Vanir: Learning and Executing Width-based Hierarchical Policies

Dominik Drexler
Linköping University, Linköping, Sweden
dominik.drexler@liu.se

Abstract

Vanir describes one of the family trees of the Norse Gods in Nordic mythology. Likewise, our planner uses trees where each node represents a subtask with polynomial complexity based on the problem width, and the children of each node represent a decomposition into simpler subtasks.

1 Introduction

The foundation of Vanir is the mathematical notion of planning width (Lipovetzky and Geffner 2012) and the language of policy sketches (Bonet and Geffner 2021). Vanir consists of two components: a learner and a planner (Drexler, Seipp, and Geffner 2023). The learner takes as input a collection of classical planning instances and outputs a hierarchical policy which is a single-rooted tree where the children of each node represents a policy sketch that decomposes the problems at the root into simpler subproblems measured by the notion of planning width. The planner takes as input a hierarchical policy and a problem instance and exploits the hierarchical policy to reach the goal without searching.

2 Learning Hierarchical Policies

A hierarchical policy Π is a rooted tree (Drexler, Seipp, and Geffner 2023). Every node n in the tree represents a class of problems $\mathcal{Q}_n$ and a sketch rule $r(n)$. The child nodes n' of a node n form a problem decomposition in the form a sketch $R(n)$ decreasing the width of $\mathcal{Q}_n$. There is a single root node n_r that represents the class of problems $\mathcal{Q}_{n_r} = \mathcal{Q}$ and the sketch rule $r(n_r) = \{\neg G\} \mapsto \{G\}$ where G is a Boolean feature that is true in every goal state of $P \in Q_{n_r}$ and false otherwise. The width of $\mathcal{Q}_{n_r}$ is usually unbounded.

Vanir learns a hierarchical policy Π for a class of problems $\mathcal{Q}$ by iteratively reducing the width of subproblems (Drexler, Seipp, and Geffner 2023). Initially, the tree of the hierarchical policy Π contains a single root node n_r. Then, Vanir iteratively decomposes the class of problems $\mathcal{Q}_n$ of a leaf node n into children n' with classes $\mathcal{Q}_{n'}$ while decreasing the width by 1 except for the first problem decomposition, which usually goes from unbounded width to width 2. The resulting tree has a depth of at most 3.

Drexler, Seipp, and Geffner (2023) addressed the problem of learning a sketch R for a class of problems $\mathcal{Q}$ by finding the best truth assignment of a propositional theory. Their method builds on top of a previous method of learning sketches (Drexler, Seipp, and Geffner 2022) with a slight change in the semantics that considers sketch rules independent of each other and is required for hierarchical execution. Next, we show the propositional theory and discuss implementational details.

2.1 Encoding for Learning Sketches

From a set of training instances $\mathcal{P} \subseteq \mathcal{Q}$, we derive a set of domain-general features $\mathcal{F}$ using the DLPlan library (Drexler, Francès, and Seipp 2022). Using the training instances $\mathcal{P}$, the set of features $\mathcal{F}$, width k, and maximum number of sketch rules m, we construct the propositional theory $T(\mathcal{P}, \mathcal{F}, k, m)$ as defined by Drexler, Seipp, and Geffner (2023). To describe its variables, we use the following symbols: s, s' range over all states in the training set, f ranges over all features in $\mathcal{F}$, v ranges over all feature conditions or '?', v' ranges over all feature effects or '?', and i ranges over all rule indices $1, \ldots, m$. The propositional variables in $T(\mathcal{P}, \mathcal{F}, k, m)$ are:

- $\mathit{select}(f)$: feature f is included in Φ
- $\mathit{cond}(i, f, v)$: rule i has condition v for f
- $\mathit{eff}(i, f, v')$: rule i has effect v' for f
- $\mathit{subgoal}(s, t, i)$: rule i has subgoal t of width $\leq k$ in s
- $\mathit{sat\_rule}(s, s', i)$: pair $[s, s']$ is compatible with rule i
- $\mathit{sat\_cond}(s, i)$: state s satisfies conditions of rule i
- $r\text{-}\mathit{reach}(s)$: state s is in $S_R(P)$

