LEARNING TO DECOMPOSE AND ACT: LANDMARK-BASED MANAGER–WORKER POLICIES FOR GENERALIZED PLANNING AT SCALE

Anonymous authors
Paper under double-blind review

ABSTRACT

General policies learn from small instances to solve an entire instance class of a classical planning domain. Yet, as instances grow, so do the plans required to solve them, as well as the cost of evaluating alternatives at each decision. We introduce Landmarker, a manager–worker hierarchy that decomposes tasks into atomic subproblems: the manager generates one atom from a landmark vocabulary as the next subgoal, and the worker generates the primitive actions that solve it. The two policies are trained independently of each other with the help of polynomial width-based search: the manager via reinforcement learning, for which the search achieves its subgoals, and the worker from the plans that the search finds. At deployment, the learned worker replaces the search. We evaluate Landmarker on the IPC 2023 learning-track benchmark against state-of-the-art learned heuristics, learned policies, and LAMA. In the experiments, Landmarker solves 648 of the 900 test instances, more than every baseline except the strongest lookahead policy (668). Earlier landmark-based decompositions selected subgoals through search or heuristic rules. Landmarker makes landmarks viable as subgoals again by learning how to order and achieve them, solving more instances than LAMA, which uses landmarks only as a heuristic.

1 INTRODUCTION

Generalized planning is concerned with learning policies that solve an entire instance class of a given domain, often scaling to significantly larger instances than seen during training. Domains provide the predicate and action schema sets shared among all instances, while the instances themselves differ in their objects, initial states, and goals. Existing approaches learn symbolic policies expressed through logical features (Francès et al., 2021), or neural policies and value functions over relational representations (Toyer et al., 2020; Ståhlberg et al., 2022a;b). In the latter case, the representation determines which states and decisions the network can distinguish, and hence which general policies it can express (Ståhlberg et al., 2025; Horčík & Šír, 2024; Drexler et al., 2024).

GOOSE and WL-GOOSE learn heuristics to guide a planner (Chen et al., 2024a;b), while Aichmüller et al. (2026a) use search both to find plans and to improve the value function guiding it. Neural sketch and lookahead policies select a subgoal state after search has generated the alternatives (Aichmüller & Geffner, 2025; Aichmüller et al., 2026b). We reverse this order to a guided form of decomposition: the manager first selects an atomic goal, which the worker attempts to achieve.

Two problems arise when policies learned on small instances are applied to larger ones. First, larger instances ask for a larger conjunctive set of goals that require longer plans, so the policy must handle combinations of requirements that differ greatly from ones seen in training. Second, selecting each action or subgoal becomes more expensive when the alternatives are evaluated enumeratively. A predicate or action schema with r arguments has up to n^r ground instances over n objects. In the initial state of the largest childsnack test instance, 47 million actions are applicable. Decomposing a long task into subproblems does not by itself remove the cost of these choices.

We address the first problem with Landmarker, a manager–worker hierarchy whose subgoals are drawn from landmarks: facts, or alternatives among facts, that every solution plan must satisfy at some point (Hoffmann et al., 2004). The manager maps the current state s and task goal G to one

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

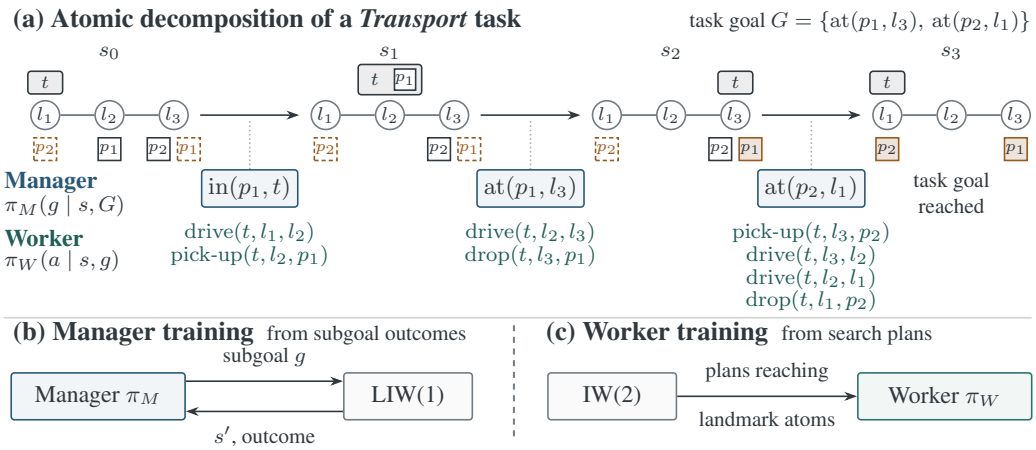

Figure 1: Landmarker trains a manager policy $\pi_M(g \mid s, G)$ that selects subgoals g, and a worker policy $\pi_W(a \mid s, g)$ that selects the primitive actions to achieve them. The two policies are trained independently of each other with the help of polynomial width-based search. (a) A transport task solved with three subgoals, each selected from the state the previous one reached. The first, $\text{in}(p_1, t)$, is a landmark absent from the task goal. Dashed boxes mark goal locations, and capacity arguments are omitted. (b) The manager is trained by reinforcement learning while LIW(1), a width-based search restricted to atom pairs containing a landmark, achieves each subgoal and returns the reached state and the outcome. At deployment, the learned worker replaces LIW(1). (c) The worker learns from IW(2) plans that reach landmark atoms.

atom g from this vocabulary. The atom defines the classical planning subproblem $P[s, g]$, with the original problem's actions, but initial state s, and goal $\{g\}$ instead. The worker applies primitive actions until g holds or the attempt fails, after which the manager selects another atom from the state reached (Figure 1). The manager thus learns a state-dependent serialization into single-atom subproblems (Lipovetzky & Geffner, 2012; Bonet & Geffner, 2024), and the worker learns a general policy for solving them. Landmarks were introduced as intermediate goals for a base planner, which pursued them in the order of the landmark graph (Porteous et al., 2001; Hoffmann et al., 2004). This loop performed worse than search guided by a landmark heuristic, and landmarks have served as heuristic estimates since (Richter & Westphal, 2010). Here we learn which landmark to pursue next in the current state and how to achieve it. Hierarchical reinforcement learning also learns both levels, but usually trains and tests in one environment (Nachum et al., 2018; Levy et al., 2019; Kim et al., 2021). Our policies must transfer to instances with many more objects, goal atoms and candidate subgoals than seen in training.

The worker only needs to learn how to achieve atomic landmark goals drawn from the domain's fixed predicate vocabulary. The same goal forms appear at train and test time, even when their objects and surrounding states change; relational representations enable the transfer. In contrast, a regular task goal can combine many atoms in ways specific to the instance. The worker can reuse its training across tasks with different goal conjunctions, because the manager resolves them.

To address the second problem, we represent both policies with Relational Graph Neural Networks (R-GNNs) that *generate* their decisions one output at a time: a predicate or action schema, followed by its object arguments. Each partial choice is added to the graph as typed relations and auxiliary nodes, so that message passing can condition the next choice on prior ones. The added relations also make both policies more expressive, since the network can then distinguish objects that were indistinguishable before.

We train the two policies *independently* of each other with polynomial width-based search (Lipovetzky & Geffner, 2012). The manager is trained by reinforcement learning while a width-based search achieves its subgoals, and the worker learns from plans that such a search finds. At deployment, the learned worker replaces the search, and we evaluate this replacement by running the same managers with both. Our contributions are the following.

- A two-level hierarchical policy for generalized planning. The manager learns a state-dependent choice of the next landmark atom over a vocabulary that grows with the instance, and the worker learns to achieve it. Both levels generate their subgoal or action autoregressively as a predicate or action schema followed by its arguments, following Janisch et al. (2023).
- A way to train the manager and the worker independently of each other, with polynomial width-based search as the manager's executor and as the source of the worker's plans.
- Experiments on the IPC 2023 learning track, whose test instances scale far beyond the training instances. Landmarker surpasses most baselines, and with the same managers the learned worker scales to hard instances where width-based search runs out of time.

2 Related Work

Hierarchical reinforcement learning. Hierarchical reinforcement learning decomposes tasks into subtasks that lower-level policies solve. Some methods take the hierarchy as input: MAXQ decomposes the value function along a given task graph (Dietterich, 2000), and hierarchies of abstract machines restrict policies through given partial programs (Parr & Russell, 1998). Options formalize temporally extended actions through initiation sets, policies and termination conditions (Sutton et al., 1999). Their policies can be learned; the option-critic architecture jointly learns option policies, termination conditions and a policy over options (Bacon et al., 2017).

In goal-based hierarchies, managers set goals for the level below (Dayan & Hinton, 1993). h-DQN learns both levels over designer-specified subgoals (Kulkarni et al., 2016), whereas FeUdal Networks learn goals expressed as directions in a latent space (Vezhnevets et al., 2017). HIRO uses two levels with subgoals expressed as relative changes in observations (Nachum et al., 2018); HAC learns multiple levels with target states as subgoals (Levy et al., 2019). Our setting requires subgoal and action vocabularies that vary with the objects in each planning instance, and Landmarker fixes the hierarchy to two levels over landmark-derived atoms.

Landmarks and subgoals. Landmarks have different meanings in reinforcement learning and classical planning. HIGL selects visited states for coverage and novelty, then uses shortest paths over them to guide subgoal generation (Kim et al., 2021). Such landmarks need not occur on every solution. Planning landmarks are atoms, or disjunctions of atoms, that hold at some point along *every solution trajectory* (Section 3). Porteous et al. (2001) and Hoffmann et al. (2004) introduced them as intermediate goals for decomposing planning tasks. Their control loop passes the available landmark-graph leaves as a disjunctive goal to a base planner, which determines which landmark is reached next. Richter & Westphal (2010) found this loop to perform worse than search with a landmark-counting heuristic, with longer plans and more dead ends, and landmarks have guided search through such heuristics since, as in LAMA (Richter et al., 2008; Richter & Westphal, 2010). Subgoal decomposition also appears in search. SIW repeatedly calls IW to consistently achieve one additional goal atom (Lipovetzky & Geffner, 2012), PROBE greedily pursues the nearest unachieved landmark (Lipovetzky & Geffner, 2011), and the goal agenda orders cumulative goal subsets before planning (Koehler & Hoffmann, 2000). In these methods, search or predefined rules choose the next subproblem.

Learning to select symbolic subgoals also appears in Núñez-Molina et al. (2022). They generate candidates from a supplied domain-specific pattern, including intermediate goals such as collecting a particular gem, and learn which one to pass to a classical planner. We derive candidates by landmark extraction without imposing landmark orderings, and the learned worker replaces the base planner at deployment.

Sketches specify subgoals through qualitative changes in logical features; termination and bounded width provide conditions for efficient solution by repeated width-based search (Lipovetzky & Geffner, 2012; Drexler et al., 2022; Bonet & Geffner, 2024). Neural sketch decomposition selects a state already reached by lookahead (Aichmüller & Geffner, 2025), and other approaches generate future states (Czechowski et al., 2021) or regress goal conjunctions (Xu et al., 2019). An atomic goal specifies only a condition to achieve: the manager chooses that condition before execution, while the executor determines which state satisfying it is reached.

Relational decisions. Policies typically either score separately encoded state transitions (Ståhlberg et al., 2023) or jointly encoded ones (Aichmüller et al., 2026b; Ståhlberg & Geffner, 2026). Generative policies construct an output instead. GABAR selects a schema and its arguments with a GRU over one graph encoding, whose construction also requires the applicable-action set (Mangannavar et al., 2025). Its graph embeddings remain fixed during argument selection. Janisch et al. (2023) instead mark previously selected arguments in the graph and rerun message passing before choosing later arguments, as graph-generation methods do (Li et al., 2018; Brockschmidt et al., 2019). We follow this approach, representing the partial choice by typed relations for both subgoals and actions.

3 BACKGROUND

Classical planning and general policies. We consider typed STRIPS domains with negative preconditions and positive goals. A domain specifies types, predicates and action schemas; a problem instance P determines a finite object set $\mathcal{O}$, an initial state I and a set of goal atoms G, interpreted conjunctively. A state s is a set of true ground atoms. A ground action a is applicable when $\mathrm{pre}^+(a) \subseteq s$ and $\mathrm{pre}^-(a) \cap s = \emptyset$, where the two sets contain the atoms required to be true and false, respectively. Applying it yields $(s \setminus \mathrm{del}(a)) \cup \mathrm{add}(a)$. Predicates absent from all action effects are static, others are fluent. A plan is an applicable action sequence from I to a state containing G, with length measured in primitive actions. We write $P[s, G]$ for the same instance with initial state s and goal G, and $P[s, g]$ for $P[s, \{g\}]$. A general policy is able to select actions from states and goals across instances of a domain.

Landmarks. A fact landmark is an atom that holds at least once along every solution plan, possibly in the initial state. A disjunctive landmark is a set of atoms such that every solution plan visits a state satisfying at least one member (Hoffmann et al., 2004). Different plans may satisfy different members, so a member need not itself be a fact landmark. We extract landmarks from a simpler task, the delete relaxation. It removes delete effects and negative fluent preconditions while preserving static conditions, so reached facts persist. Every plan for the original task also solves this relaxation. Its fact and disjunctive landmarks therefore constitute landmarks in the original task as well.

Width-based search. IW(k) performs breadth-first search, retaining a newly generated state only if some set of at most k atoms holds jointly for the first time in the search (Lipovetzky & Geffner, 2012). For any fixed search parameter k, its time and space are polynomial in the grounded task size, independently of the problem's actual width. IW(2) tests pairs of atoms. Our landmark-guided variant, LIW(1), tests only pairs of true atoms consisting of one from the landmark vocabulary and a freely chosen one, and a state in which no landmark atom holds is tested as in IW(1). The '1' counts the 'free' atom in the pair. LIW(1) thus keeps every state that IW(1) keeps and at most the states that IW(2) keeps.

Relational graph neural networks. Planning states are relational structures over predicates of different arities and ordered arguments. Relational graph neural networks (R-GNNs; Ståhlberg et al., 2022a) adapt message passing to these structures. Given ground atoms $\mathcal{R}$ over objects $\mathcal{O}$, each object is a node and each atom is a hyperedge with ordered argument positions. In our state encoder, object embeddings $h_o^l \in \mathbb{R}^d$ start at $h_o^0 = \mathbf{0}$. At each layer $0 \le l < L$, an atom $q = P(o_1, \ldots, o_r)$ of arity r sends r messages through a predicate-specific multilayer perceptron:

$$\mathrm{MLP}_P\big(h_{o_1}^l \,\|\, \cdots \,\|\, h_{o_r}^l\big) = \big(m_{q,1}^l, \ldots, m_{q,r}^l\big) \in \mathbb{R}^{r \times d}, \tag{1}$$

where $\|$ denotes concatenation and $m_{q,j}^l \in \mathbb{R}^d$ is sent to the object at position j. The same MLP_P serves all atoms of P, and its outputs distinguish argument positions. Each object aggregates its incident messages with a coordinatewise smooth maximum $m_o^l = \frac{1}{\beta} \log \sum_{(q,j) \in \mathcal{I}(o)} \exp\big(\beta m_{q,j}^l\big)$, where $\mathcal{I}(o)$ contains the occurrences of o at position j of atom q, and $\beta > 0$ controls the approximation to the maximum. Our encodings include unary type atoms, so every object receives messages; nullary atoms use an auxiliary node (Appendix C.1). Embeddings are updated synchronously as $h_o^{l+1} = h_o^l + \mathrm{MLP}_U\big(h_o^l \,\|\, m_o^l\big)$, with MLP_U shared by all objects and both message and update functions shared across layers. Object readouts use h_o^L, while a graph readout pools embeddings,

for example $H = \sum_{o \in \mathcal{O}} h_o^L$ and $V(\mathcal{R}) = \mathrm{MLP}_V(H)$. Renaming objects permutes their embeddings (equivariance) and leaves the pooled value unchanged (invariance). Shared parameters allow different numbers of objects.

