Learning Sketch Decompositions in Planning via Deep Reinforcement Learning

Michael Aichmüller[1], **Hector Geffner**[1]

[1] RWTH Aachen University, Germany

Abstract

In planning and reinforcement learning, the identification of common subgoal structures across problems is important when goals are to be achieved over long horizons. Recently, it has been shown that such structures can be expressed as feature-based rules, called sketches, over a number of classical planning domains. These sketches split problems into subproblems which then become solvable in low polynomial time by a greedy sequence of IW(k) searches. Methods for learning sketches using feature pools and min-SAT solvers have been developed, yet they face two key limitations: scalability and expressivity. In this work, we address these limitations by formulating the problem of learning sketch decompositions as a deep reinforcement learning (DRL) task, where general policies are sought in a modified planning problem where the successor states of a state s are defined as those reachable from s through an IW(k) search. The sketch decompositions obtained through this method are experimentally evaluated across various domains, and problems are regarded as solved by the decomposition when the goal is reached through a greedy sequence of IW(k) searches. While our DRL approach for learning sketch decompositions does not yield interpretable sketches in the form of rules, we demonstrate that the resulting decompositions can often be understood in a crisp manner.

Introduction

A common challenge in planning and reinforcement learning is achieving goals that require many actions. Addressing this challenge typically involves learning useful subgoals or hierarchical policies that abstract primitive actions (Sutton, Precup, and Singh 1999; McGovern and Barto 2001; Kulkarni et al. 2016; Park et al. 2024). Yet the principles underlying the corresponding problem decompositions are not well understood. Consequently, methods for learning subgoals and hierarchical policies often lack robustness, working effectively in some domains while failing completely in others, without a clear explanation for these differences in performance. Recently, a powerful language for expressing, learning, and understanding general problem decompositions has been proposed (Bonet and Geffner 2021; Drexler, Seipp, and Geffner 2022). A *sketch decomposition* for a class of problems $\mathcal{Q}$ defines a set of subgoal states $G(s)$ for each reachable state s in an instance $P \in \mathcal{Q}$. In any state s, the planner's task is not to reach the distant problem goal but to move to a closer subgoal state in $G(s)$.

The concept of a goal being easily reachable or not is formalized through the notion of *problem width* (Lipovetzky and Geffner 2012; Bonet and Geffner 2024). A class of problems with width bounded by a constant k can be solved optimally by the IW(k) algorithm in time exponential in k. Many planning domains have a width no greater than 2 when goals are restricted to single atoms. A sketch decomposition $G(\cdot)$ divides problems P in class $\mathcal{Q}$ into subproblems $P[s, G(s)]$, which resemble P but with initial state s and goal states $G(s)$. If all these subproblems have width bounded by k, the decomposition width over $\mathcal{Q}$ is bounded by k, allowing the problems in $\mathcal{Q}$ to be solved by a greedy sequence of IW(k) calls, provided the decomposition is acyclic and safe, meaning no subgoal cycles or dead-end states among the subgoals (Bonet and Geffner 2021).

Methods for learning safe, acyclic sketch decompositions with bounded width, represented by a set of sketch rules, have been developed (Drexler, Seipp, and Geffner 2022, 2023), following techniques previously used for learning general policies (Frances, Bonet, and Geffner 2021). These learning methods rely on feature pools derived from domain predicates and a min-cost SAT solver, leading to two key limitations: scalability and expressivity. Large feature pools enhance expressivity but result in large theories that are difficult for combinatorial solvers to handle.

In this work, we address these limitations by framing the problem of learning sketch decompositions as one of learning general policies in a deep reinforcement learning (DRL) context. Here, feature pools are not made explicit, and combinatorial solvers are unnecessary. We build on a novel observation connecting sketch decompositions with general policies and leverage an existing implementation for learning general policies via DRL (Ståhlberg, Bonet, and Geffner 2023). The resulting method learns sketch decompositions bounded by a given width parameter k and uses them to search for goals across various domains through a greedy sequence of IW(k) searches. Unlike symbolic methods, the DRL approach does not produce rule-based sketches but neural network classifiers. However, as we will demonstrate, while interpreting these classifiers is not straightforward, it is often possible to understand the resulting decompositions in a crisp manner.

The structure of the paper is as follows. We begin with

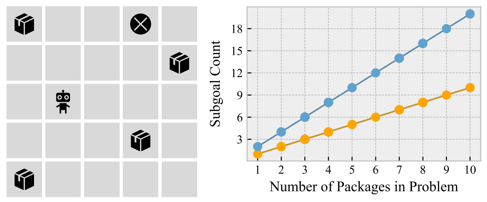

Figure 1: *Left.* A 5×5 Delivery instance with 4 packages and target cell X. *Right.* Number of subgoals resulting from learned decompositions G_k via DRL over test instances as function of number N of packages. For $k = 1$, problems are solved by $2N$ calls to IW(1); while for $k = 2$, by N calls to IW(2).

an illustrative example and relevant background. We then present the proposed formulation, followed by experiments, an analysis of the decompositions found, related work, and a concluding discussion.

Example

The Delivery domain, similar to the Taxi domain in hierarchical reinforcement learning, involves N packages spread across an $n \times m$ grid, with an agent tasked to deliver them, one by one, to a target cell. The sketch decomposition G_2, where $s' \in G_2(s)$ if the number of undelivered packages $u(\cdot)$ is smaller in s' than in s, yields subproblems $P[s, G_2(s)]$ of width bounded by 2, solved optimally by IW(2). The subproblems $P[s, G_2(s)]$ are like P but with initial state s and goal states $G_2(s)$. Similarly, the sketch decomposition G_1, where $s' \in G_1(s)$ if either $u(s') < u(s)$ and a package is held in s, or $u(s') = u(s)$ and a package is not held in s but is held in s', produces subproblems $P[s, G_1(s)]$ of width 1, solvable optimally by IW(1).

