Learning Object Priorities for Partial Grounding

Mariam Musavi[1], **Johannes K. Fichte**[1], **Daniel Gnad**[2,1]

[1]Linköping University, Sweden
[2]Heidelberg University, Germany
mariam.musavi@liu.se, johannes.fichte@liu.se, daniel.gnad@uni-heidelberg.de

Abstract

Classical planning has seen a tremendous amount of research developing techniques that work with a grounded encoding. While lifted planning has recently seen increasing interest and many methods have been adapted to the lifted setting, scalability is still far behind grounded planners if the task at hand *can* be grounded. The middle ground we suggest is partial grounding, where we obtain an incomplete ground task from learning object priorities. Concretely, we instantiate actions based on a priority metric that we learn on a per-domain basis, grounding actions that are instantiated with objects that likely need to be modified by the corresponding action schema. We demonstrate how object priorities can be obtained, exemplify in several domains how they reflect action occurrence in plans, and evaluate how a partial-grounding algorithm performs when using the metric. In contrast to purely learning-based approaches, our solution comes with strong robustness guarantees. We employ learning only to obtain a *heuristic*, a heuristic that guides the grounding process. Our approach is sound, a plan found for the partially grounded model is always a plan for the full task. Incompleteness can be tackled by iteratively grounding more actions, which in the limit results in fully grounded planning.

Introduction

Most state-of-the-art planners assume that the lifted PDDL input has been compiled into a *grounded* propositional STRIPS or finite-domain representation (Bäckström and Nebel 1995; Helmert 2006, 2009). This ground encoding is the basis of most subsequent algorithms, from heuristic search (Bonet and Geffner 2001; Hoffmann and Nebel 2001; Richter and Westphal 2010) to compilation-based approaches like SAT-planning (Kautz and Selman 1992; Rintanen 2012). The flip side is that grounding itself can become the bottleneck: the number of ground actions is exponential in the schema arity, and even state-of-the-art reachability-based grounders fail on instances whose lifted description is comparatively compact. When grounding does not finish, a grounded planner cannot even start to look for a solution.

Two complementary lines of work tackle this bottleneck. The first aims to *ground better*: Corrêa et al. (2023) cast PDDL grounding as a Datalog evaluation problem, decompose the Datalog rules with tree decompositions, and solve them with an iterated answer-set-programming (ASP) procedure, grounding noticeably more tasks than Fast Downward and off-the-shelf ASP grounders. Their method incurs a runtime overhead, however, and still produces the *full* ground task, so it does not help on instances whose ground task is intrinsically huge. The second line abandons grounding altogether and plans directly in the lifted representation, including lifted successor generation (Corrêa et al. 2020), lifted adaptations of the FF heuristic (Corrêa et al. 2022), other delete-relaxation heuristics (Corrêa et al. 2021; Lauer et al. 2025), learned domain-independent heuristics (Chen, Thiébaux, and Trevizan 2024), and lifted SAT encodings (Höller and Behnke 2022). On tasks that *can* be grounded, however, lifted planners still lack behind their grounded counterparts by a wide margin.

The middle ground we pursue is *partial grounding* (Gnad et al. 2019): instantiate only a subset of the ground actions – ideally just enough to find a plan – and run the standard ground-planning machinery on the resulting task. The key question is *which* actions to instantiate. We answer it with an *action priority* function that estimates, for each candidate ground action, how likely it is to occur in some plan for the task at hand. This priority is plugged into a grounding loop that grows the partial task in a best-first manner guided by the action priorities, following Gnad et al. (2019).

Our contribution is a new, particularly lightweight way of obtaining such priorities. We decompose the estimation *per parameter*: for an action schema $a(p_1, \ldots, p_k)$ we ask, for each parameter slot independently, how likely a given object is to be used in that slot of a in a plan. The per-slot scores are collected from the object occurrences in training plans of small instances and then aggregated into a single per-action priority. The decomposition is similar in spirit to the *unary relaxation* of Lauer et al. (2021), who split each predicate $P(x_1, \ldots, x_n)$ into unary predicates $P_i(x_i)$ – one per slot – so that delete-relaxed heuristics become polynomial in the lifted PDDL input size. We split the priority of an action schema into per-parameter object scores, so that training and inference become polynomial in the number of objects rather than in the number of ground actions. A learning model therefore only has to predict a distribution over *objects*, not over the combinatorially many ground actions.