4 Learning to Select and Achieve Subgoals

4.1 Two-Level Hierarchical Policy

The manager policy $\pi_M(g \mid s, G)$ selects the next subgoal, one landmark atom g, from state s and task goal G. The worker policy $\pi_W(a \mid s, g)$ selects primitive actions from s and g alone, defining a general policy for the single-atom problems $P[s, g]$.

We extract fact landmarks L_{fact} and disjunctive landmark sets $\mathcal{L}_{\mathrm{disj}}$ once per instance. Let L_{land} be the union of L_{fact} and all $\mathcal{L}_{\mathrm{disj}}$ sets, then the candidate vocabulary in state s is $\mathcal{C}(s) = L_{\mathrm{land}} \cap \mathcal{F}_{\mathrm{dyn}}(s)$, where $\mathcal{F}_{\mathrm{dyn}}(s)$ denotes the valid fluent atoms that do not hold in s. The extractor applies the lifted necessary-subgoal rule of Wichlacz et al. (2022) without grounding the actions. On the largest childsnack test instance, it takes 3.1 s where a grounded extractor takes 37 minutes and 24 GB of memory. Appendix A gives the procedure and its soundness conditions. Landmark membership fixes neither a useful order nor the difficulty of the induced subproblem, so the choice of which candidate to pursue next is precisely the manager's learning task. For simplicity, we refer from now on to fact and disjunctive landmarks jointly as landmarks.

Starting from the initial state $s = I$, the manager selects $g \in \mathcal{C}(s)$ and the worker attempts $P[s, g]$. This attempt returns a state s' and succeeds when $g \in s'$. If the task goal is not yet satisfied, the manager chooses its next subgoal from s'. A successful run composes these atomic subproblems into a solution. During manager training, LIW(1) achieves the subgoals instead of the worker, and at test time the learned worker replaces it without retraining the manager.

The selected subgoals induce a state-dependent serialization, with no bounded-width guarantee. Moreover, g leaves other facts unspecified, so achieving it may undo a task-goal fact that already holds.

A decision consists of a predicate or action schema followed by its ordered object arguments. For predicate p of arity k, the policy factorizes the atom $p(o_1, \ldots, o_k)$ as

$$\pi\left(p(o_1, \ldots, o_k) \mid s, G\right) = \pi_{\mathrm{pred}}(p \mid s, G) \prod_{i=1}^{k} \pi_{\mathrm{arg}}(o_i \mid s, G, p, o_{<i}) \tag{2}$$

and likewise for action $A(o_1, \ldots, o_k)$. We call the outputs the tokens $t_0, \ldots, t_k$ of the decision.

4.2 Learning

We train the two policies separately, each with a fixed width-based search procedure, so the two can be trained in parallel. LIW(1) achieves the manager's subgoals, and the worker imitates plans of IW(2). Both use advantage-weighted regression (Peng et al., 2019), in which a learned estimate of the remaining distance supplies the advantages that weight the actor's cross-entropy loss.

Manager. We treat each chosen subgoal as one decision of a goal-conditioned reinforcement learning problem. From state s with task goal G, the manager selects $g \in \mathcal{C}(s)$, and LIW(1) attempts $P[s, g]$, returning the reached state s' and whether g was achieved. A subgoal costs 1, whether or not it is achieved, and entering a dead end costs $1/(1-\gamma)$, where γ is the discount factor. The manager is thus trained to complete the task with as few subgoals as possible without knowing the primitive action cost a subgoal incurs. Since LIW(1) is a fixed algorithm, the manager trains in a stationary environment and needs no correction for changes in the worker (Nachum et al., 2018).

The actor learns from the achieved subgoals in a replay buffer of its rollouts. For a subgoal with tokens $t_0, \ldots, t_k$, we minimise the weighted cross-entropy

$$\mathcal{L}_{\mathrm{actor}} = -\mathbb{E}\Big[\frac{w}{k+1} \sum_{i=0}^{k} \log \pi_M(t_i \mid s, G, t_{<i})\Big], \qquad w = \min\big(e^{A/\beta},\, w_{\max}\big), \tag{3}$$

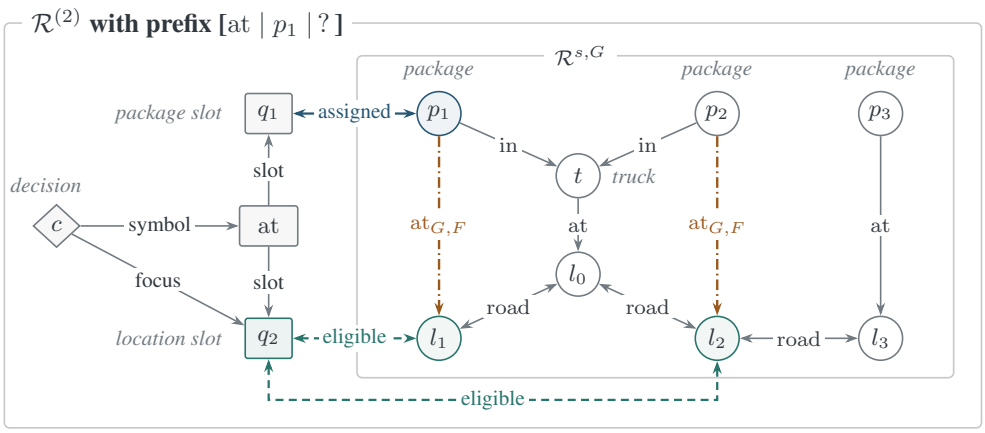

Figure 2: Decision graph $\mathcal{R}^{(2)}$ for choosing the location in $\text{at}(p_1, ?)$. The inner box contains the state and goal atoms $\mathcal{R}^{s,G}$, with $\text{at}_{G,F}$ encoding unsatisfied goals. The decision node c, the predicate node and the slots q_1, q_2 record the prefix. Package p_1 is assigned to q_1, and focus marks q_2, whose eligible locations $E_2 = \{l_1, l_2\}$ are the goal locations of the packages in the truck. The assignment lets message passing relate both to p_1 before they are scored. Italic descriptions indicate node roles, and labels on arrows name relations. Double arrows combine the two directions of a relation.

where β is a temperature, $w_{\max}$ a weight upper bound, and w is held fixed when differentiating. A distance head predicts $\delta(s, G)$, the number of subgoals still needed ($\delta = 0$ when G holds), then $A = \delta(s, G) - \delta(s', G)$. The distance head is fitted to the numbers of subgoals that remained in successful rollouts, normalised per instance. A critic that shares the network with the actor adds an auxiliary loss. Appendix C.4 adds details together with the rewards and the replay sampling.

Worker. The worker learns from plans that IW(2) finds from sampled training states. IW(2) prunes fewer states than LIW(1) and finds more plans of optimal length, but it is too costly to run for every subgoal at test scale. Each search is goal-free and records a plan for every landmark atom it reaches, so one search supplies examples for several atomic goals. For a reached atom g, every state s along its plan gives an example (s, g) with the next action of the plan. At the start state of the search, the worker increases the total probability of the first actions of all shortest plans found, and decreases that of actions that only start longer plans. At the other states of a plan, only the next plan action is known, and the worker imitates it with the weight of Eq. (3). The advantage in this weight comes from a value head $V(s, g)$, trained on the returns of these transitions with a cost of 1 per action and bounded by the known plan length (Aichmüller et al., 2026a). Appendix C.4 gives both losses.

4.3 State and Decision Representation

Both policies use the R-GNNs described in Section 3, with separate parameters for the manager and worker. A state s and a goal G over the objects $\mathcal{O}$ of an instance are encoded as the atom set

$$\mathcal{R}^{s,G} = \{P(\bar{o}) \mid P(\bar{o}) \in s\} \cup \{P_{G,T}(\bar{o}) \mid P(\bar{o}) \in G \cap s\} \cup \{P_{G,F}(\bar{o}) \mid P(\bar{o}) \in G \setminus s\}, \qquad (4)$$

where $\bar{o}$ is a tuple of objects, including domain constants. The first set contains the true atoms, while the other two represent satisfied and unsatisfied goal atoms through the predicates $P_{G,T}$ and $P_{G,F}$. The worker uses the same encoding with $G = \{g\}$.

Each output predicate or object appends to the prefix which we encode into the graph. First, a classifier-readout on the pooled object embeddings scores the predicates occurring in $\mathcal{C}(s)$. The worker instead scores its schemas after a first pass, over a graph that adds a node for each schema it can select. Afterwards, we add the selected predicate p as a node, alongside a decision node c, and a slot node q_j for each argument, indexed in decoding order. Typed relations mark the current argument, connect it to its eligible objects E_i, and connect the earlier slots to their assigned objects.

A second network processes this decision graph (Figure 2) at every step. Its first pass starts from the encoder's object embeddings and each later pass from the embeddings of the previous one, so

later choices can use information propagated during earlier choices (Appendix C.1). The score of an eligible object is computed from its own embedding, the embeddings of the decision and slot nodes, and the pooled object embeddings. All inputs are taken after the current pass, so the score depends on the prefix encoded in the graph. Objects that were indistinguishable before a choice can thus receive different scores afterwards, increasing the network's expressivity. Appendix B gives an example and its assumptions, and Appendix C.1 the relation set, the scores, normalised by a softmax over E_i, and further details on the architecture.

Eligible and Achiever objects. For the manager, E_i contains the objects that extend the prefix $(p, o_1, \ldots, o_{i-1})$ to an atom of $\mathcal{C}(s)$. We store these atoms in a trie, which records their shared prefixes and the next objects allowed at each prefix. The manager therefore generates an atom from the landmark vocabulary without separately encoding or enumeratively scoring each complete atom.

For the worker, E_i is determined by the schema's preconditions. It contains the objects of the parameter's type that satisfy every precondition literal whose last unassigned parameter is the current one, given the selected objects. Every completed action is thus applicable (Appendix C.2).

Precondition checks restrict which actions the worker can apply, but do not identify which choices help achieve g. We therefore add achiever relations $\mathrm{achiever}(o, q_i)$ by matching subgoals to schema effects, checking parameter bindings against the current state, and regressing through missing preconditions within fixed search and depth limits. The relations mark schemas and argument choices that achieve either g or a selected prerequisite. When a binding search reaches its limit without finding an applicable action, approximate argument marks are used instead. These relations inform the network without changing the eligible sets. Appendix C.3 gives the construction and its limits.

For a decision of arity k, a greedy choice requires one encoder pass and k further passes, plus one for the worker's schema step. Beam search runs a pass for each of its prefixes. No complete candidate is represented or scored separately.

5 Experiments

We evaluate how the manager and worker generalize to larger instances. We compare Landmarker with planners and policies, and replace search with the learned worker under the same manager. We then analyse which subgoals the manager learns and where the system fails.

Domains and data. We use the ten domains of the IPC 2023 learning track (Taitler et al., 2024). Each domain has 90 test instances in three tiers of 30, easy, medium and hard. We split the official training instances of each domain into 59 to 69 for training and 30 for validation. Training instances have at most 40 objects, 124 in sokoban, and test instances up to 1 461, or 9 880 in sokoban. For checkpoint selection, we generate ten further instances per domain at the scale of each test tier. Appendix D gives the sizes of all instance sets and how the selection instances are generated.

Training and selection. We train managers with four random seeds per domain for up to 400 episodes, and one worker per domain for up to 50 epochs. We save a manager checkpoint every ten episodes and evaluate it on the 30 validation instances. Checkpoints that solve all of them are then evaluated on the selection instances. We select the five best checkpoints of each domain by validation coverage, followed by coverage on the hard and medium selection instances. The worker used during selection is an earlier model that restricts arguments only by type. We replace it with the worker of Section 4.3 for testing, without repeating manager selection. Appendix C.5 gives the training schedules, the complete selection procedure and the hardware.

Testing. Each test instance has a limit of one hour for every combination of manager and executor, and a limit of 1 000 manager decisions. The learned worker may apply at most 100 primitive actions per subgoal. Landmarks are extracted before an instance is attempted, outside the time limit. Each policy returns the most probable subgoal or action found by a beam search of width 8 over its partial choices, and a closed set of visited states prevents revisits (Appendices C.4 and C.5).

Validation is weak at predicting the generalization of checkpoints to such larger test instances. We therefore pair the five selected manager checkpoints with the first-ranked worker checkpoint, and

in transport and childsnack with two further checkpoints, 66 pairings in all, and report the pairing with the highest coverage in each domain, a best case, since we ask whether the learned policies can generalize at all. Appendix E gives the tiers (Table 6) and the mean and spread over the pairings.

Baselines. We use the results reported by Aichmüller et al. (2026b) for their primitive action policy (Flat–AA), which scores each applicable action enumeratively, their Abstracted IW-policy (AIW–AD), and LAMA (Richter & Westphal, 2010; Helmert, 2006). Results for Distincter (Bai et al., 2025) and WL-GOOSE (Chen et al., 2024b) are taken from their papers. To compare executors, we keep the selected manager checkpoints and replace the learned worker with LIW(1) or with a worker based on GABAR (Mangannavar et al., 2025). The GABAR worker learns from the same IW(2) plans by behaviour cloning. We also test GABAR as a flat policy, trained by behaviour cloning on optimal plans as in its original setup (Appendix C.5).

Results. The summary rows of Table 1 compare the systems per domain and tier. Landmarker with our learned worker proves highly competitive, solving all 90 test instances in each of five domains, compared with three for AIW–AD. With the same managers, the learned worker solves 217 medium and 158 hard instances against 162 and 14 for LIW(1), which is the stronger executor only on the easy tier (298 against 273). The worker's achiever relations matter at scale. With direct achievers only, it solves no hard ferry and four hard satellite instances against 30 when the relations regress through missing preconditions (Table 7). It also improves on childsnack and, curiously, floortile over all other baselines. In particular, Landmarker with an LIW(1) worker solves 57/90 floortile instances, the highest coverage in Table 1, and all its failures there are timeouts, never dead ends. In contrast, the GABAR worker executes subgoals from the same managers beyond the easy tier mostly in miconic, satellite and ferry, and flat GABAR appears to not generalize well across the board.

Sokoban is a domain in which the worker π_W fails completely, as seen in the LIW(1) execution of the same managers, which comes close to LAMA, a best-first search over the state space (Richter & Westphal, 2010). This puzzle-like domain is known to be PSPACE-complete (Culberson, 1999), rendering it unclear whether a generalizing policy could be learned at all. Rovers is a second worker failure. On the medium instances the worker starts missing navigation subgoals within its action budget. This domain is known to require features beyond the C_2 expressivity of message passing (Ståhlberg et al., 2022a), which the relations added during decoding either do not supply or are hard to generalize. Spanner poses a concrete dead-end learning failure, in which all but the easy instance set was failed by the manager's subgoaling. With the dead ends of the training instances, known from their state spaces, supervising the distance head, coverage rises to 40, and domain knowledge in the form of a nullary derived relation on top that states whether the carried spanners suffice brings the manager to 89/90 on spanner as well, recovering the failure case (Appendix E).