A previous approach learns such decompositions using feature pools, a width parameter $k \in \{1, 2\}$, and combinatorial methods (Drexler, Seipp, and Geffner 2022). The decompositions are represented implicitly by collections of *sketch rules*. This work is aimed at learning similar width-k decompositions $G_k(s)$ but using neural networks trained via reinforcement learning. While the learned representations will not be as transparent, we will see that the resulting decompositions often are. Figure 1 shows indeed the number of IW(1) and IW(2) calls needed to solve Delivery instances as a function of the number of packages N after learning two general domain decompositions G_1 and G_2 via deep reinforcement learning. The number of calls, $2N$ and N, match exactly the number of calls that would be needed to solve these instances using the sketch decompositions defined above, despite using no rule-based representation or explicit feature pool, but solely a neural net trained via RL.

Background

We briefly review classical planning, the notion of width, general policies and sketches, and methods for learning them, following Lipovetzky and Geffner (2012), Frances, Bonet, and Geffner (2021), Bonet and Geffner (2021), Drexler, Seipp, and Geffner (2022), and Ståhlberg, Bonet, and Geffner (2023).

Classical and Generalized Planning

A *planning problem* or *instance* is a pair $P = (D, I)$ where D is a first-order *domain* with action schemas defined over predicates, and I contains the objects in the instance and two sets of ground atoms defined over the objects and predicates defining the initial and goal situations *Init* and *Goal*. An instance P defines a state model $S(P) = (S, s_0, G, Act, A, f)$ where the states in S are the possible sets of ground atoms, each one capturing the atoms that are true in the state. The initial state s_0 is *Init*, the set of goal states G are those that include the goal atoms $Goal$, and the actions Act are the ground actions obtained from the schemas and objects. The ground actions in $A(s)$ are the ones that are applicable in a state s; namely, those whose preconditions are (true) in s, and the state transition function f maps a state s and an action $a \in A(s)$ into the successor state $s' = f(a, s)$. A *plan* π for P is a sequence of actions $a_0, \ldots, a_n$ that is executable in s_0 and maps the initial state s_0 into a goal state; i.e., $a_i \in A(s_i)$, $s_{i+1} = f(a_i, s_i)$, and $s_{n+1} \in G$. A state s is *solvable* if there exists a plan starting at s, otherwise it is a *dead-end*. The cost of a plan is assumed to be given by its length, and a plan is optimal if there is no shorter plan.

A *generalized planning* problem instead is given by a *collection* $\mathcal{Q}$ of instances $P = (D, I)$ from a given domain; for example, all instances of Blocks world where the goal just involves *on* atoms. The solution of a generalized problem is not an open-loop action sequence but a closed loop policy as detailed below. In general, the instances in $\mathcal{Q}$ are assumed to be solvable, and moreover, the set $\mathcal{Q}$ is normally assumed to be closed in the sense that if P is in $\mathcal{Q}$ with initial state s_0 and P' is P but with a solvable initial state reachable from s_0, then P' is assumed to be in $\mathcal{Q}$ as well.

Width

The simplest width-based search procedure is IW(1), a modified breadth-first search over the rooted directed graph associated with the state model $S(P)$. It prunes newly generated states that fail to make an atom true for the first time in the search. IW(k), for $k > 1$, extends this concept by pruning states that do not make a collection of up to k atoms true for the first time. These algorithms can be alternatively conceptualized using the notion of state novelty. In this view, IW(k) prunes states with novelty greater than k, where a state's novelty is defined by the size of the smallest set of atoms true in that state and false in all previously generated states. Central to these algorithms is the concept of problem width. The width of a problem P is determined by the size of the smallest chain of atom tuples $t_0, \ldots t_n$ that is *admissible* in P and has size $\max_i |t_i|$ (Lipovetzky and Geffner 2012). For instance, Blocks World instances with atomic goals $on(x, y)$ and Delivery instances with goals $at(pkg, loc)$ have width 2 or less.

IW(k) algorithms find optimal (shortest) solutions in time and space exponential to the problem width. However, planning problems with multiple conjunctive goals often lack a bounded width (i.e., width independent of the instance size). To address this, a variant called SIW was developed. SIW greedily seeks a sequence of IW calls, each decreasing the number $\#g$ of unachieved atomic goals (Lipovetzky and Geffner 2012). It starts with IW(1), escalating to IW(2) and beyond if IW(1) fails to reach a state that decreases $\#g$. While SIW exploits a particular *problem decomposition* based on unachieved goals, this approach is not universally effective due to potential high-width or unsolvable subproblems. Complete, width-based search algorithms incorporate novelty measures within a best-first search (Lipovetzky and Geffner 2017; Francès et al. 2017).

For convenience, the width of problems P that can be solved in at most one step, is said to have width 0. Hence, IW(0) is defined as breadth-first search that prunes all and only nodes at depth greater than 1, and IW(k) is adjusted to never prune nodes at level 1.

General Policies and Sketches

A simple but powerful way to express the solutions to generalized planning problems $\mathcal{Q}$ made up of a collection of instances P, is by means of *rules* of the form $C \mapsto E$ defined over a set of features Φ (Bonet and Geffner 2018). A *state pair* $[s, s']$ satisfies the rule if C is true in s and the features in Φ change value when moving from s to s' in agreement with E. For example, E can express that a numerical feature must increase its value, and that a Boolean feature must become true, etc. A set of rules R defines a non-deterministic *general policy* π for $\mathcal{Q}$ which in any reachable state s in $P \in \mathcal{Q}$ selects the successor states s' of s when the state pair $[s, s']$ satisfies a rule in R. The transitions (s, s') are then called π*-transitions*, and the policy π solves an instance $P \in \mathcal{Q}$ if all the π-trajectories that start in the initial state of P reach a goal state. [1]

