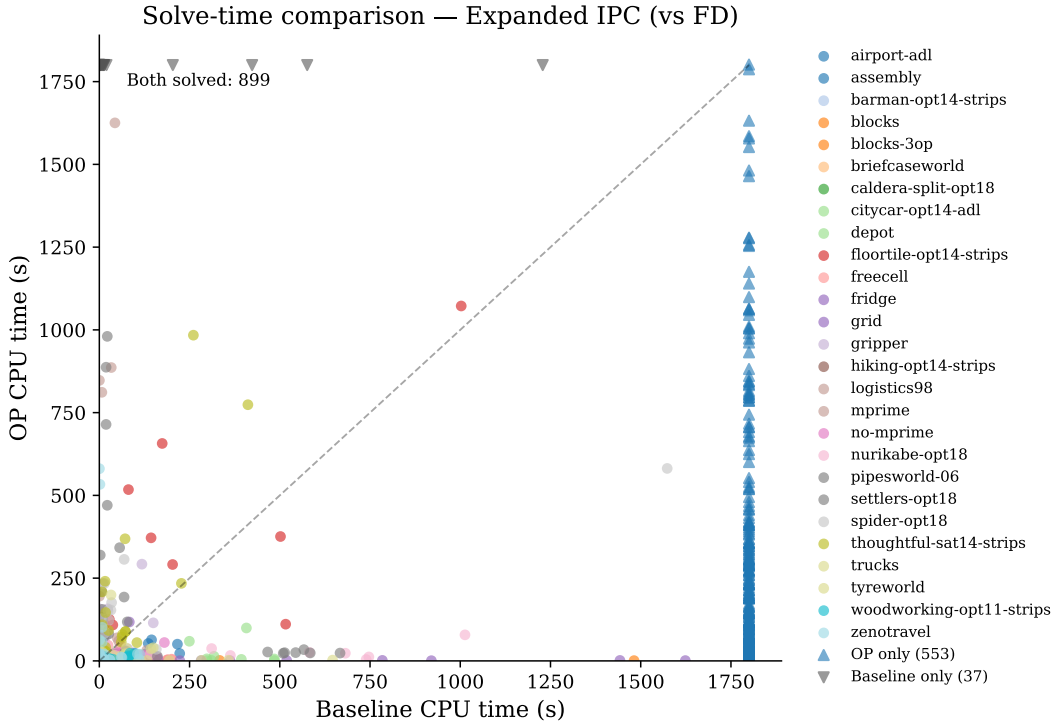

Figure 2: Solve-time comparison between FD baseline and cost-based object pruning on expanded IPC benchmarks. Points below the diagonal indicate faster solve times under object pruning.

5.2 Object Pruning for Improved Coverage

Figure 1 shows coverage as a function of CPU time across all benchmarks for FD and FD with object pruning. Object pruning substantially increases the number of problems solved within the resource limits. These results demonstrate that large object counts degrade coverage independently of problem depth—FD solves 100% of the original unmodified IPC problems but coverage drops to between $32\% - 93\%$ on the expanded versions despite unchanged goal complexity, demonstrating that the branching factor introduced by excess objects is sufficient to make previously tractable problems intractable.

Table 1 shows per-domain coverage for FD with and without pruning. Cost-based pruning increases the total problems solved from 948 to 1492 out of 1627, an improvement of 544 problems (33% relative increase), with random pruning achieving comparable gains at 1498.5, bringing total coverage from 58% to 92%. These gains hold across both the expanded IPC domains and HTG Blocksworld (Figure 1, center), confirming that object pruning enables planning in

Table 2: Per-domain coverage for Powerlifted with and without object pruning on expanded IPC and HTG Blocksworld benchmarks. The total number of problems is denoted n, Δ_{cost} reports the absolute coverage gain over the Powerlifted baseline for cost-based object pruning.

Domain	n	Powerlifted	OP (cost)	Δ_{cost}
barman-opt14-strips	48	30	48	+18.0
blocks	56	41	56	+15.0
blocks-3op	48	38	47	+9.0
depot	55	26	53	+27.0
freecell	112	63	112	+49.0
goal-2	10	7	10	+3.0
goal-3	10	2	10	+8.0
goal-4	10	1	10	+9.0
goal-5	10	0	10	+10.0
grid	38	16	38	+22.0
gripper	55	41	54	+13.0
hiking-opt14-strips	56	27	56	+29.0
logistics98	54	24	54	+30.0
mprime	56	32	56	+24.0
no-mprime	56	32	56	+24.0
pipesworld-06	56	46	56	+10.0
thoughtful-sat14-strips	65	26	64	+38.0
tyreworld	56	36	56	+20.0
woodworking-opt11-strips	56	45	56	+11.0
zenotravel	56	45	56	+11.0
TOTAL	963	578	958	+380.0

problems that are intractable for grounded planners due to object scale.