Manager ablation. Figure 3 shows that outside blocksworld and spanner almost all subgoals the manager chooses are goal atoms, so there it learns an order of the goal atoms, and that the worker needs a median of 1.5 to 4.5 actions per subgoal outside childsnack and sokoban. To see what this choice is worth, we replace the manager by a random selector and keep the learned worker, once drawing a random unsatisfied goal atom at every decision and once a random currently false landmark atom (Tables 10 and 11). The comparison separates two things the manager learns. The first is which atoms to pursue. A random landmark atom solves 95 instances, since most draws cannot be reached from the current state, whereas restricting the draw to goal atoms already gives 487, and the manager 648. The second is the order. In miconic, ferry, satellite and transport any order of the goal atoms solves the instances, and the manager's order shows in the plans: a random order's plans are 3 to 28 % longer with about the same number of subgoals. In childsnack and floortile the order decides whether the worker succeeds at all, 67 against 10 and 43 against 0. In blocksworld the manager's non-goal landmarks are the decomposition itself, with 3.4 times fewer subgoals, 4.5 times shorter plans and all 30 hard instances solved against 4.

Limitations. The manager does not reliably learn to avoid dead ends (spanner, sokoban). An atomic subgoal leaves other facts unspecified, so achieving it can undo an earlier one, and how to prevent this is not addressed. The manager is trained with LIW(1) and never sees the learned worker, so it may select subgoals the worker cannot achieve. Landmark extraction takes a median of 31 minutes on the hard sokoban instances.

Table 1: Coverage per domain and method on the 90 test instances of each domain, summed over the 300 instances of each tier and over all 900. The same managers of Landmarker run with the learned worker π_W, LIW(1), or a GABAR worker, and flat GABAR runs without a manager, all with one hour per instance. The other columns are published results with their own limits: one hour for AIW–AD and LAMA, the latter with 90 GB of memory, 30 minutes and 8 GB for Distincter and WL-GOOSE, and 15 minutes for Flat–AA. Distincter and WL-GOOSE report no results per tier. Parentheses give percentages, and the highest coverage of each row is bold.

	Landmarker			Flat policies			Systematic search		
Domain	π_W	LIW(1)	GABAR	GABAR	Flat–AA	AIW–AD	LAMA	Distincter	WL-GOOSE
blocksworld	**90 (100)**	47 (52)	0 (0)	11 (12)	76 (84)	**90 (100)**	57 (63)	88 (98)	75 (83)
miconic	**90 (100)**	74 (82)	68 (76)	36 (40)	84 (93)	**90 (100)**	**90 (100)**	**90 (100)**	**90 (100)**
spanner	30 (33)	29 (32)	29 (32)	30 (33)	**90 (100)**	**90 (100)**	30 (33)	**90 (100)**	73 (81)
satellite	**90 (100)**	45 (50)	66 (73)	16 (18)	51 (57)	88 (98)	**90 (100)**	48 (53)	53 (59)
transport	**90 (100)**	47 (52)	42 (47)	13 (14)	74 (82)	86 (96)	72 (80)	50 (56)	29 (32)
ferry	**90 (100)**	60 (67)	61 (68)	55 (61)	62 (69)	80 (89)	74 (82)	83 (92)	76 (84)
rovers	54 (60)	43 (48)	30 (33)	3 (3)	27 (30)	67 (74)	**78 (87)**	42 (47)	37 (41)
childsnack	**67 (74)**	36 (40)	33 (37)	30 (33)	41 (46)	64 (71)	36 (40)	64 (71)	29 (32)
sokoban	4 (4)	36 (40)	0 (0)	2 (2)	14 (16)	13 (14)	**44 (49)**	32 (36)	38 (42)
floortile	43 (48)	**57 (63)**	0 (0)	10 (11)	0 (0)	0 (0)	15 (17)	2 (2)	2 (2)
Easy	273 (91)	**298 (99)**	204 (68)	173 (58)	246 (82)	253 (84)	285 (95)	n/a	n/a
Medium	217 (72)	162 (54)	98 (33)	33 (11)	177 (59)	**233 (78)**	193 (64)	n/a	n/a
Hard	158 (53)	14 (5)	27 (9)	0 (0)	96 (32)	**182 (61)**	108 (36)	n/a	n/a
Total (900)	648 (72)	474 (53)	329 (37)	206 (23)	519 (58)	**668 (74)**	586 (65)	589 (65)	502 (56)

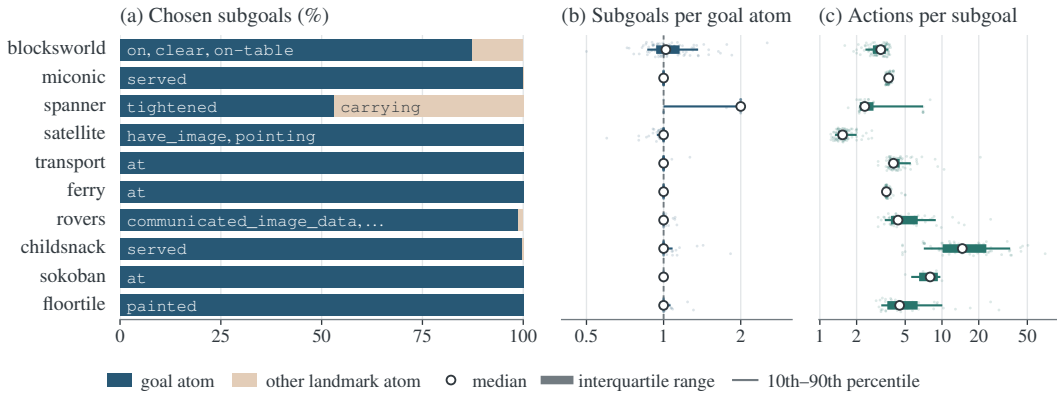

Figure 3: Learned subgoals on the test instances solved by the manager–worker pairings of Table 1. (a) Share of chosen subgoals that are goal atoms or other landmark atoms, with the most frequent predicates of each. (b) Subgoals per goal atom and (c) primitive actions per subgoal.

6 Conclusion

We introduced Landmarker, a two-level policy whose manager selects landmark atoms as subgoals and whose worker learns to achieve them, trained independently of each other with width-based search. The landmark control loop of Hoffmann et al. (2004), which pursues landmarks in the order of the landmark graph, performed worse than search with a landmark heuristic (Richter & Westphal, 2010). Landmarker learns the order instead and solves 648 of the 900 test instances of the IPC 2023 learning track, more than LAMA, the learned heuristics and the flat policies, and close to the 668 of AIW–AD. The theory of sketches says that subgoaling works in domains that admit decompositions into subproblems of low width (Drexler et al., 2022; Bonet & Geffner, 2024). The results are consistent with the manager finding such decompositions by reinforcement learning, as in Aichmüller & Geffner (2025). Our generative worker is able to generalize and scale to much larger instances, while achieving most subgoals effectively with only few actions. The manager's ability to select and order landmarks is vital in some domains in order to solve them at all, as a random order of the same goal atoms loses coverage where the goals interact, while in others the order matters less and only plan quality is reduced by random selection. Landmarker returns landmarks to their role as subgoals by learning which one to pursue next and how to achieve it.

REPRODUCIBILITY STATEMENT

Section 5 describes the domains, the data splits, training, checkpoint selection and the test protocol. The test instances are those of the IPC 2023 learning track. Appendix A explains the lifted landmark extraction and its soundness argument. Appendix C provides the relational networks, the decision graph construction, the precondition checks, the achiever relations, learning and decoding settings, the training schedules, the complete checkpoint selection and the hardware. Appendix D gives the sizes of all instance sets and how the checkpoint-selection instances are generated. Our implementation uses PyTorch and the planning library Mimir. Code and data will be released upon publication.

AI USAGE STATEMENT

In this work, we used generative AI tools to formulate the statement of the proposition in Appendix A.1, to provide the ingredients of its proof sketch and to write it, to implement parts of the method, to generate the checkpoint-selection instances of Appendix D.2 with the official IPC 2023 generators, to clean and reformat the recorded runs into the data behind the tables and figures, and to give feedback on drafts of the paper, including its experiments. We have not used generative AI tools to develop the conceptual framework, to propose or refine hypotheses, to design the experiments, to interpret the results or to assist with translation, and qualitative or thematic data analysis is not applicable to this work. Additionally, we used generative AI tools to create and edit software code, to create the tables and figures from the recorded runs, to orchestrate and monitor experiments, to search for and identify relevant literature, to draft parts of the paper, to edit it for readability and to format references. We have reviewed all AI-assisted work. The proposition and its proof sketch were derived with the help of an AI assistant from the implementation of the extractor, and the authors checked them against the code and against the empirical soundness check of Appendix A.2. The research questions, the method, the experimental design, and all experimental data are the authors' own. Every claim in the paper was checked by the authors against the code and the run artefacts, and AI-generated code was reviewed and tested by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Michael Aichmüller and Hector Geffner. Sketch decompositions for classical planning via deep reinforcement learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2025. doi: 10.24963/ijcai.2025/938. URL https://www.ijcai.org/proceedings/2025/938.

Michael Aichmüller, Yannik Hesse, and Hector Geffner. Learning to search and searching to learn for generalization in planning. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026a. URL https://openreview.net/forum?id=Ck6Ort0wqc. To appear.

Michael Aichmüller, Simon Ståhlberg, Martin Funkquist, and Hector Geffner. Efficient lookahead encoding and abstracted width for learning general policies in classical planning. arXiv preprint arXiv:2605.18674, 2026b. URL https://arxiv.org/abs/2605.18674.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 1726–1734, 2017. doi: 10.1609/aaai.v31i1.10916.

Yingbin Bai, Sylvie Thiébaux, and Felipe Trevizan. Learning efficiency meets symmetry breaking. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 35, pp. 154–159, 2025. URL https://arxiv.org/abs/2504.19738.

Pablo Barceló, Egor V. Kostylev, Mikaël Monet, Jorge Pérez, Juan Reutter, and Juan-Pablo Silva. The logical expressiveness of graph neural networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/pdf?id=r1lZ7AEKvB.

Blai Bonet and Hector Geffner. General policies, subgoal structure, and planning width. *Journal of Artificial Intelligence Research*, 80:475–516, 2024. doi: 10.1613/jair.1.15581. URL https://arxiv.org/abs/2311.05490.

Marc Brockschmidt, Miltiadis Allamanis, Alexander L. Gaunt, and Oleksandr Polozov. Generative code modeling with graphs. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1805.08490.

Dillon Z. Chen, Sylvie Thiébaux, and Felipe Trevizan. Learning domain-independent heuristics for grounded and lifted planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 20078–20086, 2024a. URL https://arxiv.org/abs/2312.11143.

Dillon Z. Chen, Felipe Trevizan, and Sylvie Thiébaux. Return to tradition: Learning reliable heuristics with classical machine learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, pp. 68–76, 2024b. URL https://arxiv.org/abs/2403.16508.

Joseph C. Culberson. Sokoban is PSPACE-complete. In *Proceedings of the International Conference on Fun with Algorithms*, pp. 65–76. Carleton Scientific, 1999.

Konrad Czechowski, Tomasz Odrzygóźdź, Marek Zbysiński, Michał Zawalski, Krzysztof Olejnik, Yuhuai Wu, Łukasz Kuciński, and Piotr Miłoś. Subgoal search for complex reasoning tasks. In *Advances in Neural Information Processing Systems*, 2021. URL https://proceedings.nips.cc/paper_files/paper/2021/hash/05d8cccb5f47e5072f0a05b5f514941a-Abstract.html.

Peter Dayan and Geoffrey E. Hinton. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 271–278, 1993.

Thomas G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000. doi: 10.1613/jair.639.

Dominik Drexler, Jendrik Seipp, and Hector Geffner. Learning sketches for decomposing planning problems into subproblems of bounded width. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 2022. doi: 10.1609/icaps.v32i1.19786. URL https://ojs.aaai.org/index.php/ICAPS/article/view/19786.

Dominik Drexler, Simon Ståhlberg, Blai Bonet, and Hector Geffner. Symmetries and expressive requirements for learning general policies. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, 2024. doi: 10.24963/kr.2024/79. URL https://proceedings.kr.org/2024/79/.

Guillem Francès, Blai Bonet, and Hector Geffner. Learning general planning policies from small examples without supervision. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021. doi: 10.1609/aaai.v35i13.17402. URL https://ojs.aaai.org/index.php/AAAI/article/view/17402.

Martin Funkquist, Simon Ståhlberg, and Hector Geffner. Learning to ground existentially quantified goals. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, 2024. URL https://arxiv.org/abs/2409.20259.

Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

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

Jörg Hoffmann, Julie Porteous, and Laura Sebastia. Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, 22:215–278, 2004. doi: 10.1613/jair.1492. URL https://fai.cs.uni-saarland.de/hoffmann/papers/jair04.pdf.

Rostislav Horčík and Gustav Šír. Expressiveness of graph neural networks in planning domains. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 2024. doi: 10.1609/icaps.v34i1.31486. URL https://ojs.aaai.org/index.php/ICAPS/article/view/31486.

Jaromír Janisch, Tomáš Pevný, and Viliam Lisý. Symbolic relational deep reinforcement learning based on graph neural networks and autoregressive policy decomposition. *arXiv preprint arXiv:2009.12462*, 2023. URL https://arxiv.org/abs/2009.12462. Version 4.

Junsu Kim, Younggyo Seo, and Jinwoo Shin. Landmark-guided subgoal generation in hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, 2021. URL https://arxiv.org/abs/2110.13625.

Jana Koehler and Jörg Hoffmann. On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. *Journal of Artificial Intelligence Research*, 12:338–386, 2000.

Tejas D. Kulkarni, Karthik R. Narasimhan, Ardavan Saeedi, and Joshua B. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems*, 2016. URL https://arxiv.org/abs/1604.06057.

Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning multi-level hierarchies with hindsight. In *International Conference on Learning Representations*, 2019. URL https://cs.brown.edu/people/gdk/pubs/multi_level_her.pdf.

Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018. URL https://arxiv.org/abs/1803.03324.

Nir Lipovetzky and Hector Geffner. Searching for plans with carefully designed probes. In *Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling (ICAPS 2011)*, pp. 154–161. AAAI Press, 2011. doi: 10.1609/icaps.v21i1.13470.

Nir Lipovetzky and Hector Geffner. Width and serialization of classical planning problems. In *Proceedings of the European Conference on Artificial Intelligence*, 2012. doi: 10.3233/978-1-61499-098-7-540. URL https://nirlipo.github.io/publication/lipovetzky-2012-width/.

Rajesh Mangannavar, Stefan Lee, Alan Fern, and Prasad Tadepalli. Graph neural network based action ranking for planning. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-5735.

Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, 2018. URL https://arxiv.org/abs/1805.08296.

Carlos Núñez-Molina, Juan Fernández-Olivares, and Raúl Pérez. Learning to select goals in automated planning with Deep-Q learning. *Expert Systems with Applications*, 202:117265, 2022. doi: 10.1016/j.eswa.2022.117265. URL https://arxiv.org/abs/2406.14779.

Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems*, pp. 1043–1049, 1998.

Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. arXiv preprint arXiv:1910.00177, 2019. URL https://arxiv.org/abs/1910.00177.

Julie Porteous, Laura Sebastia, and Jörg Hoffmann. On the extraction, ordering, and usage of landmarks in planning. In *Proceedings of the European Conference on Planning*, pp. 37–48, 2001. URL https://strathprints.strath.ac.uk/1874/.

Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39:127–177, 2010. doi: 10.1613/jair.2972. URL https://arxiv.org/abs/1401.3839.

Silvia Richter, Malte Helmert, and Matthias Westphal. Landmarks revisited. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 975–982, 2008. URL https://mlanthology.org/aaai/2008/richter2008aaai-landmarks/.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *International Conference on Learning Representations*, 2016. URL https://arxiv.org/abs/1511.05952.