The same language used to define general policies can be used to define *sketch decompositions*. Indeed, a set of rules R defines the subproblems $P[s, G_R(s)]$ over the reachable non-goal states s of instances $P \in \mathcal{Q}$, which are like P but with initial state s and goal states $s' \in G_R(s)$ for the state pairs $[s, s']$ that satisfy a rule in R. The *width* of the decomposition is the maximum width of the subproblems $P[s, G_R(s)]$, $P \in \mathcal{Q}$, and the decomposition is *safe* and *acyclic* in $\mathcal{Q}$ if there is no sequence of (subgoal) states $s_1, \ldots, s_n$, $s_{i+1} \in G^*_R(s_i)$ and $n > 1$, in any $P \in \mathcal{Q}$ that starts in a reachable, alive state s_1 (not a dead-end, not a goal), and ends in a dead-end state s_n or in the same state $s_n = s_1$. Here $G^*_R(s)$ stands for the states $s' \in G_R(s)$ that are closest to s. If the decomposition resulting from the rules R is safe, acyclic, and has width bounded by k, then the problems $P \in \mathcal{Q}$ can be solved by a slight variant of the SIW algorithm where a sequence of IW(k) calls is used to move iteratively and optimally from a state s_i to a subgoal state $s_{i+1} \in G_R(s_i)$, starting from the initial state of P and ending in a goal state (Bonet and Geffner 2021).

Learning General Policies through DRL

Rule-based policies and sketches can be learned without supervision by solving a min-cost SAT problem over the state transitions of a collection of small training instances P from $\mathcal{Q}$ and a pool of features derived from domain predicates and a fixed set of grammar rules based on description logics (Bonet, Frances, and Geffner 2019; Drexler, Seipp, and Geffner 2022). However, some domains require highly expressive features, which necessitate the application of numerous grammar rules, leading to large feature pools that challenge combinatorial solvers.[2]

To address this limitation, a recent approach introduced learning general policies in a deep reinforcement learning (DRL) setting (Ståhlberg, Bonet, and Geffner 2023). This approach employs a standard actor-critic RL algorithm (Sutton and Barto 1998) with the policy and value functions $\pi(s' \mid s)$ and $V(s)$ represented by neural networks. As shown in Fig. 1, gradient descent updates the parameters θ and ω of the policy and value functions. The key difference from standard actor-critic codes is that π selects the next state from possible successors $N(s)$ instead of the next action.

The learned policy functions generalize to larger domain instances than those used in training. This generalization is achieved by encoding the policy and value functions in terms of a relational GNN, which generates real-vector embeddings $f^s(o)$ for each object in the instance (Scarselli et al. 2008; Hamilton 2020). Suitable readout functions then map these embeddings into the values $V(s)$ and the probabilities $\pi(s' \mid s)$ (Ståhlberg, Bonet, and Geffner 2023).

Algorithm 1: Actor-Critic RL for generalized planning

```
1:  **Input:** Training MDPs $\{M_i\}_i$, each with state priors $p_i$
2:  **Input:** Policy $\pi(s|s')$ with parameter $\theta$
3:  **Input:** Value function $V(s)$ with parameter $\omega$
4:  **Ouput:** Policy $\pi(s|s')$
5:  **Parameters:** Step sizes $\alpha, \beta > 0$, discount factor $\gamma$
6:  Initialize parameters $\theta$ and $\omega$
7:  Loop forever:
8:      Sample MDP index $i \in \{1, \ldots, n\}$
9:          Sample non-goal state $S$ in $M_i$ with probability $p_i$
10:         Sample state $S'$ from $N(S)$ with prob. $\pi(S'|S)$
11:         Let $\delta = 1 + \gamma V(S') - V(S)$
12:         $\omega \leftarrow \omega + \beta\delta\nabla V(S)$
13:         $\theta \leftarrow \theta - \alpha\delta\nabla \log \pi(S'|S)$
14:         If $S'$ is a goal state, $\omega \leftarrow \omega - \beta V(S')\nabla V(S')$
```

Learning Decompositions via DRL

The contribution of this paper is a novel scheme for learning how to decompose planning problems into subproblems that can be solved through iterative applications of the IW(k) algorithm. Decomposing problems into subproblems is crucial, but the principles guiding such decompositions are not

[1]These general policies do not map states directly to actions. Instead, they select state transitions and, only indirectly, actions. This choice is convenient for relating general policies and sketches (Bonet and Geffner 2021).

[2]A second type of expressive limitation, not discussed here, involves domains where rule-based policies and sketches may require features beyond the capabilities of standard description logic grammars, which are generally fragments of first-order logic with two variables and counting. This limitation also affects non-symbolic approaches based on GNNs (Ståhlberg, Bonet, and Geffner 2022; Horčík and Šír 2024).

well-defined. Our goal is to achieve decompositions that are general (applicable across a class of problems $\mathcal{Q}$), safe (avoiding dead-ends), acyclic, and have width bounded by k. Additionally, we aim to learn these decompositions without relying on combinatorial solvers or explicit feature pools, leveraging the relationship between sketch compositions and general policies, as well as the method reviewed above for learning general policies through DRL.

It is known that general policies are sketch decompositions of zero width, which are safe and acyclic (Bonet and Geffner 2024). Our new observation is that safe, acyclic sketch decompositions with approximate width $k > 0$ for a class of problems $P \in \mathcal{Q}$ can be derived from general policies over a slightly different class of problems $P_k \in \mathcal{Q}_k$, where *the set of successor states $N(s)$ in P is replaced by the set $N_k(s)$ of states reachable from s via IW(k)*:

$$N_k(s) \coloneqq \{s' \mid s' \text{ is reachable from } s \text{ via IW}(k)\}. \quad (1)$$

The *successor states* s' that the policy $\pi(s' \mid s)$ selects in P_k will be subgoal states s' in P that can be reached from s via IW(k). The decomposition's width is approximately bounded by k because while a width bounded by k guarantees reachability via IW(k), the reverse is not necessarily true.[3]

The modification of the DRL algorithm from (Ståhlberg, Bonet, and Geffner 2023) to learn safe and acyclic decompositions of width k over a class of problems $P \in \mathcal{Q}$ is straightforward: the only change required is to replace the set of successor states $N(s)$ in line 10 of Algorithm 1 with the set $N_k(s)$ reachable from s via IW(k). This extended set of successor states is then used in the softmax normalization to yield the probabilities $\pi(s' \mid s)$. Action costs are assumed to be all 1 for reaching either N or N_k successors.