The idea of exploiting the per-parameter structure of action schemas to let a learned signal reason about partially instantiated actions is also explored by Wang and Trevizan (2025). They introduce *partial-space search*, a refinement of

state-space search in which action schemas are instantiated one parameter at a time, and guide it with learned *action set heuristics* that score sets of (partially instantiated) actions. The objective is orthogonal to ours: they reshape the *search space* of an otherwise lifted planner to lower its effective branching factor, whereas we use per-parameter object scores to decide *which ground actions to instantiate* for a downstream grounded planner. Both works share the observation that decomposing action instantiation parameter by parameter yields a more granular and learnable interface than reasoning over whole ground actions.

The appeal of this design is not just efficiency but *reliability*. Machine learning enters the pipeline in a bounded role – a heuristic over grounding decisions – while the rest of the planner remains a standard symbolic algorithm. The approach is *sound* by construction: every plan for the partial task is a valid plan for the lifted task, because we only ever remove operators, never add or alter them. The only guarantee at risk is *completeness*: a misjudged priority can yield an unsolvable partial task. This failure mode is benign, however, since it is recoverable by iteratively grounding more actions, which in the limit returns to the full grounding. Compared to end-to-end learned planners, failures can be diagnosed symbolically – which operator was missing, on which schema and at which parameter slot – and the priority signal can be retrained or overridden in a targeted way.

Our empirical study is preliminary: we do not yet *learn* the per-parameter priorities, but extract ground-truth object occurrences from optimal plans of small training instances, compare three per-slot aggregations, and feed the result into the grounding loop. With this oracle signal, object-level priorities shrink the grounded task substantially while keeping it solvable – the empirical premise on which future learned instantiations will stand or fall. We discuss candidate learners, from symbolic task analyses to GNN- and LLM-based predictors, as future work.

Background

We introduce lifted and grounded classical planning and the grounding step that connects them.

Classical Planning

A *lifted planning task* is a pair $\Pi = \langle \mathcal{D}, \mathcal{I} \rangle$ consisting of a *domain* $\mathcal{D}$ and a problem *instance* $\mathcal{I}$ (Corrêa et al. 2020). The domain $\mathcal{D} = \langle \mathcal{T}, \mathcal{P}, \mathcal{A} \rangle$ consists of a set of *types* $\mathcal{T}$, a set of *predicate symbols* $\mathcal{P}$, and a set of *action schemas* $\mathcal{A}$. Each predicate $p \in \mathcal{P}$ has a fixed arity and typed argument positions. For a predicate p of arity n and a tuple $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ of typed variables matching p's argument types, $p(\mathbf{x})$ is an *atom*; it is *ground* if $\mathbf{x}$ contains no variables.

An action schema $a \in \mathcal{A}$ is a tuple $\langle \mathit{params}(a), \mathit{pre}^+(a), \mathit{pre}^-(a), \mathit{add}(a), \mathit{del}(a) \rangle$, where $\mathit{params}(a)$ is a list of typed variables called the *parameters*; $\mathit{pre}^+(a)$ and $\mathit{pre}^-(a)$ are sets of atoms over $\mathit{params}(a)$ defining the positive and negative *preconditions*; $\mathit{add}(a)$ and $\mathit{del}(a)$ are sets of atoms over $\mathit{params}(a)$ defining the *add* and *delete effects*.

A problem instance $\mathcal{I} = \langle \mathcal{O}, I, G \rangle$ provides the *objects* $\mathcal{O}$ (a set of typed constants), the *initial state* I (a set of ground atoms), and the *goal* G (a set of ground atoms).