Pipesworld is a notable exception where pruning does not provide a benefit for FD. In this domain each pipe connects specific locations and cannot be substituted by another pipe of the same type. Our expansion process preserves this non-interchangeable structure, resulting in multiple sets of locations and location-specific pipes where each set remains isolated. Because objects are not interchangeable, the probability of a pruned set containing all required objects is not sharply concentrated near the minimal sufficient set size—the condition required for the expected cost savings established in Theorem 1. This confirms that the effectiveness of object pruning depends on the degree to which objects are substitutable, and that Theorem 1's assumptions are predictive of where the method will and will not provide benefit.

To evaluate whether object pruning extends to lifted planners, we additionally report coverage results for Powerlifted in Table 2. Powerlifted without pruning achieves less than full coverage on all domains, demonstrating that lifted representations alone are insufficient when object counts grow large—even without grounding overhead, reasoning over the full object set during search remains a bottleneck. Cost-based pruning increases total problems solved from 578 to 958 out of 963, a gain of 380 problems (39% relative increase) and near-perfect coverage across all reported domains. Notably, on the HTG Blocksworld goal complexity variants—where Powerlifted without pruning solves as few as none of the problems for the hardest goals—cost-based pruning recovers perfect coverage, showing that object pruning addresses the object count scalability bottleneck for lifted planners as effectively as for grounded ones.

5.3 Runtime of Object Pruning

Figure 2 shows solve-time comparisons between FD baseline and cost-based object pruning. For problems solved by both methods, solve times are broadly comparable — object pruning is faster on some instances while the baseline outperforms on others. Since the total machine memory is identical across all runs and object pruning's parallel threads collectively operate under the same budget as the baseline, these differences reflect genuine reductions in search effort from the reduced branching factor rather than any relaxation of resource constraints.

For problems solved only by object pruning, the baseline exhausted available time or memory without finding a solution. The runtime advantage in these cases is not merely a speedup but a qualitative change in what is computationally feasible: by reducing the branching factor, simplified tasks can be grounded and searched within the available budget where the original problem cannot.

Mineplanner illustrates this most starkly. Powerlifted exhausts available machine memory for all Mineplanner problems, while FD exhausts system memory on all failed problems. Object pruning with unsafe random sampling enables FD to solve several previously intractable problems by constraining each simplified task to a small enough object set to ground and search within the per-thread memory budget, as shown in Figure 1 (right). Coverage remains partial however, reflecting the difficulty of the sampling problem in this domain—a valid object set must simultaneously include objects required to achieve the goal, sufficient terrain for navigation, and relevant obstacles, all while remaining small enough to ground within the thread budget. Unlike the expanded IPC and HTG settings where the cost-based heuristic effectively identifies small sufficient object sets, Mineplanner requires unsafe random sampling with many attempts, pointing to the development of better heuristics for highly connected domains as an important direction for future work. However, even using unsafe random object set pruning results in improved coverage over baseline FD.

6 Conclusion

We introduce tentative object pruning, a method that overcomes the challenge of excessive branching factors resulting from large object counts by constructing candidate simplified tasks—by sampling reduced object sets—and searching through these tasks in parallel until a valid solution is found. Our approach substantially improves coverage for both grounded and lifted planners, solving 544 (33%) additional problems for Fast Downward and 380 (39%) additional problems for Powerlifted across expanded IPC and HTG Blocksworld benchmarks. These results demonstrate that object pruning effectively scales existing planners when problems contain many irrelevant or substitutable objects—a characteristic of automatically-generated domains—and that this benefit is not specific to any single planning paradigm. While our approach improves performance even in highly connected domains like Mineplanner, further improvements are possible through the development of better heuristics for object pruning in domains where objects can-

not be aggressively pruned without risking plan validity.

Code — https://github.com/AnitadeMelloKoch/object_pruning

References

Areces, C.; Bustos, F.; Dominguez, M.; and Hoffmann, J. 2014. Optimizing planning domains by automatic action schema splitting. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 24, 11–19.

Asai, M.; and Fukunaga, A. 2018. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *Proceedings of the aaai conference on artificial intelligence*, volume 32.

Bonet, B.; and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence*, 129(1-2): 5–33.

Büchner, C.; Ferber, P.; Seipp, J.; and Helmert, M. 2024. Abstraction heuristics for factored tasks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, 40–49.

Canonaco, G.; Pozanco, A.; and Borrajo, D. 2026. Semantic Partial Grounding via LLMs. *arXiv preprint arXiv:2602.22067*.

Corrêa, A. B.; and De Giacomo, G. 2024. Lifted planning: recent advances in planning using first-order representations. In *Proceedings of the 33rd International Joint Conference on Artificial Intelligence (IJCAI 2024)*, 8010–8019.