Let $\pi(s' \mid s)$ be the general stochastic policy learned by the algorithm in Fig. 1 after replacing $N(s)$ with $N_k(s)$. The resulting decomposition $G^{\pi}_k(\cdot)$ can then be defined in two ways: *greedily*, as the singleton sets:

$$G^{\pi}_k(s) \coloneqq \{\, s' \,\},\ s' = \operatorname*{arg\,max}_{s' \in N_k(s)} \pi(s' \mid s), \quad (2)$$

and *stochastically*, as the singleton sets:

$$G^{\pi}_k(s) \coloneqq \{\, s' \,\},\ s' \sim \pi(s' \mid s), s' \in N_k(s). \quad (3)$$

In the first case, a single subgoal state s' for s is chosen as the *most likely state* in $N_k(s)$ according to the learned policy π; in the second, case, s' is *sampled stochastically* from the set $N_k(s)$ with probability $\pi(s' \mid s)$. In P, s' may not be a direct successor of s but can be reached from s via IW(k). Intuitively, to decompose the problem, we are allowing the "agent" to make IW(k) "jumps" in P following the learned policy for P_k where such "jumps" are primitive actions.

If the policy π solves the problem P_k, then the decomposition G^{π}_k will be *safe* and *acyclic*. A sequence of subgoal states $s_0, s_1, \ldots, s_n$ with $n > 1$ and $s_{i+1} \in G^{\pi}_k(s_i)$ for $i = 1, \ldots, n-1$, cannot be cyclic or unsafe, as that would imply the existence of π-trajectories that do not reach the goal, contradicting the assumption that π solves P_k. Additionally, this implies the subproblems $P[s, G^{\pi}_k(s)]$ to have an *approximate* width bounded by k, as $G^{\pi}_k(s) \in N_k(s)$ only includes states reachable from s via IW(k).

In summary, Algorithm 1 is adapted with minor modifications to learn a safe, acyclic, and width-k decomposition $G_k(s)$ for a class of problems $\mathcal{Q}$, though without formal guarantees.[4] The only change involves replacing the set of successor states $N(s)$ in $P \in \mathcal{Q}$ with the set $N_k(s)$ defined in (1). The learned stochastic general policy π for the resulting class of problems $P_k \in \mathcal{Q}_k$ defines the decomposition $G_k = G^{\pi}_k$ over $\mathcal{Q}$ as described in (3).

At test time, the decomposition G^{π}_k is evaluated by running the IW(k) algorithm sequentially from a state s to a state s' in $G^{\pi}_k(s)$ until reaching the goal or a maximum number of IW(k) calls. We refer to this algorithm, which applies IW(k) searches to the decomposition $G^{\pi}_k(s)$ based on the learned policy π, as SIW$^{\pi}$(k). Unlike the SIW algorithm, SIW$^{\pi}$(k) performs a greedy sequence of IW(k) searches rather than IW searches, requiring each search to end in a state within $G^{\pi}_k(s)$, not merely a state where the number of unachieved top goals has decreased. Moreover, unlike SIW, SIW$^{\pi}$(k) requires IW(k) searches to run to completion, as this is necessary for determining the extended set of successors $N_k(s)$ in the decomposition $G^{\pi}_k(s)$ defined in (1).

Experiments

The experiments aim to address several key questions. First, are the learned decompositions $G_k = G^{\pi}_k$ both general and effective? Specifically, can the (larger) test instances be solved by a greedy sequence of IW(k) calls? This question is non-trivial, as success with $k = 1$ would imply solving instances with linear memory relative to the number of atoms by running IW(1) sequentially, a significant contrast to solving instances via exponential time and memory search. Second, can the resulting decompositions, represented in the neural network, be understood and interpreted? This is also challenging, as there is no guarantee that the learned decompositions will be meaningful. To answer the first question, we will examine the coverage of the SIW$^{\pi}$(k) algorithm using the learned decomposition G^{π}_k, the number of IW(k) calls (subgoals), and the total plan length. To address the second question, we will analyze plots showing the number of subgoals resulting from the learned decomposition G^{π}_k as a function of relevant parameters of the test instances (e.g., the number of packages). In the experiments, the subgoal states $G^{\pi}_k(s)$ are sampled stochastically according to (3), although the results (in the appendix) are not too different when they are chosen deterministically as in (2). The code and data will be made publicly available.

[3]In (Lipovetzky and Geffner 2012), reachability via IW(k) is referred to as *effective width k*, which is not a robust notion of width, as it is influenced by the order in which child nodes are explored in the breadth-first search.

[4]The symbolic method for learning sketches (Drexler, Seipp, and Geffner 2022) enforces these properties in the training set but cannot guarantee them over the test set. However, this can be addressed manually, case by case.

Learning Setup. We use the DRL implementation from (Ståhlberg, Bonet, and Geffner 2023) with the same hyperparameters to learn the policy π that defines the decomposition G^{π}_{k}. The GNN has feature vectors of size 64 and 30 layers. The Actor-Critic algorithm uses a discount factor $\gamma = 0.999$, a learning rate $\alpha = 2 \times 10^{-4}$, the Adam optimizer (Kingma and Ba 2015), and runs on a single NVIDIA A10 GPU for up to 48 hours per domain. Five models are trained independently with different seeds, and the model with the best validation score is selected for testing. The validation score is determined by the ratio L_V / L^*_V, where L_V is the plan length from SIW$^{\pi}(k)$ and L^*_V is the optimal plan length, both averaged over all states of a validation set. Training is stopped early if this ratio approaches 1.0.