A *ground action* $\hat{a}$ is obtained from a schema $a \in \mathcal{A}$ by substituting each parameter in $\mathit{params}(a)$ with a type-compatible object from $\mathcal{O}$. For a parameter $p_i \in \mathit{params}(a)$, we write $\hat{a}[p_i]$ for the object that $\hat{a}$ substitutes for p_i. $\hat{a}$ is *applicable* in a state s (a set of ground atoms) if $\mathit{pre}^+(\hat{a}) \subseteq s$ and $\mathit{pre}^-(\hat{a}) \cap s = \emptyset$; applying it yields the successor state $s' = (s \setminus \mathit{del}(\hat{a})) \cup \mathit{add}(\hat{a})$. A *plan* for Π is a sequence of ground actions $\pi = \langle \hat{a}_1, \ldots, \hat{a}_n \rangle$ such that each $\hat{a}_i$ is applicable after applying $\hat{a}_1, \ldots, \hat{a}_{i-1}$ to I, and the resulting state s_n satisfies $G \subseteq s_n$. Assuming a unit action-cost setting, a plan is *optimal* if no plan has shorter length.

A *STRIPS task* $\Pi_g = \langle F, O, I, G \rangle$ is a propositional planning task consisting of a set of *atoms* F, a set of *operators* O, an *initial state* $I \subseteq F$, and a *goal* $G \subseteq F$. Each operator $o \in O$ is defined by sets $\mathit{pre}(o), \mathit{add}(o), \mathit{del}(o) \subseteq F$ of *preconditions*, *add effects*, and *delete effects*. o is *applicable* in state $s \subseteq F$ if $\mathit{pre}(o) \subseteq s$; applying o yields the successor state $s' = (s \setminus \mathit{del}(o)) \cup \mathit{add}(o)$. A *plan* for Π_g is a sequence of operators $\pi = \langle o_1, \ldots, o_n \rangle$ iteratively applicable from I, reaching a state $s_n \supseteq G$. The plan is *optimal* if its length is minimal among all plans.

We define the *delete-relaxation* Π_g^+ of Π_g as the task obtained by setting $\mathit{del}(o) = \emptyset$ for all $o \in O$, yielding the set of relaxed operators O^+. An atom p is *delete-relaxed reachable* from I if the task $\Pi_g^{+,p} = \langle F, O^+, I, \{p\} \rangle$ is solvable.

Grounding

Given a lifted task $\Pi = \langle \mathcal{D}, \mathcal{I} \rangle$, the corresponding STRIPS task $\Pi_g = \langle F, O, I, G \rangle$ is obtained by *grounding*: F contains one atom for each type-consistent assignment of objects from $\mathcal{O}$ to the argument positions of a predicate in $\mathcal{P}$; O contains one operator for each type-consistent assignment of objects to the parameters of each schema in $\mathcal{A}$; and I, G are inherited directly from $\mathcal{I}$. Enumerating all type-consistent assignments is infeasible for large instances. In practice, only atoms and operators that are *delete-relaxed reachable* from I are instantiated (Hoffmann and Nebel 2001; Helmert 2009). This is computed by a forward reachability analysis over the delete-relaxed task, often encoded as a Datalog program. The resulting grounded task is equivalent to the original lifted task in the sense that every plan for Π corresponds to a plan for Π_g and vice versa.

Partial Grounding

We base our method on the grounding algorithm of Fast Downward. The algorithm initializes a queue with the atoms in the initial state and in each iteration pops one element. If the element is an atom, all operators whose preconditions have already been processed are added to the queue. If the element is an operator, all its add effects are pushed to the queue. The algorithm terminates when the queue is empty, at which point all processed atoms and operators are delete-relaxed reachable from I.

Algorithm 1 differs from Helmert (2009) in two ways: (1) it can stop before the queue is empty, and (2) operators are instantiated in a particular order. Both choice points aim at minimizing the size of the partially grounded task while keeping it solvable. Our focus in this paper is the operator

Algorithm 1: Partial Grounding.

Input: A lifted task $\Pi = \langle \mathcal{D}, \mathcal{I} \rangle$
Output: A STRIPS task $\Pi_g = (F, O, I, G)$

$q \leftarrow \mathrm{I}$;
$F \leftarrow \emptyset$; // Processed atoms
$O \leftarrow \emptyset$; // Processed operators
while $\neg$(*q*.empty() $\vee$ $G \subseteq F$) $\wedge$ $\neg$StoppingCondition **do**
  if *q*.containsAtom() **then**
    $f \leftarrow q$.popAtom() ;
    $F \leftarrow F \cup \{f\}$;
    for $o \notin O \wedge pre(o) \subseteq F$ **do**
      q.insert(o) ;
  else
    $o \leftarrow q$.popHighPriorityOperator() ;
    $O \leftarrow O \cup \{o\}$;
    for $f \notin F \wedge f \in add(o)$ **do**
      q.insert(f) ;
return (F, O, I, G);

ordering; for the stopping condition we use the simple criterion described below.