To describe the constraints in $T(\mathcal{P}, \mathcal{F}, k, m)$, we use the same symbols as above, with the difference that s now only ranges over all *alive* states. In addition, t ranges over subgoal tuples with width at most k in s, $\mathit{dist}(s, s')$ is the shortest distance from s to s', $\mathit{dist}(s, t)$ is the length of an admissible chain that ends in subgoal tuple t for state s, $S^*(s, t)$ are all states that result from applying optimal plans from $P[s, t]$ in s. The constraints are

C1 $\mathit{cond}(i, f, v) \vee \mathit{eff}(i, f, v') \rightarrow \mathit{select}(f)$, unique v, v',
C2 $r\text{-}\mathit{reach}(s) \rightarrow \vee_i \mathit{sat\_cond}(s, i)$,
C3 $r\text{-}\mathit{reach}(s) \wedge \mathit{sat\_cond}(s, i) \rightarrow \vee_t \mathit{subgoal}(s, t, i)$,
C4 $r\text{-}\mathit{reach}(s_0)$ for initial state s_0,
C5 $r\text{-}\mathit{reach}(s) \wedge \mathit{sat\_rule}(s, s', i) \rightarrow r\text{-}\mathit{reach}(s')$,
C6 $\mathit{subgoal}(s, t, i) \rightarrow \wedge_{s' \in S^*(s,t)} \mathit{sat\_rule}(s, s', i)$,

C7 $sat\_rule(s, s', i) \rightarrow \vee_{dist(s,t) \leq dist(s,s')} subgoal(s, t, i)$,

C8 $sat\_rule(s, s', i) \leftrightarrow [s, s']$ compatible with rule i,

C9 $sat\_cond(s, i) \leftrightarrow s$ satisfies conditions of rule i, and

C10 the collection of m rules is acyclic.

C11 $\min \sum_{f \in \mathcal{F}} \mathbb{1}_{\{select(f)\}} \cdot complexity(f)$

2.2 Incremental Learning

The encoding scales polynomially in the size of the instances and exponentially in k. Hence, using small instances can make learning substantially faster. To automate the selection of small instances from all training instances $\mathcal{P}$ to form a propositional theory $T(\mathcal{P}, \mathcal{F}, k, m)$, we use the same method from previous work on learning policy sketches (Drexler, Seipp, and Geffner 2022). The method orders the training instances increasingly by their number of states and, in each step, adds the smallest instance to the propositional theory where the current sketch fails to find a new sketch R until R solves all training instances. When adding an instance that is larger than all others in the training set, it removes all smaller instances. We encode the propositional theory as an answer set program (ASP) and solve it using the Clingo solver (Gebser et al. 2019).

2.3 Data Augmentation

Every problem instance $P \in \mathcal{P}$ comes with a single initial state s_0. However, in many domains several states s' reachable from the initial s_0 can be the initial state s'_0 for another instance $P' \in \mathcal{P}$. Therefore, from $\mathcal{P}$ we generate a closed set of instances $\mathcal{P}^*$ such that every $P \in \mathcal{P}$ is also in $\mathcal{P}^*$ and additionally if s' is reachable from the initial state $s_0 \in P$ then the problem P' with initial state s' is also in $\mathcal{P}^*$.

2.4 Exploiting Indistinguishability of Constraints

We exploit indistinguishable constraints to reduce the encoding size (Francès, Bonet, and Geffner 2021). The method defines an equivalence relation over the state pairs where two state pairs $[s, s']$ and $[t, t']$ are in the same equivalence class iff they are indistinguishable by feature conditions evaluated in s and t, and indistinguishable by feature changes evaluated in $[s, s']$ and $[t, t']$.

2.5 Feature Complexity

The feature complexity is defined as the number of applied grammar rules. When the instances are extremely small, for example, if the instance consists of single objects of a specific type, then Boolean features that map to true or false often suffice to find a sketch. However, the generalization capabilities of Boolean features to larger instances are usually smaller in comparison to their numerical counterpart. Hence, we penalize the selection of Boolean features with an additional cost of 1 which often allows generalization from much smaller instances.

3 Executing Hierarchical Policies

The execution of a hierarchical policy follows its hierarchical structure and requires no search at all.