Data. The domains and training data are primarily from previous works on learning sketches and general policies (Drexler, Seipp, and Geffner 2022; Ståhlberg, Bonet, and Geffner 2023). This includes Blocks with single and multiple target towers (19 and 22 training instances, respectively), Childsnack (49 instances), Delivery (75), Grid (13), Gripper (10), Logistics (25), Miconic (63), Reward (15), Spanner (140), and Visitall (12). Each domain is tested on 40 larger instances, which extend those used in prior studies (details in the appendix).

Results

Table 1 presents the performance of the SIW$^{\pi}(k)$ algorithm using the learned G^{π}_{k} decomposition, where π is the policy derived from the RL algorithm after replacing the set of successors $N(s)$ with $N_k(s)$. Key performance metrics include coverage (Cov), subgoal count (SL), and plan length (L). The table's upper section shows results for IW(1), while the lower section displays IW(2) results for selected domains.

In the table, L_M indicates the plan length computed by the classical planner LAMA, run on an Intel Xeon Platinum 8352M CPU with a 10-minute time and 100 GB memory limit. The columns labeled "subgoal cycle prevention" reflect a minor SIW$^{\pi}(k)$ algorithm modification that avoids revisiting a subgoal state. For this, states that have already been selected as subgoals before are not considered as future subgoals. This adjustment impacts performance in three of the eleven domains, including two (Grid and Logistics) where the width-1 decompositions learned were poor.

Coverage The SIW$^{\pi}(k)$ algorithm achieves nearly perfect coverage across all domains, except in Logistics, Grid, and Blocksworld Multiple with IW(1). However, for width-2 decompositions, coverage improves to near 100% in all domains, including these three. The reason for this discrepancy is not entirely clear, but one possibility is that width-1 sketch decompositions cannot be fully captured in the logical fragment represented by GNNs. It is known that GNNs cannot represent width-0 sketch decompositions (i.e., general policies) for Logistics and Grid (Ståhlberg, Bonet, and Geffner 2022), suggesting that the same might hold for width-1 decompositions. Interestingly, GNNs do accommodate width-2 sketch decompositions in these domains, as shown in the table, aligning with findings from previous research, which observed that while certain feature pools cannot express rule-based general policies, they can express rule-based sketches.

Subgoal Count The column SL in the table presents the average number of subgoals encountered by SIW$^{\pi}(k)$ on the path to the goal. Although this number alone may not be highly meaningful, it is significantly lower than the average plan lengths, indicating that each subgoal requires multiple actions to be achieved. More interestingly, Figure 2 illustrates the number of subgoals as a function of the number of objects of a selected type per domain (e.g., packages in Delivery, balls in Gripper, children in Childsnack) for $k = 1$. In these cases, the relationship is nearly linear, with a coefficient of 1 in Spanner and Reward, and 2 in Delivery, Gripper, Childsnack, and Miconic. This suggests that the decomposition divides the problem into subproblems, one for each relevant object, which in some cases are further split in two (e.g., in Delivery, each package must be picked up and dropped off in separate IW(1) calls). Despite the use of neural networks and DRL, the resulting decompositions can be understood. In Blocksworld, however, the situation is different. The plots in Fig. 3 show that the width-1 decomposition generates more subgoals than there are on-atoms in the goal (shown in blue versus black), while the width-2 decomposition generates fewer subgoals than on-atoms in the goal (shown in yellow). While individual on-atoms have a width of 2 and are thus always reachable by IW(2), certain states allow for pairs of on-atoms to be reached by IW(2) as well. The result is that the plots of subgoal counts in Blocksworld are not showing strict linear relationships.

Plan Quality The column L/L_M in the table shows the ratio between the average plan lengths found by SIW$^{\pi}(k)$ and those found by LAMA. Generally, this ratio is close to 1, but there are exceptions in certain domains, such as Reward and Visitall for IW(1) and several others for IW(2). The general explanation is that decompositions simplify the problems, allowing them to be solved by a linear search like IW(1), rather than an exponential search as in LAMA. This simplification, however, can preclude shortcuts, resulting in longer plans. A more specific explanation is that the DRL algorithm minimizes the number of IW(k) subproblems on the path to the goal, without considering the cost of solving these subproblems, as measured by the number of actions required. For instance, if a single package is to be delivered from a state s with two options—a nearby package and a distant one-SIW$^{\pi}(k)$ does not prefer one over the other since both states s' and s'' where either package is held are N_k-successors of s with the same cost of 1. To address this limitation, two approaches could be considered: 1) modifying SIW$^{\pi}(k)$ to prefer N_k-successors s' that have a high probability $\pi(s' \mid s)$ and are closer to s, or 2) retaining the true cost information of N_k-successors and using this in the DRL algorithm to optimize a combination of the number of subgoals and the cost of achieving them. The first approach would involve modifying the SIW$^{\pi}(k)$ algorithm, while the second would require changes to the training process. Exploring these approaches, however, is beyond the scope of this work.