Round-robin grounding. The straightforward implementation of the operator queue q is a single priority queue ordered by action priority. In practice, such a single queue tends to be dominated by whichever schema currently holds the highest-scoring candidates, so that other schemas are barely instantiated. We therefore use a *round-robin* scheme: q is maintained as one priority queue *per action schema* $a \in \mathcal{A}$, each ordered by the action priority $\rho(\hat{a})$. popHighPriorityOperator() cycles through the per-schema queues in a fixed order and pops the top-scoring operator of the next non-empty queue. Every schema thus receives grounding budget proportional to the number of rounds rather than to the absolute magnitude of its priorities, which balances the number of grounded operators across schemas.

Grounding beyond goal-reachability. The condition $G \subseteq F$ in the while-loop lets the algorithm terminate as soon as the goal is delete-relaxed reachable. While sufficient for delete-relaxed solvability, the resulting model is typically lean and brittle: a single missing operator on the non-relaxed plan can render the partial task unsolvable, even though the lifted task is solvable. To improve robustness, we continue the fix-point past $G \subseteq F$ and ground an additional $x\%$ of operators in the same priority order. With $x = 0$ we recover the minimal goal-reachable model; larger values trade model size for the likelihood that a real plan exists in the partially grounded task, and in the limit recover the full grounding.

Grounding with Object Priorities

This section describes how we obtain object priorities from training data and aggregate them into the action priority $\rho(\hat{a})$ used by the partial-grounding algorithm.

Obtaining Object Priorities

We assume access to a set of small *training instances* $\mathcal{I}_1, \ldots, \mathcal{I}_m$ over the same domain $\mathcal{D}$. For each $\mathcal{I}_j$, we solve $\Pi_j = \langle \mathcal{D}, \mathcal{I}_j \rangle$ optimally and collect the set U_j of ground actions that participate in *at least one* optimal plan for Π_j. This set is computed by BDD-based symbolic search (Torralba et al. 2017), as in the partial-grounding pipeline of Gnad et al. (2019). The sets U_j are the only signal we extract from the training data: an action $\hat{a} \in U_j$ is *useful*; all other ground actions in $O_j \setminus U_j$ are not.

Per-parameter object priorities. From U_j, we extract object priorities on a per-schema, per-parameter basis. Let $a \in \mathcal{A}$ be an action schema with parameters $params(a) = \langle p_1, \ldots, p_k \rangle$, let $U_j^a \subseteq O_j$ denote the useful ground actions of a of instance $\mathcal{I}_j$, and let O_j^a denote the set of all ground actions of a. The priority of an object o at parameter position p_i of schema a is

$$\rho_a(o, p_i) \;=\; \frac{\left|\{\, \hat{a} \in U_j^a \mid \hat{a}[p_i] = o \,\}\right|}{|O_j^a|}\,, \qquad (1)$$

the fraction of ground actions of a that are useful and have o at parameter position p_i. By construction $\rho_a(o, p_i) \in [0, 1]$. In the extreme case in which every ground action of a with o at position p_i is useful, we have $\rho_a(o, p_i) = 1$. Objects that never appear at p_i in a in an optimal plan receive priority 0.

From Object Priorities to Action Priorities

Algorithm 1 requires a single priority score per candidate ground action. Given $\hat{a}$ with schema $a \in \mathcal{A}$ and substitution $o_i = \hat{a}[p_i]$, we aggregate the k per-parameter priorities $\rho_a(o_i, p_i)$ into one action priority $\rho(\hat{a})$ in one of three ways.

Sum. The direct aggregation is additive,

$$\rho^{\Sigma}(\hat{a}) \;=\; \sum_{i=1}^{k} \rho_a(o_i, p_i)\,. \qquad (2)$$

A ground action whose objects all have high priority receives a high score; a single low or zero priority only lowers the score additively and can be outweighed by strong evidence at other positions.