Simon Ståhlberg and Hector Geffner. First-order representation languages for goal-conditioned RL. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026. doi: 10.1609/aaai.v40i43.40960. URL https://ojs.aaai.org/index.php/AAAI/article/view/40960.

Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning general optimal policies with graph neural networks: Expressive power, transparency, and limits. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 2022a. doi: 10.1609/icaps.v32i1.19851. URL https://ojs.aaai.org/index.php/ICAPS/article/view/19851.

Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning generalized policies without supervision using GNNs. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, 2022b. URL https://arxiv.org/abs/2205.06002.

Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning general policies with policy gradient methods. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, 2023. doi: 10.24963/kr.2023/63. URL https://proceedings.kr.org/2023/63/kr2023-0063-stahlberg-et-al.pdf.

Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning more expressive general policies for classical planning domains. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2025. doi: 10.1609/aaai.v39i25.34872. URL https://ojs.aaai.org/index.php/AAAI/article/view/34872.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2):181–211, 1999. doi: 10.1016/S0004-3702(99)00052-1. URL https://www.sciencedirect.com/science/article/pii/S0004370299000521.

Ayal Taitler, Ron Alford, Joan Espasa, Gregor Behnke, Daniel Fišer, Michael Gimelfarb, Florian Pommerening, Scott Sanner, Enrico Scala, Dominik Schreiber, Javier Segovia-Aguas, and Jendrik Seipp. The 2023 International Planning Competition. *AI Magazine*, 45(2):280–296, 2024.

Sam Toyer, Sylvie Thiébaux, Felipe Trevizan, and Lexing Xie. ASNets: Deep learning for generalised planning. *Journal of Artificial Intelligence Research*, 68:1–68, 2020. doi: 10.1613/jair.1.11633. URL https://arxiv.org/abs/1908.01362.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pp. 2094–2100, 2016.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. FeUdal networks for hierarchical reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 2017. URL https://proceedings.mlr.press/v70/vezhnevets17a.html.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, 2015. URL https://arxiv.org/abs/1506.03134.

Xiyuan Wang, Pan Li, and Muhan Zhang. Improving graph neural networks on multi-node tasks with the labeling trick. *Journal of Machine Learning Research*, 2025. URL https://jmlr.org/papers/v26/23-0560.html.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1995–2003. PMLR, 2016. URL https://proceedings.mlr.press/v48/wangf16.html.

Julia Wichlacz, Daniel Höller, and Jörg Hoffmann. Landmark heuristics for lifted classical planning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2022. doi: 10.24963/ijcai.2022/647. URL https://www.ijcai.org/proceedings/2022/647.

Danfei Xu, Roberto Martín-Martín, De-An Huang, Yuke Zhu, Silvio Savarese, and Li Fei-Fei. Regression planning networks. In *Advances in Neural Information Processing Systems*, 2019. URL https://arxiv.org/abs/1909.13072.

Jiaxuan You, Jonathan M. Gomes-Selman, Rex Ying, and Jure Leskovec. Identity-aware graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 10737–10745, 2021.

Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. Labeling trick: A theory of using graph neural networks for multi-node representation learning. In *Advances in Neural Information Processing Systems*, 2021. URL https://papers.nips.cc/paper_files/paper/2021/file/4be49c79f233b4f4070794825c323733-Paper.pdf.

A Lifted Landmark Extraction

For the largest childsnack instance p30-hard a grounded landmark extractor needs 37 minutes and 24 GB of memory. We instead extract landmarks directly from the action schemas, building on the *lifted necessary-subgoal* rule of Wichlacz et al. (2022), which takes only 3.1 s on the same instance instead (Table 2). We extend this rule with filters based on static preconditions and relaxed reachability, and test whether individual members of disjunctive landmarks are themselves necessary. Every returned landmark by this extractor is a landmark of the delete relaxation and hence of the original task (Appendix A.1).

Setting. We consider typed STRIPS domains with negative preconditions and positive literals as goals. Delete relaxation ignores delete effects and negative preconditions on fluent predicates. Static preconditions keep their meaning, including negative literals and parameter inequalities, since their truth never changes.

A partially ground atom $P(v_1, \ldots, v_k)$ fixes some arguments to objects and leaves the others unbound, written $*$. Its ground instances fill the unbound positions with objects of the required types. For example, $\mathrm{at}(p, *)$ fixes package p but leaves its location open. An achiever (A, σ) of $P(\boldsymbol{v})$ pairs an action schema A with a substitution σ that makes one of its add effects $+P(x_1, \ldots, x_k)$ agree with the bound positions of $\boldsymbol{v}$. At each such position, a parameter x_j is bound to v_j, while a constant must already equal v_j. Repeated parameters must receive consistent bindings. Each matching add effect gives a separate achiever.

Backward chaining. We start from the goal atoms and derive further landmarks through the preconditions of their achievers. A landmark is represented by a partially ground atom $P(w)$ and a set M of its ground instances: every plan must pass through a state containing at least one member of M. Each goal atom starts as a landmark whose only member is that atom. If a member already holds in I, we record the landmark but do not expand it, since no action is needed to establish it.

To expand a landmark, we collect the achievers $(A_1, \sigma_1), \ldots, (A_m, \sigma_m)$ that survive the filters below. If none survives, expansion stops at this landmark. For each fluent predicate Q occurring in a positive precondition of every A_i, we choose one such precondition $Q(v^i)$ per achiever. We then retain the object bindings on which these preconditions agree: the derived atom $Q(w')$ binds position j to o when $v^i_j \sigma_i = o$ for every i, and leaves it unbound otherwise. Its members are the ground instances of the chosen atoms $Q(v^i)\sigma_i$ that respect the types and static preconditions of their achievers. For example, two achievers requiring $\mathrm{at}(p, l_1)$ and $\mathrm{at}(p, l_2)$, respectively, yield $\mathrm{at}(p, *)$ with these two members. At least one must hold before the landmark being expanded is first achieved.

The rule is sound because the first action that achieves a member of M must be an instance of one of the surviving achievers. Its chosen Q-precondition holds immediately before that action, so the derived member set is again a landmark (Wichlacz et al., 2022). If an achiever has several preconditions on Q, different choices can give different landmarks. We use all combinations when

there are at most 64. Otherwise, we choose only the first Q-precondition in each achiever. A set with one member gives a fact landmark. We continue until no derived landmark remains to expand.

Static filter. Static preconditions can rule out achievers or force parameter bindings before we compare their fluent preconditions. We match each positive static precondition against the instance's static atoms. If none matches, the achiever has no applicable instance and is dropped. If all matches bind a parameter to the same object, we fix that binding. We also drop an achiever when a fully bound negative static precondition is violated, and repeat these steps until no further change occurs. The remaining achievers may then agree on more argument positions, yielding more specific landmarks. For $\mathrm{served}(c)$ in childsnack, for example, the static allergy status of child c excludes one of its two serving schemas, and $\mathrm{waiting}(c, p)$ fixes the child's place p.

Reachability in the relaxation. The next filters use the atoms reachable in the delete relaxation. We compute a set R of these atoms without grounding actions. Each add effect of a schema becomes the head of a rule whose body contains the positive fluent preconditions and the static preconditions of both signs. Starting from I, we apply the rules until no new atom is derived. Rule bodies are evaluated by joins, which match precondition atoms while requiring agreement on shared parameters, as in the grounding of the Fast Downward translator (Helmert, 2006).

For a set of atoms M, we also compute a restricted closure $R_{\neg M}$, starting from $I \setminus M$ and discarding every derived atom in M. This set contains every atom reachable along a sequence of relaxed actions in which no member of M ever holds. It may contain additional atoms because discarding one add effect does not discard the other effects of the same action. For example, if $I = \emptyset$ and the only action has no preconditions and adds both f and q, then $R_{\neg\{f\}}$ contains q, although reaching q necessarily makes f true. These additional atoms can weaken the filters below, but do not affect their soundness.

Reachable members and objects. We remove members outside R, since no plan can make them true. In miconic `p30-hard`, this reduces the possible `origin` atoms from 95 060 to the 485 true in I: `board` deletes `origin`, and no action adds it. We also restrict each achiever's parameters to objects that occur in a joint match of all its positive preconditions, static and fluent, against R. For example, $\mathrm{at}(t, l)$ and $\mathrm{at}(p, l)$ must match at the same location l; separate matches at different locations are insufficient. If a parameter has no object left, we drop the achiever.

Restricted first achievers. An action may achieve a landmark without being able to achieve it for the first time. Before the first achievement of a member of M, no member has held, so the action's positive preconditions must lie in $R_{\neg M}$. We therefore narrow the achievers using $R_{\neg M}$ in place of R. At the unbound positions of $P(w)$, we also retain only objects occurring at those positions in some member of M, and restrict the members of each derived landmark to $R_{\neg M}$. The remaining schema instances form a superset of the possible first achievers of Richter et al. (2008). In blocksworld, if x initially covers b and b is not held, only $\mathrm{unstack}(x, b)$ can first make $\mathrm{clear}(b)$ true: putting down or stacking b would require holding it, which already requires clearing it. The precondition $\mathrm{clear}(x)$ then becomes a fact landmark, allowing the chain to continue up the tower.

The restricted closure depends on the member set, so we identify each landmark by both its partially ground atom and its members. For example, a sokoban box that must pass through several corridor cells gives a separate fact landmark for each cell, although all share $\mathrm{at}(box, *)$. Merging them by the partially ground atom alone would retain only the weaker statement that the box visits some cell in the set. We discard a derived member set when it contains the members of a recorded landmark of the same predicate, since the recorded landmark already implies it. Each expanded landmark requires one restricted closure.

Fact landmarks from the restricted closure. The restricted closure also lets us test whether an individual atom $f \notin I$ is necessary. If a goal atom lies outside $R_{\neg\{f\}}$, no plan of the relaxation can reach the goal without making f true, so f is a fact landmark. If all goal atoms remain in the closure, the test is inconclusive because the closure may contain atoms that cannot be reached while avoiding f. The test of Hoffmann et al. (2004) is exact for the relaxation: it removes every action that adds f, including all of that action's other add effects.

We use our test to complete the fact-landmark set by checking each member of a disjunctive landmark that is false in I. Members that pass become fact landmarks, and any disjunctive landmark

containing one is dropped. This requires one restricted closure per tested member. We turn completion off in sokoban because of its cost (Appendix A.2), and use all other filters in every domain.

Returned landmarks. The extractor returns the fact landmarks L_{fact} and the remaining member sets of size at least two as disjunctive landmarks $\mathcal{L}_{\mathrm{disj}}$. It drops a disjunctive landmark that contains a fact landmark or strictly contains another disjunctive landmark, since the smaller set already implies the larger one. It also records that each derived landmark holds immediately before the first achievement of its parent landmark. The manager does not use these orderings. LIW(1) uses all atoms in the candidate vocabulary L_{land} of Section 4 as its landmark atoms.

A.1 Soundness

Proposition 1. *Suppose that R contains every atom reachable in the relaxation, and that $R_{\neg M}$ contains every atom that holds after a sequence of relaxed actions during which no member of M holds. Then every fact landmark and every disjunctive landmark returned by the procedure of Appendix A is a landmark of both the relaxation and the task, for any choice of the optional filters and completion.*

Both assumptions hold for the closures defined above, with $R = R_{\neg\emptyset}$. Consider a relaxed action sequence during which no member of M holds. Its initial atoms belong to $I \setminus M$, and none of its actions adds a member of M. By induction along the sequence, each action's positive preconditions are in $R_{\neg M}$ and its static preconditions hold, so its add effects are derived there as well.

Proof sketch. We proceed by induction over the derivation. Goal atoms are landmarks. Suppose that a landmark $P(w)$ with members M is expanded, and let π be any plan of the relaxation. Since $M \cap I = \emptyset$, the plan has a first action a that adds a member of M. This action is an instance of an achiever (A, σ) of $P(w)$.

The achiever and the objects used by a survive every filter. Its static preconditions hold, and its positive fluent preconditions hold together in the state before a, whose atoms lie in R. No member of M has held before a, so these atoms also lie in $R_{\neg M}$. Finally, because a adds a member of M, its objects at the unbound positions of $P(w)$ occur at those positions in M.

Now let $Q(w')$ be derived from $P(w)$, with $Q(v)$ chosen as the precondition in A. Its ground instance in a holds immediately before a. This atom has the required types, respects the static preconditions, and belongs to both R and $R_{\neg M}$. It also agrees with every bound position of $Q(w')$, since those positions were fixed only where all chosen preconditions agree. The atom therefore survives as a member of the derived landmark. Since π was arbitrary, every plan passes through a state containing such a member.

A landmark with one member is a fact landmark. If duplicate derivations are merged by taking the union of their member sets, either derivation implies the resulting set. Dropping a disjunctive landmark implied by a smaller one, discarding a derivation whose members contain those of a recorded landmark, and limiting the combinations of preconditions only omit landmarks.

Finally, a member f promoted by completion is false in I and satisfies $G \not\subseteq R_{\neg\{f\}}$. Any relaxed plan avoiding f would reach only atoms in $R_{\neg\{f\}}$ and thus fail to reach the goal. Hence f is a fact landmark of the relaxation. Every plan of the original task also solves the relaxation, so all returned landmarks are landmarks of the task as well. □

The extractor can still miss landmarks; the proposition guarantees that those it returns are necessary.

A.2 Comparison with Grounded Extraction

We compare the lifted extractor with a grounded extractor from the planning library we build on. The grounded extractor instantiates all actions of the relaxation. For each atom, it intersects the preconditions of the achievers that add the atom at its first occurrence in the relaxed planning graph (Hoffmann & Nebel, 2001). An action that first achieves the atom in a plan need not be among these earliest graph achievers. Their shared preconditions can therefore include atoms that some plans avoid, yielding invalid landmarks.

Soundness. We checked both extractors on 1 542 training and validation instances from thirteen domains: the ten in our evaluation, plus barman, logistics and a second blocksworld encoding. For each landmark with no member true in I, we ground the relaxation and remove every action that adds a member. If the goal then becomes unreachable, every relaxed plan must achieve a member, confirming the landmark. All 26 788 fact landmarks and 7 640 disjunctive landmarks checked for the lifted extractor pass. For the grounded extractor, breadth-first search in the original task instead finds plans avoiding 141 of its reported fact landmarks: 73 in sokoban, 37 in transport, 23 in floortile and 8 in logistics.

Vocabularies. To measure how the vocabularies affect LIW(1), we ran it with each vocabulary on 75 training instances from the thirteen domains. Each pair of runs starts in the same initial state or a state reached by a short random walk, and pursues the same atom, chosen to be false in that state and present in both vocabularies. Both runs use the same tie-breaking and limits of 50 000 states and 20 seconds. They agree on whether the goal is reached in 99.6 % of 2 548 pairs. In the 11 pairs that disagree, only the run with the grounded vocabulary reaches the goal.

Before adding the filters based on R and $R_{\neg M}$, agreement was 97.0 %, with most differences in logistics and sokoban. Restricted first achievers have the largest effect on the vocabulary, increasing it from 3 161 to 34 237 atoms in floortile `p29-hard`. Narrowing the objects of achievers also exposes more fact landmarks: blocksworld `p30-hard` has 1 947 with this step and 1 462 without it. Completion promotes 204 members to fact landmarks across 186 of the 1 542 instances.

Cost. Table 2 reports extraction time on the largest test instance of each domain. Completion accounts for most of the time in floortile, rovers, satellite and transport. In sokoban `p30-hard`, restricted first achievers produce 14 218 disjunctive landmarks with 196 778 members. Testing these members for completion prevents extraction from finishing within an hour, so we turn completion off in this domain. On sokoban's 99 training and validation instances, this removes 3 atoms from the vocabulary and adds 103, while the paired LIW(1) runs still agree on all 187 sokoban atoms. Turning off restricted first achievers instead would be faster, but removes 97 to 99 % of sokoban's disjunctive landmarks and lowers agreement to between 74.7 and 91.2 %.