It is also worth noting that problem representations (in

	LAMA	No Cycle Prevention				Subgoal Cycle Prevention				Validation
Domain (#)	Cov. (%) ↑	Cov. (%) ↑	SL ↓	L ↓	PQ = L / L_M ↓	Cov. (%) ↑	SL ↓	L ↓	PQ = L / L_M ↓	L_V / L_V^* ↓
					SIW$^\pi(1)$					
Blocks (40)	40 (100 %)	39 (98 %)	21	80	1.05 = 80 / 76	40 (100 %)	21	81	1.05 = 81 / 77	1.22
Blocks-mult. (40)	39 (98 %)	32 (80 %)	19	57	1.08 = 57 / 53	39 (98 %)	23	68	1.15 = 66 / 57	1.32
Childsnack (40)	40 (100 %)	40 (100 %)	6	11	1.06 = 11 / 10	40 (100 %)	6	11	1.05 = 11 / 10	1.00
Delivery (40)	40 (100 %)	40 (100 %)	10	52	1.02 = 52 / 50	40 (100 %)	10	52	1.02 = 52 / 50	1.00
Grid (40)	38 (95 %)	23 (58 %)	7	39	1.18 = 39 / 33	38 (95 %)	71	353	10.03 = 353 / 35	11.85
Gripper (40)	40 (100 %)	40 (100 %)	83	165	1.33 = 165 / 124	40 (100 %)	83	164	1.33 = 164 / 124	1.00
Logistics (40)	38 (95 %)	10 (25 %)	8	19	1.30 = 16 / 12	24 (60 %)	113	188	10.90 = 199 / 18	60.36
Miconic (40)	40 (100 %)	40 (100 %)	32	60	1.15 = 60 / 52	40 (100 %)	32	61	1.16 = 61 / 52	1.00
Reward (40)	40 (100 %)	40 (100 %)	15	197	2.32 = 197 / 85	40 (100 %)	15	196	2.31 = 196 / 85	1.00
Spanner (40)	30 (75 %)	40 (100 %)	24	44	1.31 = 41 / 31	40 (100 %)	24	44	1.30 = 41 / 31	1.00
Visitall (40)	40 (100 %)	40 (100 %)	8	68	1.65 = 68 / 41	40 (100 %)	8	68	1.66 = 68 / 41	1.03
					SIW$^\pi(2)$					
Blocks (40)	40 (100 %)	40 (100 %)	9	133	1.71 = 133 / 77	40 (100 %)	9	133	1.71 = 133 / 77	1.27
Blocks-mult. (40)	39 (98 %)	40 (100 %)	8	78	1.35 = 77 / 58	40 (100 %)	8	78	1.34 = 77 / 58	1.07
Childsnack (40)	40 (100 %)	40 (100 %)	3	13	1.21 = 13 / 10	40 (100 %)	3	13	1.21 = 13 / 10	1.00
Delivery (40)	40 (100 %)	40 (100 %)	5	57	1.12 = 57 / 50	40 (100 %)	5	56	1.12 = 56 / 50	1.00
Grid (40)	38 (95 %)	38 (95 %)	3	47	1.34 = 47 / 35	38 (95 %)	3	47	1.34 = 47 / 35	1.00
Logistics (40)	38 (95 %)	40 (100 %)	4	34	1.33 = 34 / 26	40 (100 %)	4	34	1.33 = 34 / 26	1.01

Table 1: Performance metrics for learned general decompositions per domain and width. Rows show results over 40 test instances per domain. Coverage (Cov.) indicates the number of successful plans (fraction in parentheses). SL and L represent average subgoal and plan lengths (rounded to nearest integer), while L^* and L_M represent optimal and LAMA plan lengths resp. averaged. Plan quality (PQ) is the ratio of average plan lengths to those of LAMA. Validation score L_V / L_V^* is shown on the right. Arrows indicate preferred metric directions.

PDDL) influence problem width and overall plan length. For example, the width-1 decomposition for Gripper results in moving balls from one room to the next one by one, even though two grippers are available. This explains why the plans generated by SIW$^\pi(1)$ are 33% longer than those computed by LAMA, which are optimal. One reason for this inefficiency is that IW(1) cannot load two grippers in a single call; the problem indeed has a width of 2. This situation would differ if the problem representation included a single atom that is true when both grippers are full. Thus, the ability to learn width-k decompositions is not just limited by the expressive power of GNNs, but also by the problem representation. Making these limitations and these dimensions explicit is a strength of our approach, enabling us to understand the general decompositions that are learned and those that are not or cannot be learned, rather than relying solely on performance metrics.

Analysis

The symbolic approach to problem decompositions using sketch rules offers transparency, allowing direct interpretation of the defined decompositions. Interestingly, the inverse process is also possible: equivalent sketch rules can be reconstructed from learned decompositions by analyzing subgoal counts and plan structures. Indeed, the width-1 decompositions learned for Delivery, Gripper, Miconic, Childsnack, and Spanner can be understood in terms of four sketches (Bonet and Geffner 2021) with numerical features N_i and Boolean feature H:

$$\{N > 0\} \rightarrow \{N \downarrow\} \qquad (R1)$$

$$\{\neg H, N > 0\} \rightarrow \{H\} \qquad (R2)$$
$$\{H, N > 0\} \rightarrow \{\neg H, N \downarrow\}$$

$$\{N_2 > 0\} \rightarrow \{N_2 \downarrow\} \qquad (R3)$$
$$\{N_1 > 0, N_2 = 0\} \rightarrow \{N_1 \downarrow\}$$

$$\{N_2 > 0\} \rightarrow \{N_2 \downarrow\} \qquad (R4)$$
$$\{N_1 > 0, N_2 = 0\} \rightarrow \{N_1 \downarrow, N_2 ?\}$$

Sketch $R1$ represents a simple feature-decrementing sketch, capturing the decomposition in Reward, where N denotes the number of uncollected rewards. $R2$ models the decomposition in Delivery, where H indicates whether an undelivered package is held, and N tracks the number of undelivered packages. $R3$ uses two-counters, and prioritizes decrementing N_2 until exhaustion, then focuses on N_1. This ruleset characterizes the decomposition of domains like Miconic and Childsnack, where N_2 represents intermediate goals (e.g., sandwiches to make, passengers to board) and N_1 denotes final objectives (e.g., children/passengers to serve).