Product. A multiplicative aggregation reflects a joint-probability interpretation in which the parameter slots are filled independently:

$$\rho^{\Pi}(\hat{a}) \;=\; \prod_{i=1}^{k} \max\big(\rho_a(o_i, p_i),\, \varepsilon\big)\,. \qquad (3)$$

The lower bound $\varepsilon > 0$ is essential: a single $\rho_a(o_i, p_i) = 0$ would otherwise zero out the whole product and discard the action entirely. Since the training data is finite and may simply miss legitimate (o_i, p_i) combinations, treating any zero as hard evidence of irrelevance is too aggressive. Clipping at ε keeps such actions viable but strongly down-weighted.

Optimistic. The third variant is deliberately permissive: it only asks whether o_i has *ever* appeared at p_i in a useful action. We binarize the object priorities and sum,

$$\rho^{\mathrm{opt}}(\hat{a}) \;=\; \sum_{i=1}^{k} \mathbf{1}[\rho_a(o_i, p_i) > 0] \;, \tag{4}$$

where $\mathbf{1}[\cdot]$ is the indicator function. The optimistic score is more robust to the limited coverage of small training instances than sum and product, at the price of ignoring how strongly the data supports each (o_i, p_i) combination.

Experiments

We empirically evaluate whether object priorities produce partially grounded tasks that are both small and solvable in practice. Our implementation extends the partial-grounding system of Gnad et al. (2019) in the translator component of Fast Downward (Helmert 2006), using our action priorities as the keys of the per-schema priority queues.

Setup. We run the LAMA-first configuration (Richter and Westphal 2010) as the search component for every configuration, so that any differences are attributable to the grounding step alone. Our preliminary evaluation uses a subset of the domains from the learning track of the International Planning Competition 2023 (Taitler et al. 2024). We use runtime and memory limits of 10 minutes and 4 GiB, respectively. We focus on the reduction in task size, measured as the number of actions in the partially grounded task. Coverage (the number of instances solved within the resource limits) is reported alongside, since a smaller task is only useful if a plan is still found.

For every instance $\mathcal{I}_j$ in our benchmark set, we obtain the useful action set U_j via the BDD-based symbolic search of Torralba et al. (2017). Object priorities $\rho_a(o, p_i)$ are computed according to Eq. 1 and aggregated into action priorities $\rho(\hat{a})$ using the sum, product, and optimistic binary variants. The product variant uses $\varepsilon = 10^{-4}$. All partial-grounding configurations use the round-robin queue scheme described in the background and continue grounding for an additional 10% of operators after the goal becomes delete-relaxed reachable. We compare standard Fast Downward grounding (*Full-Grounding*) against partial grounding with four priority sources: uniformly random scores (*Random*) as a baseline, and the three aggregations introduced above (*Sum*, *Product*, *Binary*). The random configuration isolates the effect of the partial-grounding mechanism itself, separating it from the contribution of the priority signal.

Results. Figure 1 shows our results across instances of the blocksworld, caldera, and satellite domains, comparing full grounding against partial grounding guided by aggregated object priorities. The partially grounded tasks are substantially smaller in all three domains while mostly preserving solvability. In several instances, the reduction exceeds a factor of 10, which, given the absolute size of the instances, is remarkable. For larger instances we expect the reduction to be even more pronounced, as the number of ground actions typically grows much faster with instance size than the number of actions required to find a plan. A second observation is that the *Random* configuration failed to solve any instance, which is why we omit it from the plots. This confirms that the object priorities themselves carry the relevant signal, and that the reduction in task size cannot be attributed to the partial-grounding mechanism alone. Missing bars in some plots indicate that an instance could not be solved, which is due to an insufficient grounding that misses essential operators. We plan to address this by assessing the robustness of the partially grounded task during grounding, moving from the simple 10% increment to a more informed mechanism for deciding when to stop instantiating more actions, and – in the longer term – by triggering targeted re-grounding when the search runs into a dead end.