Corrêa, A. B.; Frances, G.; Hecher, M.; Longo, D. M.; and Seipp, J. 2023. The powerlifted planning system in the IPC 2023. *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

Corrêa, A. B.; Frances, G.; Pommerening, F.; and Helmert, M. 2021. Delete-relaxation heuristics for lifted classical planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, 94–102.

Corrêa, A. B.; and Seipp, J. 2022. Best-first width search for lifted classical planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, 11–15.

Fishman, M.; Kumar, N.; Allen, C.; Danas, N.; Littman, M.; Tellex, S.; and Konidaris, G. 2023. Task scoping: Generating task-specific simplifications of open-scope planning problems. In *PRL Workshop Series—Bridging the Gap Between AI Planning and Reinforcement Learning*.

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL—The Planning Domain Definition Language. Technical report, Technical Report.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: theory and practice*. Elsevier.

Gnad, D.; Torralba, A.; Domínguez, M.; Areces, C.; and Bustos, F. 2019. Learning how to ground a plan–partial grounding in classical planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 7602–7609.

Guan, L.; Valmeekam, K.; Sreedharan, S.; and Kambhampati, S. 2023. Leveraging pre-trained large language models to construct and utilize world models for model-based task planning. *Advances in Neural Information Processing Systems*, 36: 79081–79094.

Guo, H.; Wu, F.; Qin, Y.; Li, R.; Li, K.; and Li, K. 2023. Recent trends in task and motion planning for robotics: A survey. *ACM Computing Surveys*, 55(13s): 1–36.

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

Helmert, M.; and Domshlak, C. 2009. Landmarks, critical paths and abstractions: what's the difference anyway? In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 19, 162–169.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM (JACM)*, 61(3): 1–63.

Hill, W.; Liu, I.; De Mello Koch, A.; Harvey, D.; Kumar, N.; Konidaris, G.; and James, S. 2023. Mineplanner: A benchmark for long-horizon planning in large minecraft worlds. *arXiv preprint arXiv:2312.12891*.

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

Höller, D.; and Behnke, G. 2022. Encoding lifted classical planning in propositional logic. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, 134–144.

Jiang, Y.-q.; Zhang, S.-q.; Khandelwal, P.; and Stone, P. 2019. Task planning in robotics: an empirical comparison of pddl-and asp-based systems. *Frontiers of Information Technology & Electronic Engineering*, 20(3): 363–373.

Konidaris, G.; Kaelbling, L. P.; and Lozano-Perez, T. 2018. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61: 215–289.

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

Lang, T.; and Toussaint, M. 2009. Relevance grounding for planning in relational domains. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 736–751. Springer.

Lauer, P.; Torralba, A.; Fišer, D.; Höller, D.; Wichlacz, J.; and Hoffmann, J. 2021. Polynomial-time in PDDL input size: Making the delete relaxation feasible for lifted planning. In *ICAPS 2021 Workshop on Heuristics and Search for Domain-independent Planning*.

Mahdavi, S.; Aoki, R.; Tang, K.; and Cao, Y. 2024. Leveraging environment interaction for automated pddl translation and planning with large language models. *Advances in Neural Information Processing Systems*, 37: 38960–39008.

McDermott, D. 1999. Using regression-match graphs to control search in planning. *Artificial Intelligence*, 109(1-2): 111–159.

Nebel, B.; Dimopoulos, Y.; and Koehler, J. 1997. Ignoring irrelevant facts and operators in plan generation. In *European Conference on Planning*, 338–350. Springer.

Partalas, I.; Vrakas, D.; and Vlahavas, I. 2008. Reinforcement learning and automated planning: A survey. In *Artificial Intelligence for Advanced Problem Solving Techniques*, 148–165. IGI Global Scientific Publishing.

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

Ridder, B.; and Fox, M. 2014. Heuristic evaluation based on lifted relaxed planning graphs. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 24, 244–252.

Schwarting, W.; Alonso-Mora, J.; and Rus, D. 2018. Planning and decision-making for autonomous vehicles. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(1): 187–210.

Sievers, S. 2018. Merge-and-shrink heuristics for classical planning: Efficient implementation and partial abstractions. In *Proceedings of the International Symposium on Combinatorial Search*, volume 9, 90–98.

Ståhlberg, S. 2023. Lifted successor generation by maximum clique enumeration. In *ECAI 2023*, 2194–2201. IOS Press.

Wichlacz, J.; Höller, D.; and Hoffmann, J. 2022. Landmark Heuristics for Lifted Classical Planning. In *IJCAI*, 4665–4671.

A Acknowledgments

This work was supported by ONR REPRISM MURI N00014-24-1-2603 and ONR grant 00014-22-1-2592. This article solely reflects the opinions and conclusions of its authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the sponsors.