Table 2: Lifted landmark extraction on the largest test instance of each domain, in seconds of wall-clock time for one process per instance on a single machine. The vocabulary size $|L_{\text{land}}|$ counts the fact landmarks and the members of disjunctive landmarks, with completion off in sokoban. Extraction takes place before an instance is attempted and outside its time limit (Section 5).

Domain	Instance	Objects	$\lvert L_{\text{land}}\rvert$	Time (s)	Without completion (s)
blocksworld	`p30-hard`	488	1 947	28.9	29.2
childsnack	`p30-hard`	1 327	6 307	3.1	1.5
ferry	`p30-hard`	1 461	3 402	110.7	112.5
floortile	`p29-hard`	1 022	34 237	117.4	31.4
miconic	`p30-hard`	681	1 649	0.8	0.8
rovers	`p30-hard`	596	23 235	609.5	55.5
satellite	`p30-hard`	402	9 713	12.1	2.3
sokoban	`p30-hard`	9 884	196 968	$> 3\,600$	2 897.6
spanner	`p30-hard`	833	2 294	0.5	0.4
transport	`p30-hard`	354	15 488	480.7	17.7

Use in training and testing. The manager uses the lifted extractor in both training and testing. The worker is trained on plans for landmarks from the grounded extractor, except in sokoban, where its training landmarks also come from the lifted extractor. An invalid fact landmark of the grounded extractor only adds an atom for which the worker imitates plans. At test time, the worker receives the manager's subgoals from the lifted vocabulary in every domain.

B Candidate Binding and Sequential Individualisation

Section 4.3 scores each argument after the prefix has been written into the decision graph. We give an example, with its assumptions, in which this distinguishes candidates that a joint encoding

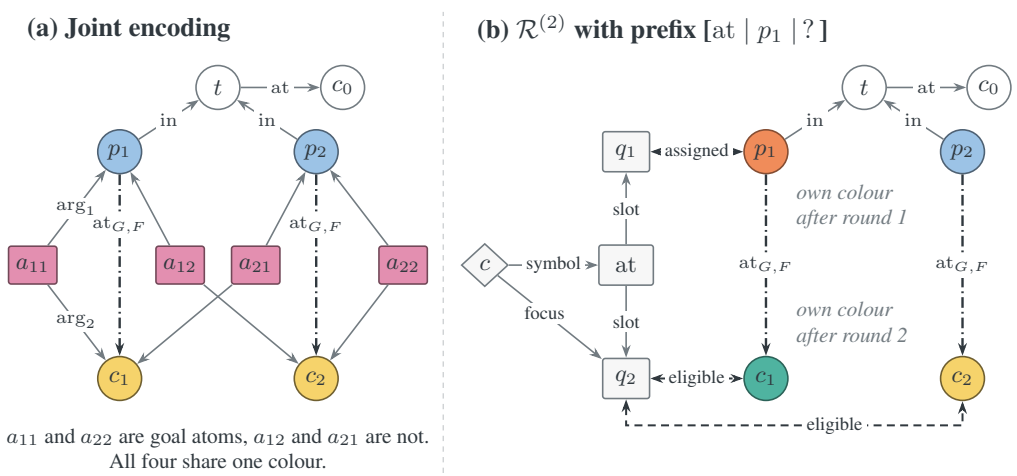

Figure 4: Colour refinement in the example for $n = 2$, after p_1 was loaded at c_2 and p_2 at c_1. The fill of an object or candidate node shows its colour: nodes with the same fill share a colour, and white objects have colours of their own. Relations are drawn in grey and named on the arrows. (a) A joint encoding adds a node a_{ij} for each candidate $\mathrm{at}(p_i, c_j)$. Only a_{11} and a_{22} form a triangle with a goal atom, but every package, location and candidate node sees the same colours among its neighbours, so all four candidates keep one colour. (b) The decision graph after p_1 is chosen, with the relations of Figure 2. The assignment gives p_1 a colour of its own in the first round. In the second, c_1 receives one as well, since the goal atom $\mathrm{at}_{G,F}(p_1, c_1)$ joins it to p_1, so the two eligible locations can receive different scores. Roads, the capacity of the truck and, in (a), the candidates for the location and capacity of the truck are omitted, since they do not change the colours shown.

of all complete candidates cannot distinguish, and compare the cost with encoding each candidate separately.

Colour refinement. The R-GNN of Section 3 updates an object from its own embedding and the messages of the atoms it occurs in. Each message depends only on the predicate, the argument position and the embeddings of the atom's arguments, so the network distinguishes at most the objects that colour refinement over the same atoms distinguishes (Barceló et al., 2020; Ståhlberg et al., 2022a). Colour refinement starts with one colour for each initial embedding. Each round then refines the colour of an object by the multiset that contains, for each atom the object occurs in, the predicate, the object's position and the colours of all arguments. If two objects keep the same colour in every round, induction over the layers shows that they have the same embedding after every layer. Any readout that combines an object's embedding with pooled embeddings then gives both the same score.

Transport example of symmetry. Consider a transport state with $n \geq 2$ packages $p_1, \ldots, p_n$ in one truck and n locations $c_1, \ldots, c_n$, where the goal contains the unsatisfied atoms $\mathrm{at}(p_i, c_i)$. The truck is at a further location c_0, and roads connect all locations, so no other atom distinguishes the packages or the locations c_j. We assume that every atom $\mathrm{at}(p_i, c_j)$ is a candidate. This holds when all atoms of $\mathcal{F}_{\mathrm{dyn}}(s)$ are candidates. Of these atoms, the landmark vocabulary of a task that starts in this state contains only the goals, since a plan can deliver each package without dropping it elsewhere. For $n = 2$, it contains all four when the packages start at each other's goal locations, p_1 at c_2 and p_2 at c_1. Their initial atoms $\mathrm{at}(p_1, c_2)$ and $\mathrm{at}(p_2, c_1)$ are landmarks that hold in I. Since $\mathcal{C}(s)$ removes only the atoms that hold in s, they become candidates once both packages are in the truck. The encoding $\mathcal{R}^{s,G}$ of Eq. (4) contains the goals as $\mathrm{at}_{G,F}(p_i, c_i)$, and a policy should prefer the candidates $\mathrm{at}(p_i, c_i)$, which are goal atoms, over candidates $\mathrm{at}(p_i, c_j)$ for $j \neq i$.

Joint encoding of candidates. Suppose that each candidate is added to the graph as a node a_{ij}, connected by the relations $\mathrm{arg}_1(a_{ij}, p_i)$ and $\mathrm{arg}_2(a_{ij}, c_j)$, and scored from the embedding of a_{ij}. Colour refinement then gives all packages one colour, all locations c_j a second and all candidate nodes a third (Figure 4a). Every package occurs in one atom with the truck, in one goal atom with a

location and in n relations with candidate nodes. Each location c_j and each candidate node likewise occurs in atoms of the same kinds, with arguments of the same colours. No colour ever splits, and a candidate a_{ii} receives the same score as a candidate a_{ij} with $i \neq j$, even with the pooled embedding of the graph as additional input. Whether a is a goal atom is expressed by

$$\exists p\, \exists c\, \big(\arg_1(a, p) \wedge \arg_2(a, c) \wedge \mathrm{at}_{G,F}(p, c)\big),$$

which relates three objects, and the example shows that message passing over this encoding cannot evaluate it.

Conditioning on the prefix. Our decoder adds no candidate nodes. After the predicate at is chosen, all packages receive the same score, and by symmetry any of them can be chosen. Suppose it is p_1. The decision graph $\mathcal{R}^{(2)}$ then contains $\mathrm{assigned}^{\leftrightarrow}(q_1, p_1)$, which gives p_1 a colour of its own in the first round of refinement. In the second round, the goal atom $\mathrm{at}_{G,F}(p_1, c_1)$ passes this colour on to c_1, which is thereafter distinguished from every other location (Figure 4b). After two layers of message passing, the score of Eq. (6) can prefer c_1. With a unary predicate $\mathrm{Assigned}$ for the chosen object, the tested property is $\exists p\, (\mathrm{Assigned}(p) \wedge \mathrm{at}_{G,F}(p, c))$, which relates only two objects. Marking chosen objects before message passing is the labeling trick for tasks on sets of nodes (Zhang et al., 2021; Wang et al., 2025), and identity-aware GNNs mark a node in the same way (You et al., 2021). Figure 2 shows this step with two packages in the truck.

Fixed embeddings. The prefix has to be written into the graph before message passing. Suppose instead that the locations are scored from embeddings computed before p_1 was chosen, together with a context vector computed from the embedding of p_1, as in recurrent or pointer decoders over one encoding of the state (Vinyals et al., 2015; Mangannavar et al., 2025). Since all packages share one embedding and all locations another, every location c_j then receives the same score. Sequential argument selection faces the same problem then.

Encoding candidates separately. The symmetry break can also be enforced by encoding each candidate in a graph of its own, in which its arguments are effectively marked by the candidate. In the graph of $\mathrm{at}(p_i, c_j)$, message passing can then test whether $\mathrm{at}_{G,F}(p_i, c_j)$ holds. This requires one graph per candidate, n^2 in the example and up to $|\mathcal{O}|^r$ for a predicate of arity r, although types and the landmark vocabulary may remove many of them. When grounding existential goals, Funkquist et al. (2024) make each partial binding visible to message passing by re-encoding every candidate extension. With N candidates and time T per graph, scoring all candidates takes time NT. The number of decoding steps of Section 4.3 is independent of N, but each step processes the whole graph of the state instead. The manager also still builds the trie of candidate atoms symbolically, although this compute is negligible in comparison.

C Architecture and Training Details

This appendix completes the description of the decision graph in Section 4.3 and records the settings of the manager and worker reported in Section 5.

C.1 Message Passing During Decoding

Decision graph. First, a predicate occurring in $\mathcal{C}(s)$ is sampled by a classifier-readout that scores predicates via $\boldsymbol{\ell}_0 = \mathrm{MLP}_{\mathrm{pred}}\big(H^{(0)}\big) \in \mathbb{R}^{|\mathcal{P}|}$ through $\mathcal{R}^{s,G}$, where $H^{(i)}$ are the pooled object-embeddings at decode step i. The worker instead scores its schemas after a first decoding pass, over a graph that adds a node for each schema it can select. Afterwards, we add the selected predicate p as an explicit node to the graph, alongside a decision node c, and a slot node q_j for each argument, indexed in decoding order. At decode step i, after having selected predicate p and objects $o_{<i}$, we form the relation set

$$\begin{aligned}\mathcal{R}^{(i)} = \mathcal{R}^{s,G} &\cup \{\mathrm{symbol}(c, p),\, \mathrm{focus}(c, q_i)\} \cup \{\mathrm{slot}(p, q_j) \mid 1 \le j \le k\} \\ &\cup \{\mathrm{assigned}^{\leftrightarrow}(q_j, o_j) \mid j < i\} \cup \{\mathrm{eligible}^{\leftrightarrow}(o, q_i) \mid o \in E_i\}.\end{aligned} \tag{5}$$

The symbol and slot relations connect the selected predicate to its argument positions, and focus identifies the argument currently to choose. The last two relations connect the current slot to its

eligible objects E_i, defined below. Assignment and eligibility relations are forced symmetric by emplacing both directions, denoted by the superscript $\leftrightarrow$. At argument step i, the score of an eligible object o is

$$\ell_i(o) = \mathrm{MLP}_{\arg}\Big(h_o^{(i)} \parallel h_c^{(i)} \parallel h_{q_i}^{(i)} \parallel \sum_{j=1}^{k} h_{q_j}^{(i)} \parallel H^{(i)}\Big). \tag{6}$$

All inputs are taken after the current decoding pass, so the score depends on the prefix encoded in $\mathcal{R}^{(i)}$.

For both decision-making levels we provide two R-GNN parameter sets. An encoder network applies the updates of Section 3 to $\mathcal{R}^{s,G}$ once at the beginning of the decision. We denote its final object embeddings by $\{h_o^{\mathrm{s}} \mid o \in \mathcal{O}\}$. A second network processes the decision graph at each argument step, and the worker also runs it at the schema step. The manager's predicate classifier reads the encoder's output, $H^{(0)} = \sum_{o\in\mathcal{O}} h_o^{\mathrm{s}}$. At the worker's schema step, the graph contains $\mathcal{R}^{s,G}$, the decision node c and a node n_A for each schema A the worker can select, connected to c by the symbol relation. This extra decoding step for the worker is useful, because only schemas whose preconditions without parameters hold and whose first argument has an eligible object can be selected (Appendix C.2). Such a choice benefits from representation in the graph; predicates have no conditions that must hold, thus the manager's first decode step may use embeddings from $\mathcal{R}^{s,G}$ directly. The score of schema A is computed from $h_{n_A}^{(0)}$, the initial embedding of n_A and the pooled object embeddings $H^{(0)}$ of this pass.

At the first decoding step, object embeddings are initialised with the encoder's output, and the added nodes receive learned initial embeddings. We choose such over unary relations when they are not required 'to reappear' during message passing, as learned embeddings are a smaller parameter set and also decrease message aggregation pressure that may otherwise compete with more important signal. Initial node embeddings $e_p, e_{\mathrm{ar}(p)}$ of the chosen predicate and its arity are emplaced in the respective node, while the slot nodes receive embeddings marking their position in the predicate. In the manager, each slot's initial embedding also includes its argument type. The initial embedding of the decision node identifies the decoding step and the current slot. Each added node's initial embedding is a linear map of its concatenated learned embeddings, followed by $\tanh$. A nullary atom $P()$ is represented as $P(c)$, where c is the decision node. This gives the atom a node to which it can send a message. The encoder attaches nullary atoms to an auxiliary node in the same way.

We write $h_v^{(i)}$ for the embedding of node v after message passing at step i, with $i = 0$ denoting the worker's schema step. At later steps, existing nodes are initialised with the embeddings computed at the previous step, while new nodes receive learned initial embeddings again. We rescale reused embeddings by a common factor to keep their mean norm fixed. An existing node v thus starts step $i+1$ from $\alpha_i h_v^{(i)}$ with $\alpha_i = \sqrt{d}\,|\mathcal{K}_i| / \sum_{u\in\mathcal{K}_i} \|h_u^{(i)}\|_2$, where $\mathcal{K}_i$ contains the nodes with nonzero embeddings, so that their mean norm is $\sqrt{d}$. The readouts use the embeddings before this rescaling.

Networks. Both encoders have 30 layers of width 64, with smooth-maximum message aggregation at temperature 12. The manager's decoding network has 10 layers and the worker's has 30. Besides the atoms of Eq. (4), the input contains unary atoms that state the types of each object. Every relation has its own message function MLP_P. This includes the goal predicates $P_{G,T}$ and $P_{G,F}$ and, in the decoding network, every relation of the decision graph, where the two directions of the assignment and eligibility relations count as separate relations. For an atom of arity r, MLP_P maps the concatenated argument embeddings in $\mathbb{R}^{rd}$ through one hidden layer of $2d$ units with SiLU activation back to $\mathbb{R}^{rd}$. The update function MLP_U maps $h_o \| m_o \in \mathbb{R}^{2d}$ through one hidden layer of $2d$ units with Mish activation to $\mathbb{R}^d$. No layer uses normalisation or dropout. In the implementation, each message also adds the embedding of the object it is sent to, which shifts the smooth maximum of that object's messages by h_o and only reparametrises the first layer of MLP_U.