Conclusions and Future Work

We present a partial-grounding approach in which the operator order inside Fast Downward's translator component is guided by per-parameter *object priorities*, aggregated into action-level scores. Priorities are derived from the set of useful ground actions appearing in any optimal plan on small training instances, extracted with BDD-based symbolic search (Torralba et al. 2017). Combined with a round-robin queue over action schemas and a small post-goal grounding buffer, the partial grounding is sound by construction – any plan for the partial task is a plan for the lifted task – and incompleteness can be recovered by iteratively grounding more actions.

The per-parameter priorities implicitly relax the joint structure of useful ground actions: by counting how often an object o occupies a single parameter slot p_i, we treat the occurrences of objects at different positions as independent and ignore correlations between them, similar to the unary relaxation by Lauer et al. (2021). This relaxation generalizes naturally to tuples of objects – recording the frequency of pairs (o, o') at parameter positions (p_i, p_j), or higher-order combinations – which yields a strictly more informative signal, at the cost of estimating up to $|\mathcal{O}|^k$ priorities per schema.

The most immediate direction for future work is to replace our ground-truth statistics with a learned or analytically derived signal, broadly along four lines.

Sub-symbolic learning. Object priorities are a natural target for neural and kernel-based learners. The Weisfeiler–Leman (WL) approach of Chen, Trevizan, and Thiébaux (2024) shows that classical ML on WL features yields reliable, domain-independent planning heuristics; the same features can be evaluated per object, producing object priorities once the embeddings have been computed. On the GNN side, the line of work by Ståhlberg, Bonet, and Geffner (2022, 2023) on general policies already produces per-object scores during message passing, which can be repurposed as priorities. Recent LLM-based heuristic generation (Corrêa, Pereira, and Seipp 2025) suggests a third sub-symbolic option, in which an LLM emits Python-level priority functions over PDDL objects.

Symbolic learning. A complementary path is to obtain interpretable, symbolic predictors of object priority. The

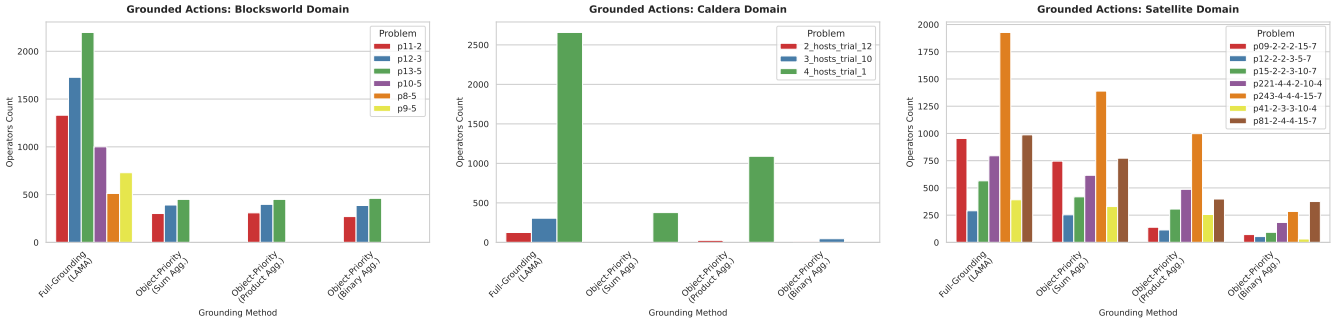

Figure 1: Reduction in task size, measured as the number of ground actions, across three domains. We omit the bars for instances not solved by a configuration within the limits, which indicates insufficient grounding.

description-logic feature language and general policies of Bonet, Francès, and Geffner (2019); Francès, Bonet, and Geffner (2021), and the sketch-based decompositions of Drexler, Seipp, and Geffner (2021, 2024), provide compact inspectable rules that classify which objects are relevant for which action schemas in a given state. The transition-classifier framework of Musayev et al. (2025) is also closely related to our setting: it learns per-transition (and hence per-ground-action) labels that distinguish useful from useless steps, which is essentially the signal we extract from optimal plans, but produced in a learned, lifted fashion rather than by BDD enumeration.

Lifted heuristics. Object priorities can also be derived from lifted relaxation heuristics, without any learning. The lifted unary relaxation (Lauer et al. 2021), the lifted delete relaxation (Corrêa et al. 2021), and the lifted FF heuristic (Corrêa et al. 2022) can be used to assign relevance estimates to parameter-object combinations as a by-product of their reachability. Exploiting these intermediate quantities – e.g. the objects participating in some lifted landmark (Wichlacz et al. 2023) or in some unary-relaxed plan – yields a principled, sound source of object priorities that requires no training data at all.