The plans in the Spanner domain generally follow the sketch $R3$, where the agent first picks up all the spanners (decrementing N_1) and then tightens all the nuts (decrementing N_2). If this process were perfectly aligned with the sketch, the points in the plot would fall exactly on the line $y = x$. However, the slight deviations from this line allow two observations. Points above the line represent unneces-

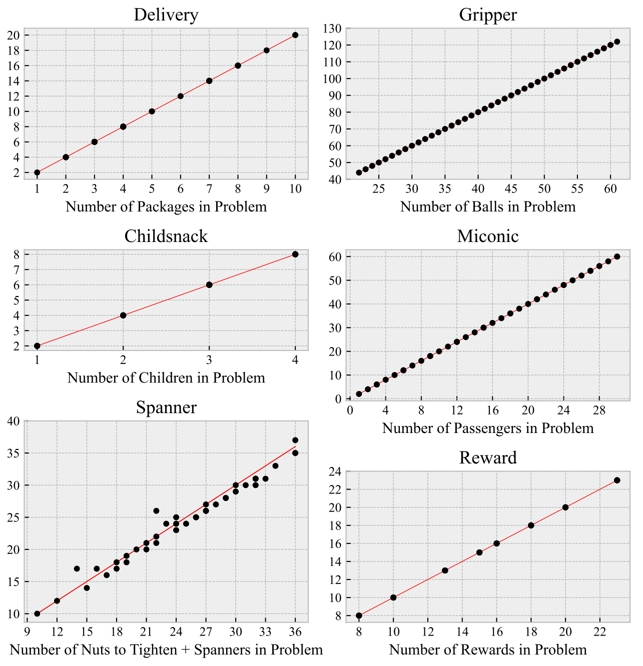

Figure 2: Plots showing number of subgoals (y) as function of number of objects (x) of selected types in six domains. The curves are perfect lines $y = 2x$ in four domains, $y = x$ in one domain, and approximately $y = x$ in one domain (Spanner).

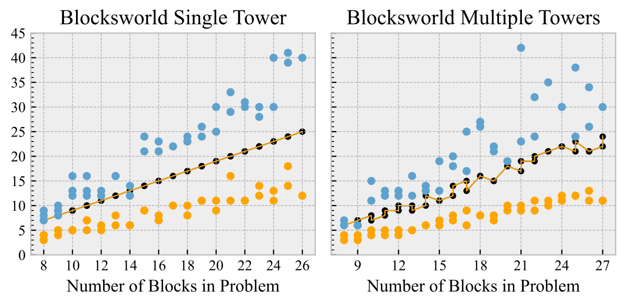

Figure 3: Subgoal counts (N_1, N_2) in Blocks domain for width-1 and width-2 decompositions (blue, yellow) vs. block count. Black dots show on-atom count (N). $N_2 < N < N_1$, indicating IW(2) reaches on-atoms in each call, while IW(1) does not.

sary subgoals where neither N_1 nor N_2 change. Points below the line indicate shortcuts in instances with more spanners than nuts where some of the spanners are left uncollected. Often, in these cases however, more spanners are picked up than strictly needed. A possible explanation is that computing the minimum number of spanners to collect, in order to solve the task, may be beyond the expressivity of GNNs given the state encodings.

Finally, $R4$ extends $R3$ with a potential increment of N_2 when decrementing N_1. These rules describe the decomposition in Gripper, where N_1 represents undelivered balls and N2 denotes balls available for pickup, i.e., the minimum of undelivered balls and free grippers. Indeed, the model alternates between picking and delivering balls, but only once both grippers hold a ball.

Related Work

Methods for decomposing problems into subproblems have been extensively studied in hierarchical planning (Erol, Hendler, and Nau 1994; Nau et al. 1999; Georgievski and Aiello 2015). Hierarchical representations can be derived from precondition relaxations (Sacerdoti 1974) and causal graphs (Knoblock 1994), or learned from annotated traces (Hogg, Munoz-Avila, and Kuter 2008; Zhuo et al. 2009). In RL, problem substructure emerges in the form of options (Sutton, Precup, and Singh 1999), hierarchies of abstract machines (Parr and Russell 1997), MaxQ hierarchies (Dietterich 2000), reward machines (Icarte et al. 2018; De Giacomo et al. 2020), and intrinsic rewards (Singh et al. 2010; Zheng et al. 2020), among others. Although this knowledge is often provided by hand, methods for learning these structures have leveraged concepts such as "bottleneck states" (McGovern and Barto 2001), eigenvectors of the transition dynamics (Machado, Bellemare, and Bowling 2017), and informal width-based notions (Junyent, Gómez, and Jonsson 2021). Additionally, two-level hierarchical policies in DRL have been explored (Kulkarni et al. 2016; Park et al. 2024), with the assumption that a master policy can make multi-step "jumps" executed by a low-level worker policy. However, the challenge with bounding these jumps by a fixed number of steps (e.g., 8 steps) is that it fails to capture meaningful substructure that generalizes to larger instances.

Our approach is closely related to these two-level hierarchies but differs in that the "jumps" are bounded by the concept of width, and instead of being executed by a low-level policy, they are managed by polynomial-time IW(k) procedures. In principle, these two ideas can be combined. Indeed, symbolic methods for learning hierarchical policies based on width-based considerations have also been developed (Drexler, Seipp, and Geffner 2023).

Conclusion

We have demonstrated that DRL methods can effectively learn the common subgoal structures of entire collections of planning problems, enabling them to be solved efficiently through greedy sequences of IW(k) searches. While the resulting decompositions are represented by neural networks rather than symbolic rules, they can often be interpreted logically. The experimental results highlight the strength of this approach, where decompositions learned from small instances are applied to solve much larger instances using sequences of linear and quadratic IW(1) and IW(2) searches. This approach leverages the concepts of width, sketches, and the logic of GNNs, and its limitations can be understood and potentially addressed within this framework.

Two specific challenges for future work include: 1) incorporating the cost of subproblems to reduce plan lengths while still learning meaningful decompositions, and 2) developing two-level hierarchical policies to eliminate the need for IW(k) searches in subproblems altogether.