Readout heads. The manager's heads apply layer normalisation to their input, followed by two hidden layers of twice the input width with SiLU activation and a linear output. In the argument score of Eq. (6), the manager uses the mean of the slot embeddings instead of their sum. The worker's readouts differ, because its value head predicts the number of actions needed to achieve the chosen subgoal, which does not grow with the size of the instance. A sum of object embeddings grows with the number of objects independently of this value, so the worker averages the object embeddings of a decoding pass instead. A residual MLP with layer normalisation and one hidden

layer of 128 units maps this average to the pooled embedding that all heads of the worker share. Its schema and argument scores apply layer normalisation and one hidden layer of 128 units with SiLU activation. The worker's argument score reads $h_o^{(i)}$, $h_{q_i}^{(i)}$ and the pooled embedding. The worker's value and applicability heads are linear, the first on the pooled embedding of the schema step and the second on the embedding of the schema node after all arguments are chosen.

C.2 Precondition Checks

Let $T_{A,i}$ be the objects allowed by the type of the parameter chosen at step i, and let $\mathrm{pre}_i(A)$ contain the precondition literals whose last unassigned parameter is this parameter. The worker's eligible set is then

$$E_i = T_{A,i} \cap \bigcap_{\ell \in \mathrm{pre}_i(A)} \big\{ o \mid \ell[o_{<i}, o] \text{ holds in } s \big\}, \tag{7}$$

where $\ell[o_{<i}, o]$ substitutes the selected objects and candidate o into the literal ℓ. A positive literal holds if the resulting atom is in s, and a negative literal if it is absent. The checks use an index over the atoms of the state, built once per instance for static atoms and once per state for fluent atoms. It groups atoms by predicate and maps the objects already assigned in a literal to the objects that complete it to an atom of the state, so no ground action is enumerated. Numeric and quantified preconditions are not supported and do not occur in the ten domains.

Each literal is checked as soon as its last parameter is bound, so the decoding order determines how early infeasible bindings can be pruned. We fix this order per schema, rather than per state, to ensure consistent teacher-forcing tokenization. The order is computed greedily from the lifted precondition: parameters appearing in non-type unary literals are decoded first; thereafter, we repeatedly select the parameter whose binding completes the largest number of precondition literals, breaking ties by declaration order. For example, for `board(car, loc)` in ferry, `loc` is decoded first because `at-ferry(loc)` determines the ferry's current location. Then `at(car, loc)` restricts `car` to vehicles at that location. Under the declared parameter order, by contrast, all cars would initially be eligible. If a prefix has no eligible next object, decoding discards that prefix and resumes from the schema choice within the retry budget.

Singleton widening. During training, if the precondition checks leave exactly one eligible object, we expand the choice to all objects of the required type, so the choice still gives a cross-entropy gradient. Inapplicable actions proposed this way enter the second term of Eq. (8), and deployment uses the strict eligible sets. In the recorded measurements, 66–84 % of slots in miconic, 41 % in transport and 20 % in spanner had a single eligible object. The miconic filter often identifies the lift's floor through the unary lift-at precondition. Without widening, these choices receive zero cross-entropy gradient. A worker trained with the strict filter solved 14 of 150 hard miconic instances, compared with 148 of 150 for one trained with type restrictions alone. These results motivated retaining a learning signal for choices determined by the filter.

C.3 Achiever Relations

The precondition checks determine which actions the worker can execute, but do not inform of the effects. Emplacing all possible ground effects that can still be acquired at any decode step would add a large relation set of potentially little value. Instead, we give the worker information about which continuation may achieve the subgoal. An applicable action achieves g if g is one of its add-effects. To find such actions, we match g to the add effects of each schema A. This binds the parameters to the objects that occur in the matched effect. We then assign the remaining parameters, checking the preconditions against the current state. Parameters with the fewest eligible objects are assigned first, a failed precondition causes backtracking, and each check explores at most 2048 partial assignments and retains at most 64 applicable actions. We record the applicable actions found this way in the decision graph through two relations. At the schema step, $\mathrm{achiever}(n_A, o)$ connects the node of schema A to each object o fixed by matching its effect to g. At argument step i, we add $\mathrm{achiever}(o, q_i)$ when one of these actions agrees with the prefix and uses o as its next argument. The network can thus use both the effects of a schema and the argument choices that make it achieve g. The eligible sets E_i remain unchanged.

A check ends in one of three ways. If it finds applicable actions, they give the $\mathrm{achiever}$ relations above. If it reaches its limit, a separate relation connects the schema node to the fixed objects instead. At argument steps, $\mathrm{achiever}(o, q_i)$ is then added for each eligible object o that satisfies every precondition whose other parameters are fixed by the effect match and the prefix, so these objects need not have a jointly applicable completion. If it finishes without finding an applicable action, a third relation connects the schema node to the fixed objects, and we apply one step of regression.

In this step, we examine the missing positive fluent preconditions of the schemas checked and apply the same check to each of them, with the objects already fixed by the first check retained. For example, communicating soil data from waypoint w requires $\mathrm{have_soil_analysis}(r, w)$ for some rover r. If that atom is false, the check can identify an applicable $\mathrm{sample_soil}$ action at w. We add the same $\mathrm{achiever}$ relations for these actions, and $\mathrm{missing}_{P,j}(o)$ for each fixed object o at position j of the missing precondition P. In the example, $\mathrm{missing}_{\mathrm{have_soil_analysis},2}(w)$ records which waypoint's sample is needed. We consider all missing preconditions of a check, such as both $\mathrm{holding}(a)$ and $\mathrm{clear}(b)$ for $\mathrm{stack}(a, b)$, except those with no fixed object or static restriction. We prioritise those for which an applicable achiever is found, retaining at most four after a bounded scan of at most sixteen checks. The step is repeated for a missing precondition without an applicable achiever, up to a fixed depth, and the reported worker uses depth 2. Table 3 lists which applicable actions are marked at each depth, and Table 7 in Appendix E compares depths 0, 1 and 2.

Table 3: Regression depth of the achiever relations for a subgoal g. A deeper level is used only when the previous one marks no action. An achiever can lack several preconditions, so an action marked at depth d is followed by at least d further actions before g holds.

Depth	Applicable actions marked as achievers	Actions until g holds
0	actions that make g true	1
1	if there are none, actions that establish a missing precondition of such an action	≥ 2
2	if there are still none, actions that establish a missing precondition of that precondition	≥ 3

C.4 Learning and Decoding Settings

Manager rewards. Every chosen subgoal has reward -1. Training rollouts use no closed set, and LIW(1) runs without a limit on actions, time or expanded states, so a subgoal fails only when LIW(1) finds no plan for it. The state then does not change, and the atom is excluded at that state. A transition into a dead end, a state without applicable actions, adds -1000. There is no success bonus, and $\gamma = 0.999$ per subgoal. Rollouts end at the task goal, at a dead end, after 100 subgoals, at the third consecutive failed subgoal, or when no subgoal can be decoded.

Manager data. Each of the up to 400 training episodes collects 20 rollouts and then performs 32 optimisation steps on batches of 32 transitions. Rollouts sample subgoals at a temperature that decreases linearly from 1 towards 0.1 over 500 episodes. Training instances enter in blocks of ten, ordered by objects and goal atoms, and the next block is added when greedy decoding solves 80 % of the active instances. Once 100 start states are stored, half of the rollouts start from a stored state instead of an initial state, 30 % of these from a chained IW(1) exploration walk and the rest from states of earlier rollouts. Three buffers hold the transitions of solved rollouts (at most 2000, kept across episodes), of the unsolved rollouts of the current episode (emptied after each update), and of solved rollouts on instances with a smoothed solve rate of at most 0.1 (at most 500). A batch draws 26, 5 and 1 transitions from them with prioritised replay (Schaul et al., 2016), using each transition's loss as its priority, exponent 0.6 and an importance correction annealed from 0.4 to 1. Within the first and third buffer, half of each quota is drawn uniformly over four intervals of normalised distance and used by the critic only. The manager receives no planner supervision, such as plan repair or DAgger, and its subgoals are not relabelled.

Manager actor and distance head. Subgoals with non-positive advantage are kept only from rollouts that solved their instance, so the manager retains its behaviour on instances it already solves. In Eq. (3), $\beta = 1$ and $w_{\max} = 10$, and the advantage A is multiplied by 15, since the normalised

distances make the progress of one subgoal small. The weights of subgoals with non-positive advantage are scaled to sum to at most half the number of achieved subgoals in the batch. All weights are renormalised over the transitions available to the actor and multiplied by the replay's importance weights, and the loss has no entropy term. The distance head reads the pooled encoder output and is fitted with the Huber loss to labels from solved rollouts. The subgoal at position t of a rollout with T subgoals, counted from zero, is labelled $T - t$, reduced to the shortest remaining count observed for the same instance, state, goal and subgoal, and divided by the largest of the instance's last 512 labels. The head's gradient trains the shared network, and the advantage uses its predictions without gradient.

Manager critic. A critic estimates token values in the dueling form (Wang et al., 2016) and learns them by temporal differences. It shares the network with the actor, so its loss also trains the shared embeddings, but its values are used neither in Eq. (3) nor to select subgoals, since they proved unstable. At token step i, let $\mathcal{T}_i$ be the allowed tokens, V_i the value prediction for the current prefix, and $A_i(t)$ the predicted advantage of token t. The token value has the dueling form $Q_i(t) = V_i + A_i(t) - |\mathcal{T}_i|^{-1} \sum_{u \in \mathcal{T}_i} A_i(u)$ (Wang et al., 2016), and a subgoal generated by the tokens $t_0, \ldots, t_k$ has value $\bar{Q} = (k+1)^{-1} \sum_{i=0}^{k} Q_i(t_i)$. The critic uses separate heads on the same features as the actor. The target of token $i < k$ is the target network's value of the next token that the online critic ranks highest (van Hasselt et al., 2016). The target of the last token is $\max(G_t^\lambda, F)$, where $G_t^\lambda = r_t + \gamma\big[(1-\lambda)\bar{Q}^-_{t+1} + \lambda G^\lambda_{t+1}\big]$ is a λ-return with $\lambda = 0.5$ over the stored rollout, r_t is the reward, and $\bar{Q}^-_{t+1}$ is the target network's value of the rollout's next subgoal. This value is zero at the task goal, the maximum over candidate subgoals after the 100-subgoal limit, and -10^4 after a dead end. A rollout that ends at the third consecutive failed subgoal is not bootstrapped. The floor F is the best discounted return observed from the same instance, state, goal and subgoal, once one has been recorded. Each token value is fitted to its target with the Huber loss, and a second Huber term penalises values below F. A ranking term with weight 0.5 requires the value of an achieved subgoal to exceed that of each failed subgoal from the same state in the batch by a margin of 0.1. The target network is copied every 32 optimisation steps.

Worker search and examples. From each start state, eight goal-free search runs with different tie-breaking record plans for all landmark atoms that are false in the state, including members of disjunctive landmarks. The shortest plan found for an atom supplies its examples, and its length counts as exact when all eight runs complete and reach the atom at that depth. The breadth-first search from the start state is limited to 300 000 states and to the depth of the longest plan, at most 16, and atoms it does not resolve keep only the labels of these runs. Start states are the initial states of the training instances, states of a chained goal-free IW(1) exploration walk, the first state in which some atom holds along IW(1) rollouts from the initial state, and states of earlier search trees. They also include the first four states of the worker's own attempts, searched with the attempt's visited states excluded, and states on a plan for the task goal found by greedy best-first search with h^{FF}, whose actions are not imitated. None of these start states comes from the manager or the validation instances. These runs and the validation tasks below use IW(2) for every reported worker, including the GABAR worker (Section 4.2). Its traces are not guaranteed to subsume the plans that LIW(1) finds, although empirically we observed this to hold in every test over the IPC benchmark.

Worker actor. At the start state of the search, $\mathcal{A}^*(s, g)$ contains the first actions of the shortest plans found, and a bounded breadth-first search identifies the applicable actions $\mathcal{N}(s, g)$ that only start longer plans. The worker increases the total probability of $\mathcal{A}^*(s, g)$ and decreases that of $\mathcal{N}(s, g)$,

$$\mathcal{L}_{\mathrm{set}} = -\log \sum_{a \in \mathcal{A}^*(s,g)} \pi_W(a \mid s, g) + \eta \sum_{a \in \mathcal{N}(s,g)} \big[\log \pi_W(a \mid s, g) - \log \epsilon\big]_+, \tag{8}$$

so any optimal first action may be chosen, and a worse action is only pushed down until its probability falls below ϵ. In Eq. (8), each action's log-probability is divided by its number of tokens $|a|$. The first term is implemented as a cross-entropy on each action $a \in \mathcal{A}^*$, weighted by $\pi_W(a \mid s, g) / \sum_{b \in \mathcal{A}^*} \pi_W(b \mid s, g)$ and with this weight held fixed when differentiating. Its gradient equals that of $-|a|^{-1} \log \sum_{a \in \mathcal{A}^*} \pi_W(a \mid s, g)$ when all actions in $\mathcal{A}^*$ have the same number of tokens. This term has weight 1 when $\mathcal{N}$ is non-empty. At later plan states, and at start states with an empty $\mathcal{N}$, it has the weight of Eq. (3) with $A = y - V(s, g)$, $\beta = 1$ and $w_{\max} = 20$. The negative term uses $\eta = 0.1$ and $\epsilon = 0.01$. On childsnack, satellite and transport, at most 16 actions of $\mathcal{N}$

are drawn per example, anew at every step, and scaled by $|\mathcal{N}|/16$. For a quarter of each batch, the worker samples one action at temperature 1, and inapplicable samples, which can only come from widened slots, enter the negative term. An applicability head, trained with binary cross-entropy and weight 0.1, labels the actions of $\mathcal{A}^*$ and $\mathcal{N}$ as applicable and the inapplicable samples as not. It is not used for decoding.

Worker value head. The advantage in this weight comes from a value head $V(s, g)$, trained on the returns of these transitions with a cost of 1 per action. The target of a transition (s, a, s') is bootstrapped from a target network $\bar{V}$ and bounded by the known plan, $y = \max\big(-d,\ -1+\gamma\,\bar{V}(s', g)\big)$, where d is the length of the known plan from s through a, following Aichmüller et al. (2026a), and $A = y - V(s, g)$. The other actions applicable along a plan provide transitions without a known plan, which train only the value head. The value head's gradient trains both networks. Its targets use $\gamma = 0.999$ and a target network copied every 32 steps. Transitions that achieve g have target -1. Where repeated searches from a start state agree on the plan length ℓ, $-\ell$ itself is the target. A transition into a state without applicable actions has target -10^4, which is checked for the first 64 leaves of one run per start state. The value loss is the Huber loss with weight 0.5. In an earlier configuration, training without the value head, which removes the advantage weight and the value loss, diverged in 11 of 79 runs against 0 of 60 with it.

Decoding settings of the reported passes. The manager's beam ranks prefixes by the cumulative log-softmax of their token scores, checks that the completed atom is currently false, and falls back to greedy decoding. Subgoals found unreachable and subgoals blocked by the closed set before any action are excluded at the state where this happened, except in floortile, by removing them from the trie of candidate atoms. The exclusions are cleared when the state changes and relaxed if they would remove every candidate. A run ends after 100 consecutive subgoals blocked before any action. The worker's beam has width 8 per schema and excludes actions leading to a closed state.