Symbolic task analysis. Lifted analyses of the task structure – lifted mutex groups and operator mutexes (Fišer, Torralba, and Shleyfman 2019; Fišer 2020), or the lifted causal graph – expose which objects can possibly occupy which parameter positions on any path to the goal. Such analyses provide upper bounds on object relevance that can serve either as priorities directly or as a pruning layer that zeroes out provably irrelevant (o, p_i) combinations before any aggregation.

Training data and supervision. A second direction concerns the training pipeline itself. Our setup collects all ground actions appearing in any optimal plan, which requires solving each training instance optimally and enumerating its optimal plan space. A weaker alternative is to use a single, possibly satisficing, plan per instance, trading priority quality against training cost; the optimistic aggregation should be largely robust to this change, since it only depends on which (o, p_i) combinations occur at all. Conversely, richer supervision – weighting plans by quality, or using transition classifiers (Musayev et al. 2025) to label non-optimal plans as well – could give finer-grained priorities than the relative-frequency estimator.

Tighter integration with search. A more speculative direction is to couple grounding with search. Our current pipeline grounds first and plans second; object priorities also provide a natural signal for iterative grounding, where unsolvability or excessive search effort on the partial task triggers a targeted expansion of the grounded action set.

References

Bäckström, C.; and Nebel, B. 1995. Complexity Results for SAS$^+$ Planning. *Computational Intelligence*, 11(4): 625–655.

Bonet, B.; Francès, G.; and Geffner, H. 2019. Learning Features and Abstract Actions for Computing Generalized Plans. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI 2019)*, 2703–2710. AAAI Press.

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

Chen, D. Z.; Thiébaux, S.; and Trevizan, F. 2024. Learning Domain-Independent Heuristics for Grounded and Lifted Planning. In Dy, J.; and Natarajan, S., eds., *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2024)*, 20078–20086. AAAI Press.

Chen, D. Z.; Trevizan, F.; and Thiébaux, S. 2024. Return to Tradition: Learning Reliable Heuristics with Classical Machine Learning. In Bernardini, S.; and Muise, C., eds., *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*, 68–76. AAAI Press.

Corrêa, A. B.; Francès, G.; Pommerening, F.; and Helmert, M. 2021. Delete-Relaxation Heuristics for Lifted Classical Planning. In Goldman, R. P.; Biundo, S.; and Katz, M., eds., *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)*, 94–102. AAAI Press.

Corrêa, A. B.; Hecher, M.; Helmert, M.; Longo, D. M.; Pommerening, F.; and Woltran, S. 2023. Grounding Plan-

ning Tasks Using Tree Decompositions and Iterated Solving. In Koenig, S.; Stern, R.; and Vallati, M., eds., *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS 2023)*. AAAI Press.

Corrêa, A. B.; Pereira, A. G.; and Seipp, J. 2025. Classical Planning with LLM-Generated Heuristics: Challenging the State of the Art with Python Code. In *Proceedings of the Thirty-Ninth Annual Conference on Neural Information Processing Systems (NeurIPS 2025)*.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Francès, G. 2020. Lifted Successor Generation using Query Optimization Techniques. In Beck, J. C.; Karpas, E.; and Sohrabi, S., eds., *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020)*, 80–89. AAAI Press.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Francès, G. 2022. The FF Heuristic for Lifted Classical Planning. In Honavar, V.; and Spaan, M., eds., *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI 2022)*, 9716–9723. AAAI Press.

Drexler, D.; Seipp, J.; and Geffner, H. 2021. Expressing and Exploiting the Common Subgoal Structure of Classical Planning Domains Using Sketches. In Erdem, E.; Bienvenu, M.; and Lakemeyer, G., eds., *Proceedings of the Eighteenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2021)*, 258–268. IJCAI Organization.

Drexler, D.; Seipp, J.; and Geffner, H. 2024. Expressing and Exploiting Subgoal Structure in Classical Planning Using Sketches. *Journal of Artificial Intelligence Research*, 80: 171–208.