Acknowledgements

Thanks to Simon Ståhlberg for providing the code of Muninn. The research has been supported by the Alexander von Humboldt Foundation with funds from the Federal Ministry for Education and Research, by the European Research Council (ERC), Grant agreement No. No 885107, and by the Excellence Strategy of the Federal Government and the Länder, Germany.

References

Bonet, B.; Frances, G.; and Geffner, H. 2019. Learning features and abstract actions for computing generalized plans. In *Proc. AAAI*, 2703–2710.

Bonet, B.; and Geffner, H. 2018. Features, Projections, and Representation Change for Generalized Planning. In *Proc. IJCAI*, 4667–4673.

Bonet, B.; and Geffner, H. 2021. General Policies, Representations, and Planning Width. In *Proc. AAAI*.

Bonet, B.; and Geffner, H. 2024. General policies, subgoal structure, and planning width. *Journal of Artificial Intelligence Research*, 80: 475–516.

De Giacomo, G.; Iocchi, L.; Favorito, M.; and Patrizi, F. 2020. Restraining bolts for reinforcement learning agents. In *Proc. AAAI*, 13659–13662.

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of artificial intelligence research*, 13: 227–303.

Drexler, D.; Seipp, J.; and Geffner, H. 2022. Learning sketches for decomposing planning problems into subproblems of bounded width. In *Proc. Int. Conf. on Automated Planning and Scheduling*, 62–70.

Drexler, D.; Seipp, J.; and Geffner, H. 2023. Learning Hierarchical Policies by Iteratively Reducing the Width of Sketch Rules. In *Proc. 20th Int. Conf. Principles of Knowledge Representation and Reasoning*, 208–218.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. HTN Planning: Complexity and Expressivity. In *Proc. AAAI-94*, 1123–1123.

Frances, G.; Bonet, B.; and Geffner, H. 2021. Learning general planning policies from small examples without supervision. In *Proc. AAAI*, 11801–11808.

Francès, G.; Ramírez, M.; Lipovetzky, N.; and Geffner, H. 2017. Purely Declarative Action Representations are Overrated: Classical Planning with Simulators. In *Proc. IJCAI 2017*, 4294–4301.

Georgievski, I.; and Aiello, M. 2015. HTN planning: Overview, comparison, and beyond. *Artificial Intelligence*, 222: 124–156.

Hamilton, W. L. 2020. Graph representation learning. *Synth. Lect. on Artif. Intell. Mach. Learn.*, 14(3): 1–159.

Hogg, C.; Munoz-Avila, H.; and Kuter, U. 2008. HTN-MAKER: Learning HTNs with Minimal Additional Knowledge Engineering Required. In *Proc. AAAI*, 950–956.

Horčík, R.; and Šír, G. 2024. Expressiveness of Graph Neural Networks in Planning Domains. In *Proc. ICAPS*, 281–289.

Icarte, R. T.; Klassen, T.; Valenzano, R.; and McIlraith, S. 2018. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*, 2107–2116.

Junyent, M.; Gómez, V.; and Jonsson, A. 2021. Hierarchical width-based planning and learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, 519–527.

Kingma, D.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*. San Diego, CA, USA.

Knoblock, C. A. 1994. Automatically generating abstractions for planning. *Artificial intelligence*, 68(2): 243–302.

Kulkarni, T. D.; Narasimhan, K.; Saeedi, A.; and Tenenbaum, J. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Advances in neural information processing systems*, 29.

Lipovetzky, N.; and Geffner, H. 2012. Width and serialization of classical planning problems. In *Proc. ECAI*, 540–545.

Lipovetzky, N.; and Geffner, H. 2017. Best-First Width Search: Exploration and Exploitation in Classical Planning. In *Proc. AAAI 2017*, 3590–3596.

Machado, M. C.; Bellemare, M. G.; and Bowling, M. 2017. A laplacian framework for option discovery in reinforcement learning. In *International Conference on Machine Learning*, 2295–2304.

McGovern, A.; and Barto, A. G. 2001. Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density. In *Proc. Int. International Conference on Machine Learning*, 361–368.

Nau, D.; Cao, Y.; Lotem, A.; and Munoz-Avila, H. 1999. SHOP: Simple hierarchical ordered planner. In *Proc. IJCAI*, 968–973.

Park, S.; Ghosh, D.; Eysenbach, B.; and Levine, S. 2024. Hiql: Offline goal-conditioned rl with latent states as actions. *Advances in Neural Information Processing Systems*, 36.

Parr, R.; and Russell, S. 1997. Reinforcement learning with hierarchies of machines. *Advances in neural information processing systems*, 10.

Sacerdoti, E. D. 1974. Planning in a hierarchy of abstraction spaces. *Artificial intelligence*, 5(2): 115–135.

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2008. The graph neural network model. *IEEE transactions on neural networks*, 20(1): 61–80.

Singh, S.; Lewis, R. L.; Barto, A. G.; and Sorg, J. 2010. Intrinsically motivated reinforcement learning: An evolutionary perspective. *IEEE Transactions on Autonomous Mental Development*, 2(2): 70–82.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022. Learning General Optimal Policies with Graph Neural Networks: Expressive Power, Transparency, and Limits. In *Proc. ICAPS*, 629–637.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2023. Learning general policies with policy gradient methods. In *Proc. KR*, 647–657.

Sutton, R.; and Barto, A. 1998. *Introduction to Reinforcement Learning*. MIT Press.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211.

Zheng, Z.; Oh, J.; Hessel, M.; Xu, Z.; Kroiss, M.; Van Hasselt, H.; Silver, D.; and Singh, S. 2020. What can learned intrinsic rewards capture? In *International Conference on Machine Learning*, 11436–11446. PMLR.

Zhuo, H. H.; Hu, D. H.; Hogg, C.; Yang, Q.; and Munoz-Avila, H. 2009. Learning HTN method preconditions and action models from partial observations. In *Proc. IJCAI*.