C.5 Experimental Settings

Training schedules. The manager's optimiser is Adam with gradient norms clipped at 5 and a cosine learning-rate schedule from 10^{-4} to 10^{-5} over 9 600 steps, that is 300 episodes of 32 steps. The learning rate then stays at 10^{-5}. Each worker epoch first adds new search trees, in batches of 16 start states until 64 trees or 10 seconds are reached. The worker then makes up to 16 attempts on atomic tasks within 10 seconds, with at most 15 steps each and half of them greedy, and the states they visit become start states for later searches. Finally, it performs 64 optimisation steps on batches of 128 transitions. The buffer keeps up to 2 048 trees for at most 8 epochs and at most 512 transitions per instance and predicate. Batches are drawn uniformly over groups of transitions with the same predicate, distance, instance size, kind of example and start-state source. Adam uses a constant learning rate of 10^{-3} without gradient clipping.

Worker checkpoint selection. For validation, we construct a fixed set of 256 atomic tasks per domain from the same searches on the 30 validation instances. Each task specifies a state and a landmark atom reached by search, together with the length of the plan found. Worker checkpoints are evaluated on these tasks every fifth epoch, with action budgets equal to the recorded plan length and twice that length. We rank them by the validation success rate, with ties broken by the mean ratio of executed actions to recorded plan length on the 32 validation tasks evaluated at each epoch.

Manager checkpoint ranking. Both selection stages decode with the beams of the test passes and the closed set. On the selection instances, LIW(1) and the frozen learned worker run in parallel for each subgoal, and the first to achieve the atom is kept. Each instance has limits of 1 200 s, 500 manager decisions and 1 000 primitive actions, the last checked between subgoals, and the learned worker executes at most 50 actions per subgoal. The first stage evaluates every saved checkpoint on the 30 validation instances. After the first five checkpoints of a run, an evaluation stops as soon as the checkpoint can no longer reach the coverage of the fifth best checkpoint of the same run. Checkpoints solving all 30 validation instances proceed to the selection instances, evaluated in the order easy, medium, hard. A timeout skips the remaining larger tiers, and the evaluation of a checkpoint stops after two timeouts.

Checkpoints are ranked first by validation coverage, then by the number of validation instances tested, and then by the numbers of hard and medium selection instances solved. Remaining ties favour fewer primitive actions and fewer manager decisions on the validation instances, and then

a later training episode. In spanner and sokoban, no checkpoint solves all validation instances, so validation coverage alone ranks them.

Flat GABAR. Of the 677 training instances, 78 did not allow an optimal plan to be found within generous budgets, and these are the largest of each domain. We opt to exclude them instead of producing suboptimal plans to remain consistent with the method's original methodology. The policy decodes greedily with a closed set of visited states and parses the test instances without grounding. Each instance allows one hour and 20 000 actions; the published GABAR experiments use a limit of 1 000 steps (Mangannavar et al., 2025). Our policy solves 206 test instances, including 186 within 200 actions. We test the checkpoint of the last epoch, and in floortile, whose training reached its time limit, that of epoch 580.

Test outcomes. Execution continues until the task goal is reached, a dead end is detected, or a resource limit is reached. We record failures due to the action, decision, time or memory limit, and distinguish them from dead ends and closed-set blocking. With LIW(1) as executor, each plan found is executed in full. If search fails without exhausting a resource budget, we repeat it without the closed set. Success on this second search identifies a subproblem blocked by the visited-state restriction. Otherwise, we record a search failure, which does not prove that the atom is unreachable.

Hardware. Training and testing ran on a cluster whose GPU nodes have four NVIDIA A10 GPUs (24 GB) or four NVIDIA L40S GPUs (48 GB), shared between processes, and on two machines with two NVIDIA RTX PRO 6000 GPUs (96 GB) each. Each test run with the learned worker had 9 CPU cores and 24 GB of memory. LIW(1) needs no GPU, and its test runs shared one AMD EPYC 9655 node, 180 at a time.

D Planning Instances

D.1 IPC 2023 Instances

Table 4 gives the number and size of the instances in our split of the official training set and in the three test tiers. Blocksworld grows from at most 20 objects in training to 488 in testing and sokoban from 124 to 9 880, while ferry's goals grow from at most 13 to 974 atoms.

Table 4: IPC 2023 instance sizes. Training and validation columns give instance counts and object ranges for our split of the official training set. Test columns give object and goal-atom ranges; each tier contains 30 instances per domain.

	Train		Validation		Test easy		Test medium		Test hard	
Domain	#	Objects	#	Objects	Objects	Goal atoms	Objects	Goal atoms	Objects	Goal atoms
blocksworld	69	2–20	30	21–29	5–29	6–33	35–146	39–158	160–488	179–537
childsnack	68	6–40	30	40–51	20–51	4–10	65–187	15–40	207–1326	50–292
ferry	69	3–25	30	26–35	7–35	2–20	30–146	10–97	300–1461	200–974
floortile	67	5–40	30	37–45	15–37	9–24	116–283	100–247	667–1022	625–952
miconic	69	3–21	30	22–30	5–30	1–10	50–137	20–78	130–681	50–485
rovers	69	10–28	30	28–36	12–36	2–36	49–249	37–325	294–596	38–728
satellite	69	5–32	30	34–43	11–43	2–29	48–153	9–141	145–402	44–840
sokoban	69	50–124	30	124–173	65–173	1–4	405–2435	5–34	3640–9880	40–79
spanner	59	6–22	30	22–28	9–28	1–5	63–184	15–49	203–833	50–244
transport	69	6–34	30	34–45	12–39	1–15	40–109	5–44	111–354	20–194

D.2 Checkpoint-Selection Instances

Many manager checkpoints solve all 30 validation instances, so validation alone cannot rank them. Table 5 gives the object and goal-atom ranges of the training, validation and checkpoint-selection instances. We generate ten selection instances at each test tier's scale using the official IPC 2023 generators with new random seeds. Parameters are sampled uniformly along the range used in that tier, rounded and kept within its bounds, then adjusted to satisfy the generator's constraints. Instances that duplicate any training, validation, test or earlier selection instance in a canonical representation, which ignores formatting, are discarded and generated again with a new seed.

Table 5: Object and goal-atom ranges of the training and validation instances and of the checkpoint-selection instances, ten per test tier and domain.

	Train		Validation		Selection easy		Selection medium		Selection hard	
Domain	Objects	Goal atoms	Objects	Goal atoms	Objects	Goal atoms	Objects	Goal atoms	Objects	Goal atoms
blocksworld	2–20	3–27	21–29	24–34	6–28	7–35	84–140	91–156	218–463	241–519
childsnack	6–40	1–8	40–51	8–10	20–51	4–10	75–164	17–35	517–1193	117–263
ferry	3–25	1–13	26–35	14–20	9–31	3–17	54–146	28–97	441–1144	294–763
floortile	5–40	1–27	37–45	28–30	18–41	12–30	138–283	120–255	693–1015	650–957
miconic	3–21	1–7	22–30	7–10	6–29	1–10	75–132	37–75	221–462	122–312
rovers	10–28	1–27	28–36	3–32	20–36	6–35	54–216	14–231	296–574	72–501
satellite	5–32	1–16	34–43	4–29	12–43	1–34	56–145	20–158	171–358	73–553
sokoban	50–124	1–3	124–173	3–4	65–173	1–4	447–2147	6–31	4270–8170	45–70
spanner	6–22	1–4	22–28	4–5	13–26	2–5	64–175	15–47	203–758	50–221
transport	6–34	1–12	34–45	12–17	12–37	1–14	40–109	5–44	141–273	42–136

Table 6: Coverage per domain and tier, with the columns of Table 1. Distincter and WL-GOOSE report no results per tier. Numbers in parentheses are percentages of row totals, and the highest coverage of each row is bold.

	Landmarker			Flat policies			Systematic search		
Domain	π_W	LIW(1)	GABAR	GABAR	Flat–AA	AIW–AD	LAMA	Distincter	WL-GOOSE
blocksworld	**90 (100)**	47 (52)	0 (0)	11 (12)	76 (84)	**90 (100)**	57 (63)	88 (98)	75 (83)
easy (30)	**30 (100)**	**30 (100)**	0 (0)	11 (37)	**30 (100)**	**30 (100)**	**30 (100)**	n/a	n/a
medium (30)	**30 (100)**	17 (57)	0 (0)	0 (0)	**30 (100)**	**30 (100)**	24 (80)	n/a	n/a
hard (30)	**30 (100)**	0 (0)	0 (0)	0 (0)	16 (53)	**30 (100)**	3 (10)	n/a	n/a
miconic	**90 (100)**	74 (82)	68 (76)	36 (40)	84 (93)	**90 (100)**	**90 (100)**	**90 (100)**	**90 (100)**
easy (30)	**30 (100)**	**30 (100)**	**30 (100)**	**30 (100)**	**30 (100)**	**30 (100)**	**30 (100)**	n/a	n/a
medium (30)	**30 (100)**	**30 (100)**	**30 (100)**	6 (20)	**30 (100)**	**30 (100)**	**30 (100)**	n/a	n/a
hard (30)	**30 (100)**	14 (47)	8 (27)	0 (0)	24 (80)	**30 (100)**	**30 (100)**	n/a	n/a
spanner	30 (33)	29 (32)	29 (32)	30 (33)	**90 (100)**	**90 (100)**	30 (33)	**90 (100)**	73 (81)
easy (30)	29 (97)	29 (97)	29 (97)	**30 (100)**	**30 (100)**	**30 (100)**	**30 (100)**	n/a	n/a
medium (30)	1 (3)	0 (0)	0 (0)	0 (0)	**30 (100)**	**30 (100)**	0 (0)	n/a	n/a
hard (30)	0 (0)	0 (0)	0 (0)	0 (0)	**30 (100)**	**30 (100)**	0 (0)	n/a	n/a
satellite	**90 (100)**	45 (50)	66 (73)	16 (18)	51 (57)	88 (98)	**90 (100)**	48 (53)	53 (59)
easy (30)	**30 (100)**	**30 (100)**	**30 (100)**	16 (53)	**30 (100)**	**30 (100)**	**30 (100)**	n/a	n/a
medium (30)	**30 (100)**	15 (50)	25 (83)	0 (0)	21 (70)	**30 (100)**	**30 (100)**	n/a	n/a
hard (30)	**30 (100)**	0 (0)	11 (37)	0 (0)	0 (0)	28 (93)	**30 (100)**	n/a	n/a
transport	**90 (100)**	47 (52)	42 (47)	13 (14)	74 (82)	86 (96)	72 (80)	50 (56)	29 (32)
easy (30)	**30 (100)**	**30 (100)**	**30 (100)**	13 (43)	29 (97)	**30 (100)**	**30 (100)**	n/a	n/a
medium (30)	**30 (100)**	17 (57)	12 (40)	0 (0)	25 (83)	29 (97)	**30 (100)**	n/a	n/a
hard (30)	**30 (100)**	0 (0)	0 (0)	0 (0)	20 (67)	27 (90)	12 (40)	n/a	n/a
ferry	**90 (100)**	60 (67)	61 (68)	55 (61)	62 (69)	80 (89)	74 (82)	83 (92)	76 (84)
easy (30)	**30 (100)**	**30 (100)**	**30 (100)**	**30 (100)**	29 (97)	**30 (100)**	**30 (100)**	n/a	n/a
medium (30)	**30 (100)**	**30 (100)**	24 (80)	25 (83)	29 (97)	**30 (100)**	**30 (100)**	n/a	n/a
hard (30)	**30 (100)**	0 (0)	7 (23)	0 (0)	4 (13)	20 (67)	14 (47)	n/a	n/a
rovers	54 (60)	43 (48)	30 (33)	3 (3)	27 (30)	67 (74)	**78 (87)**	42 (47)	37 (41)
easy (30)	**30 (100)**	**30 (100)**	28 (93)	3 (10)	27 (90)	**30 (100)**	**30 (100)**	n/a	n/a
medium (30)	23 (77)	13 (43)	2 (7)	0 (0)	0 (0)	24 (80)	**30 (100)**	n/a	n/a
hard (30)	1 (3)	0 (0)	0 (0)	0 (0)	0 (0)	13 (43)	**18 (60)**	n/a	n/a
childsnack	**67 (74)**	36 (40)	33 (37)	30 (33)	41 (46)	64 (71)	36 (40)	64 (71)	29 (32)
easy (30)	**30 (100)**	**30 (100)**	27 (90)	28 (93)	27 (90)	**30 (100)**	**30 (100)**	n/a	n/a
medium (30)	**30 (100)**	6 (20)	5 (17)	2 (7)	12 (40)	**30 (100)**	5 (17)	n/a	n/a
hard (30)	**7 (23)**	0 (0)	1 (3)	0 (0)	2 (7)	4 (13)	1 (3)	n/a	n/a
sokoban	4 (4)	36 (40)	0 (0)	2 (2)	14 (16)	13 (14)	**44 (49)**	32 (36)	38 (42)
easy (30)	4 (13)	29 (97)	0 (0)	2 (7)	14 (47)	13 (43)	**30 (100)**	n/a	n/a
medium (30)	0 (0)	7 (23)	0 (0)	0 (0)	0 (0)	0 (0)	**14 (47)**	n/a	n/a
hard (30)	**0 (0)**	**0 (0)**	**0 (0)**	**0 (0)**	**0 (0)**	**0 (0)**	**0 (0)**	n/a	n/a
floortile	43 (48)	**57 (63)**	0 (0)	10 (11)	0 (0)	0 (0)	15 (17)	2 (2)	2 (2)
easy (30)	**30 (100)**	**30 (100)**	0 (0)	10 (33)	0 (0)	0 (0)	15 (50)	n/a	n/a
medium (30)	13 (43)	**27 (90)**	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	n/a	n/a
hard (30)	**0 (0)**	**0 (0)**	**0 (0)**	**0 (0)**	**0 (0)**	**0 (0)**	**0 (0)**	n/a	n/a
Total (900)	648 (72)	474 (53)	329 (37)	206 (23)	519 (58)	**668 (74)**	586 (65)	589 (65)	502 (56)

E Full Results

Tested pairings. Table 8 gives, per domain, the number of tested manager–worker pairings and the best, mean, standard deviation and range of their coverage, for the learned worker and for LIW(1).

Table 7: Coverage of Landmarker with workers whose achiever relations regress through missing preconditions up to depth 0, 1 and 2 (Appendix C.3), under the same five manager checkpoints. Depth 2 is the worker of Table 1. Each entry is the pairing with the highest coverage, as in Table 1. Numbers in parentheses are percentages of row totals, and the highest coverage of each row is bold.