Fišer, D. 2020. Lifted Fact-Alternating Mutex Groups and Pruned Grounding of Classical Planning Problems. In Conitzer, V.; and Sha, F., eds., *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*, 9835–9842. AAAI Press.

Fišer, D.; Torralba, Á.; and Shleyfman, A. 2019. Operator Mutexes and Symmetries for Simplifying Planning Tasks. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI 2019)*, 7586–7593. AAAI Press.

Francès, G.; Bonet, B.; and Geffner, H. 2021. Learning General Planning Policies from Small Examples Without Supervision. In Leyton-Brown, K.; and Mausam, eds., *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021)*, 11801–11808. AAAI Press.

Gnad, D.; Torralba, Á.; Domínguez, M. A.; Areces, C.; and Bustos, F. 2019. Learning How to Ground a Plan – Partial Grounding in Classical Planning. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI 2019)*, 7602–7609. AAAI Press.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26: 191–246.

Helmert, M. 2009. Concise Finite-Domain Representations for PDDL Planning Tasks. *Artificial Intelligence*, 173: 503–535.

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 Thiébaux, S.; and Yeoh, W., eds., *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)*, 134–144. AAAI Press.

Kautz, H.; and Selman, B. 1992. Planning as Satisfiability. In Neumann, B., ed., *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI 1992)*, 359–363. John Wiley and Sons.

Lauer, P.; Torralba, Á.; 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 Zhou, Z.-H., ed., *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI 2021)*, 4119–4126. IJCAI.

Lauer, P.; Torralba, Á.; Höller, D.; and Hoffmann, J. 2025. Continuing the Quest for Polynomial Time Heuristics in PDDL Input Size: Tractable Cases for Lifted h^{add}. In Lipovetzky, N.; Sardina, S.; Harabor, D.; and Ramirez, M., eds., *Proceedings of the Thirty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2025)*. AAAI Press.

Musayev, F.; Drexler, D.; Gnad, D.; and Seipp, J. 2025. Combining Heuristics and Transition Classifiers in Classical Planning. In Lynce, I.; and Murano, A., eds., *Proceedings of the 28th European Conference on Artificial Intelligence (ECAI 2025)*, 4694–4701. IOS Press.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *Journal of Artificial Intelligence Research*, 39: 127–177.

Rintanen, J. 2012. Planning as Satisfiability: Heuristics. *Artificial Intelligence*, 193: 45–86.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022. Learning Generalized Policies without Supervision Using GNNs. In Kern-Isberner, G.; Lakemeyer, G.; and Meyer, T., eds., *Proceedings of the Nineteenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2022)*, 474–483. IJCAI Organization.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2023. Learning General Policies with Policy Gradient Methods. In Marquis, P.; Son, T. C.; and Kern-Isberner, G., eds., *Proceedings of the Twentieth International Conference on Principles of Knowledge Representation and Reasoning (KR 2023)*, 647–657. IJCAI Organization.

Taitler, A.; Alford, R.; Espasa, J.; Behnke, G.; Fišer, D.; Gimelfarb, M.; Pommerening, F.; Sanner, S.; Scala, E.; Schreiber, D.; Segovia-Aguas, J.; and Seipp, J. 2024. The 2023 International Planning Competition. *AI Magazine*, 45(2): 280–296.

Torralba, Á.; Alcázar, V.; Kissmann, P.; and Edelkamp, S. 2017. Efficient Symbolic Search for Cost-optimal Planning. *Artificial Intelligence*, 242: 52–79.

Wang, R. X.; and Trevizan, F. 2025. Leveraging Action Relational Structures for Integrated Learning and Planning. In Lipovetzky, N.; Sardina, S.; Harabor, D.; and Ramirez, M., eds., *Proceedings of the Thirty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2025)*, 269–278. AAAI Press.

Wichlacz, J.; Höller, D.; Fišer, D.; and Hoffmann, J. 2023. A Landmark-Cut Heuristic for Lifted Optimal Planning. In Gal, K.; Nowé, A.; Nalepa, G. J.; Fairstein, R.; and Rădulescu, R., eds., *Proceedings of the 26th European Conference on Artificial Intelligence (ECAI 2023)*, 2623–2630. IOS Press.