Domain	Depth 0	Depth 1	Depth 2
blocksworld	**90 (100)**	86 (96)	**90 (100)**
easy (30)	**30 (100)**	**30 (100)**	**30 (100)**
medium (30)	**30 (100)**	**30 (100)**	**30 (100)**
hard (30)	**30 (100)**	26 (87)	**30 (100)**
miconic	**90 (100)**	**90 (100)**	**90 (100)**
easy (30)	**30 (100)**	**30 (100)**	**30 (100)**
medium (30)	**30 (100)**	**30 (100)**	**30 (100)**
hard (30)	**30 (100)**	**30 (100)**	**30 (100)**
spanner	27 (30)	29 (32)	**30 (33)**
easy (30)	27 (90)	**29 (97)**	**29 (97)**
medium (30)	0 (0)	0 (0)	**1 (3)**
hard (30)	**0 (0)**	**0 (0)**	**0 (0)**
satellite	63 (70)	**90 (100)**	**90 (100)**
easy (30)	**30 (100)**	**30 (100)**	**30 (100)**
medium (30)	29 (97)	**30 (100)**	**30 (100)**
hard (30)	4 (13)	**30 (100)**	**30 (100)**
transport	56 (62)	61 (68)	**90 (100)**
easy (30)	**30 (100)**	**30 (100)**	**30 (100)**
medium (30)	21 (70)	28 (93)	**30 (100)**
hard (30)	5 (17)	3 (10)	**30 (100)**
ferry	59 (66)	**90 (100)**	**90 (100)**
easy (30)	**30 (100)**	**30 (100)**	**30 (100)**
medium (30)	29 (97)	**30 (100)**	**30 (100)**
hard (30)	0 (0)	**30 (100)**	**30 (100)**
rovers	32 (36)	30 (33)	**54 (60)**
easy (30)	17 (57)	18 (60)	**30 (100)**
medium (30)	15 (50)	12 (40)	**23 (77)**
hard (30)	0 (0)	0 (0)	**1 (3)**
childsnack	57 (63)	61 (68)	**67 (74)**
easy (30)	**30 (100)**	**30 (100)**	**30 (100)**
medium (30)	27 (90)	28 (93)	**30 (100)**
hard (30)	0 (0)	3 (10)	**7 (23)**
sokoban	**7 (8)**	3 (3)	4 (4)
easy (30)	**7 (23)**	3 (10)	4 (13)
medium (30)	**0 (0)**	**0 (0)**	**0 (0)**
hard (30)	**0 (0)**	**0 (0)**	**0 (0)**
floortile	33 (37)	38 (42)	**43 (48)**
easy (30)	**30 (100)**	**30 (100)**	**30 (100)**
medium (30)	3 (10)	8 (27)	**13 (43)**
hard (30)	**0 (0)**	**0 (0)**	**0 (0)**
Total (900)	514 (57)	578 (64)	**648 (72)**

Over the ten domains, the best pairings of the learned worker sum to 648, their means to 631 and their minima to 608. The spread comes from transport, rovers, childsnack and floortile. In the other six domains, every pairing is within one instance of the best. For LIW(1), the sums are 474, 462 and 446.

Spanner dead ends. In spanner, Landmarker solves almost no instance beyond the easy tier. Every unsolved instance ends in a dead end, since the man walks past a spanner that a remaining nut needs. Two causes can be identified. The advantage that weights the actor comes from the distance head, which is fitted on solved rollouts, so a subgoal that enters a dead end is pushed down only indirectly, by the alternatives the advantage prefers. Moreover, whether the carried spanners suffice for the remaining nuts compares two counts, which the network does not extrapolate from at most four nuts in training to 244. Table 9 reports two managers that address these causes, trained as the reported manager otherwise and tested with the same worker. Dead ends are known exactly in spanner from an enumeration of the state spaces of the training instances, which have at most 500 000 states. The dead-end advantage pins the predicted distance of a state that enters one to the rollout horizon, so that the subgoal entering it receives a negative advantage in Eq. (3). This raises coverage from 30 to 40. The sufficiency relation adds one of three derived nullary atoms to the state encoding, computed

Table 8: Coverage over the tested pairings of the five selected manager checkpoints with worker checkpoints, per domain and out of 90. Mean, standard deviation and range are over the pairings whose 90 runs finished.

	π_W				LIW(1)			
Domain	Pairings	Best	Mean ± SD	Range	Pairings	Best	Mean ± SD	Range
blocksworld	5	90	90.0 ± 0.0	90–90	5	47	45.2 ± 1.3	44–47
miconic	5	90	90.0 ± 0.0	90–90	5	74	73.2 ± 0.8	72–74
spanner	5	30	29.0 ± 1.2	27–30	5	29	28.6 ± 0.9	27–29
satellite	5	90	89.8 ± 0.4	89–90	5	45	44.8 ± 0.4	44–45
transport	15	90	86.7 ± 3.3	81–90	5	47	46.2 ± 0.4	46–47
ferry	5	90	90.0 ± 0.0	90–90	5	60	60.0 ± 0.0	60–60
rovers	5	54	49.2 ± 2.9	46–54	5	43	42.2 ± 0.4	42–43
childsnack	10	67	61.7 ± 4.5	55–67	5	36	36.0 ± 0.0	36–36
sokoban	5	4	4.0 ± 0.0	4–4	5	36	35.4 ± 0.5	35–36
floortile	5	43	40.6 ± 2.7	36–43	5	57	50.0 ± 8.1	40–57
Total	65	648	631.0	608–648	56	474	461.6	446–474

from the state at training and test time. They state that the usable spanners carried already suffice for the nuts that remain to be tightened, that the state is a dead end, or neither. Added to the dead-end advantage, it raises coverage to 89 of 90. It is specific to spanner. Table 1 reports the manager without either change, since the sufficiency relation is domain knowledge.

Table 9: Spanner coverage per tier of the reported manager and of the two managers above, each with its best of five selected checkpoints, and the sum over the five checkpoints in the last column.

Manager	Easy	Medium	Hard	Total	Five checkpoints
Reported manager	29	1	0	30	145/450
Dead-end advantage	30	10	0	40	200/450
Dead-end advantage + sufficiency	30	30	29	89	427/450

Random Manager ablation. To see what the manager's choice is worth, we replace it by random selectors over its own candidates and keep everything else: the candidate vocabulary and the exclusions of the manager, the flags and limits of the reported pairings, and the executor, once the learned worker and once LIW(1). The first selector draws a random unsatisfied goal atom at every decision. When no goal atom is left, it falls back to a random currently false landmark atom, and then to any candidate. The second draws a random currently false landmark atom, which is the manager's whole vocabulary. A third fixes one random order of the goal atoms per instance and pursues the first unsatisfied atom of that order, with the fallbacks of the first. Each selector runs on all 900 test instances with three seeds, the third with one. As Table 1 reports the best tested pairing, we report a random selector by its best seed in each domain. Table 10 gives the coverage per tier and Table 11 the mean number of subgoals proposed and the mean plan length over the solved runs of a domain. Rows that solve different instances do not compare on these means, so the common-set columns restrict a random selector and the manager to the instances that both solve.

Coverage. With the learned worker, the random goal atom solves 487 instances, the random landmark 95 and the fixed order 472, against 648 for the manager. Figure 3 showed that outside blocksworld and spanner nearly every subgoal of the manager is a goal atom. In miconic, ferry, satellite and transport any order of the goal atoms works. The random selectors solve all 90 instances or all but one, and on the common instances they propose essentially as many subgoals as the manager, one per goal atom, every one reached. Childsnack and floortile see nothing but goal atoms from the manager as well, yet a random order solves 10 and 0 instances instead of 67 and 43. There the order matters, since the worker cannot achieve a goal atom drawn out of turn. In childsnack 55 runs end after 100 subgoals in a row are blocked before any action and 25 time out, in floortile 34 and 56. Blocksworld is where the manager's other landmarks count, the clear and on-table atoms of Figure 3. A random order of the goal atoms solves 60 instances and 4 of the hard

tier where the manager solves all 90, and 26 runs use up the 1 000 decisions. Spanner and sokoban are lost by every selector as they are by the manager beyond the easy tier: 84 runs end in dead ends in spanner and 86 hit the blocked cap in sokoban. In rovers the random order solves 47 against 54, and the rest time out.

Random landmark. The random landmark is hopeless. A currently false landmark atom is mostly not achievable from the current state, since it needs other landmarks first, and the worker's 100 actions do not reach it. The atom is then excluded and the next one drawn, and the run spends its decisions this way. In blocksworld 83 runs and in ferry 72 use up the 1 000 decisions, in satellite 73, and in childsnack and sokoban the runs end at the blocked cap. Where it solves instances, 44 in miconic and 17 in satellite, it needs 3 and 27 times the subgoals of the manager on the common instances, and its plans are 4 and 18 times as long. Without learning, the choice among all landmarks is no baseline at all. Among the goal atoms alone, the order is what the manager learns.

Plan quality. Where a random order of the goal atoms keeps the coverage, it pays in plan length. On the common instances the plans are 13 % longer in miconic and ferry, 3 to 9 % in transport and 28 to 36 % in satellite, at about the same number of subgoals. Since the worker is the same, the manager's order is what shortens the plans. Blocksworld is the exception in the other direction. On the 60 common instances the random order needs 264 subgoals and 1 074 actions where the manager needs 78 and 240. Rovers is the one domain where the random order gives shorter plans, by 3 to 5 %, while the manager solves more of the medium tier. Committing to one order changes little: the fixed order solves 472 instances against 487 for the per-decision draw, and outside blocksworld its plans are the same length up to a few percent.

LIW(1) as executor. The picture is the same with LIW(1) in place of the learned worker. The random goal atom solves 338 instances, the random landmark 51 and the fixed order 337, against 474 for the manager. The random order matches the manager within two instances in miconic, satellite, transport, ferry and rovers, and loses in blocksworld (24 against 47), sokoban (32 against 36), childsnack (9 against 36) and floortile (0 against 57). The failures are timeouts, since LIW(1) is slow at scale, and in childsnack and floortile subgoals that the search finds unreachable, 28 and 49 of them. On the common instances the plans are 10 to 23 % longer in miconic, ferry and satellite, and the same in transport, rovers and childsnack.

Table 10: Selector ablation: coverage per tier (E, M, H, 30 instances each) and over all 90 instances (All), for the learned worker π_W and for LIW(1) as executor. Learned is the reported manager of Table 1, and a random selector is reported by its best seed in each domain.

		π_W				LIW(1)			
Domain	Manager	E	M	H	All	E	M	H	All
blocksworld	Learned	30	30	30	90	30	17	0	47
	Random goal	26	30	4	60	15	9	0	24
	Random landmark	0	0	0	0	0	0	0	0
	Fixed order	22	24	3	49	13	9	0	22
miconic	Learned	30	30	30	90	30	30	14	74
	Random goal	30	30	30	90	30	30	12	72
	Random landmark	17	19	8	44	8	0	0	8
	Fixed order	30	30	30	90	30	30	12	72
spanner	Learned	29	1	0	30	29	0	0	29
	Random goal	6	0	0	6	6	0	0	6
	Random landmark	5	0	0	5	4	0	0	4
	Fixed order	6	0	0	6	6	0	0	6
satellite	Learned	30	30	30	90	30	15	0	45
	Random goal	30	30	30	90	30	14	0	44
	Random landmark	17	0	0	17	14	0	0	14
	Fixed order	30	30	29	89	30	14	0	44
transport	Learned	30	30	30	90	30	17	0	47
	Random goal	30	30	30	90	30	16	0	46
	Random landmark	7	0	0	7	7	0	0	7
	Fixed order	30	29	30	89	30	17	0	47
ferry	Learned	30	30	30	90	30	30	0	60
	Random goal	30	30	30	90	30	30	0	60
	Random landmark	0	0	0	0	5	0	0	5
	Fixed order	30	30	30	90	30	30	0	60
rovers	Learned	30	23	1	54	30	13	0	43
	Random goal	30	17	0	47	30	15	0	45
	Random landmark	19	0	0	19	12	1	0	13
	Fixed order	30	14	0	44	30	14	0	44
childsnack	Learned	30	30	7	67	30	6	0	36
	Random goal	7	2	1	10	7	2	0	9
	Random landmark	1	0	0	1	0	0	0	0
	Fixed order	8	2	1	11	8	2	0	10
sokoban	Learned	4	0	0	4	29	7	0	36
	Random goal	4	0	0	4	26	6	0	32
	Random landmark	2	0	0	2	0	0	0	0
	Fixed order	4	0	0	4	25	7	0	32
floortile	Learned	30	13	0	43	30	27	0	57
	Random goal	0	0	0	0	0	0	0	0
	Random landmark	0	0	0	0	0	0	0	0
	Fixed order	0	0	0	0	0	0	0	0
Total	Learned	273	217	158	648	298	162	14	474
	Random goal	193	169	125	487	204	122	12	338
	Random landmark	68	19	8	95	50	1	0	51
	Fixed order	190	159	123	472	202	123	12	337

Table 11: Selector ablation: plan quality over all 90 instances of each domain. Sg. is the mean number of subgoals proposed and Len. the mean plan length over the solved runs. Common set: n instances solved by both the random selector and the learned manager with the same executor, and the two means on them as random / learned.

		π_W					LIW(1)				
		Solved runs		Common set			Solved runs		Common set		
Domain	Manager	Sg.	Len.	n	Sg.	Len.	Sg.	Len.	n	Sg.	Len.
blocksworld	Learned	161	527				47	119			
	Random goal	264	1074	60	264 / 78	1074 / 240	98	357	24	98 / 42	357 / 105
	Random landmark	–	–	0	– / –	– / –	–	–	0	– / –	– / –
	Fixed order	304	1262	49	304 / 77	1262 / 236	105	339	22	105 / 44	339 / 115
miconic	Learned	107	378				50	181			
	Random goal	107	428	90	107 / 107	428 / 378	45	178	72	45 / 45	178 / 162
	Random landmark	140	721	44	140 / 45	721 / 163	11	15	8	11 / 3	15 / 12
	Fixed order	107	428	90	107 / 107	428 / 378	45	178	72	45 / 45	178 / 162
spanner	Learned	7	16				6	14			
	Random goal	1	7	6	1 / 1	7 / 7	1	7	6	1 / 1	7 / 7
	Random landmark	2	7	5	2 / 1	7 / 7	2	7	4	2 / 1	7 / 7
	Fixed order	1	7	6	1 / 1	7 / 7	1	7	6	1 / 1	7 / 7
satellite	Learned	117	173				21	32			
	Random goal	119	220	90	119 / 117	220 / 173	20	36	44	20 / 19	36 / 30
	Random landmark	179	236	17	179 / 7	236 / 13	141	180	14	141 / 6	180 / 11
	Fixed order	124	220	89	124 / 109	220 / 161	21	37	44	21 / 19	37 / 30
transport	Learned	47	207				11	48			
	Random goal	46	213	90	46 / 47	213 / 207	10	46	46	10 / 10	46 / 45
	Random landmark	389	2658	7	389 / 3	2658 / 11	110	196	7	110 / 2	196 / 9
	Fixed order	47	224	89	47 / 47	224 / 206	11	47	46	10 / 10	46 / 45
ferry	Learned	217	760				32	113			
	Random goal	217	866	90	217 / 217	866 / 760	32	127	60	32 / 32	127 / 113
	Random landmark	–	–	0	– / –	– / –	181	300	5	181 / 5	300 / 19
	Fixed order	217	866	90	217 / 217	866 / 760	32	127	60	32 / 32	127 / 113
rovers	Learned	51	361				31	121			
	Random goal	41	298	47	41 / 42	298 / 315	34	132	43	31 / 31	123 / 121
	Random landmark	55	1091	19	55 / 14	1091 / 62	40	77	13	40 / 11	77 / 45
	Fixed order	33	223	44	33 / 34	223 / 230	32	125	43	31 / 31	122 / 121
childsnack	Learned	25	723				9	41			
	Random goal	11	165	10	11 / 11	165 / 157	7	33	9	7 / 7	33 / 33
	Random landmark	3	32	1	3 / 4	32 / 20	–	–	0	– / –	– / –
	Fixed order	11	144	11	11 / 11	144 / 143	7	32	10	7 / 7	32 / 32
sokoban	Learned	1	10				4	75			
	Random goal	1	9	4	1 / 1	9 / 10	3	67	31	3 / 3	59 / 59
	Random landmark	2	16	0	– / –	– / –	–	–	0	– / –	– / –
	Fixed order	1	10	4	1 / 1	10 / 10	4	76	32	4 / 4	76 / 78
floortile	Learned	77	846				104	320			
	Random goal	–	–	0	– / –	– / –	–	–	0	– / –	– / –
	Random landmark	–	–	0	– / –	– / –	–	–	0	– / –	– / –
	Fixed order	–	–	0	– / –	– / –	–	–	0	– / –	– / –