3.1 Execution of Hierarchical Policies

It follows the definition of the execution of a hierarchical policy Π by Drexler, Seipp, and Geffner (2023). A hierarchical policy Π is executed on a problem $P \in Q$ by making use of a stack with entries $\langle s, n \rangle$, where s is a state from P and n is a node in Π. The execution also tracks a current state s'. During the execution of the policy, the entries $\langle s, n \rangle$ in the stack at any point describe the subproblems $P[s, G_r(s)]$ that are being solved, where $r = r(n)$. Initially, the stack contains the pair $\langle s_0, n_r \rangle$ where s_0 is the initial state of P and n_r is the root node of Π, and the current state is $s' := s_0$. Then the execution considers three cases iteratively until the stack becomes empty:

1. If the top stack entry is $\langle s, n \rangle$, the current state s' is not in $G_r(s)$ for $r = r(n)$, and n is not a leaf node, a (any) child n' of n is chosen with rule $r(n')$ whose condition is true in s'. Then the entry $\langle s', n' \rangle$ is pushed onto the stack without changing the current state s'.
2. If the top stack entry is $\langle s, n \rangle$, s' is not in $G_r(s)$ for $r = r(n)$, and n is a leaf node, then an applicable action a in s', that will be equal to s, is selected and applied in s' for obtaining a state s'' that is in $G_r(s)$ (this must be possible because subproblem $P[s, G_r(s)]$ has width 0). The current state is set to s''.
3. If the top stack entry is $\langle s, n \rangle$ and s' is $G_r(s)$ for $r = r(n)$, then the entry is popped from the stack without changing the current state s'.

4 Analysis of the Competition Results

In this section, we analyze the results of the competition. First, we present whether the learning succeeded or failed and provide the reason if it failed for each planning domain that was used in the competition. Second, we describe implementational issues together with their fixes and further improvements. Last, we present the competition results with the reevaluation of Vanir after fixing the issues.

4.1 Per-domain Analysis of the Learning Step

In the competition, there were 10 domains used from previous IPCs. Vanir learned domain control knowledge in Ferry, Miconic, and Rovers. It follows a summary of the learning step on each of the domains.

- Blocksworld: The learner ran out of memory.
- Childsnack: The learner ran out of time.
- Ferry: The learner returned a valid hierarchical policy for the training instances.
- Floortile: The learner ran out of time.
- Miconic: The learner returned a valid hierarchical policy for the training instances.
- Rovers: The learner returned an "incomplete" hierarchical policy for the training instances. We say that a hierarchical policy is *incomplete* for a class of problems $\mathcal{Q}$ if constraints 1 and 2 of a valid hierarchical policy for $\mathcal{Q}$ are satisfied. However, the width $w(\mathcal{Q}_n)$ of a leaf node n can be greater than 0. In this domain, the decomposition at a

	Baselines			Competitors				
	LAMA	FDSS	SMAC	ASNets	GOFAI	HUZAR	Muninn	Vanir
Blocksworld	47.9	49.4	31.5	4.6	46.4	39.3	40.6	**47.9**
Childsnack	26.2	35.4	20.2	0.0	**26.5**	22.0	11.0	26.2
Ferry	64.0	61.5	64.4	–	58.5	58.7	42.1	**76.3**
Floortile	12.0	22.7	24.7	–	**34.4**	21.3	0.0	11.0
Miconic	84.4	89.6	52.3	7.2	**81.4**	72.4	30.0	75.2
Rovers	66.8	64.0	58.1	6.5	54.4	60.0	14.2	**66.8**
Satellite	87.3	88.7	71.0	–	74.0	79.9	16.0	**87.4**
Sokoban	37.7	39.0	30.8	0.0	**38.4**	28.1	24.3	37.7
Spanner	30.0	60.7	30.0	8.9	30.0	30.0	**32.0**	30.0
Transport	61.4	63.0	62.7	2.0	64.5	55.4	16.2	61.4
Sum	517.6	574.1	445.7	29.1	508.5	467.0	226.3	**519.9**

Table 1: Planning step: the quality scores from the competition after the reevaluation of Vanir.

leaf failed because no sketch exists for the generated pool of features.

- Satellite: The learner ran out of memory.
- Sokoban: The learner returned an incomplete hierarchical policy that is valid on a small subset of training instances. This was possible because we removed large instances for which solving the logic program is too complicated.
- Spanner: The learner ran out of time.
- Transport: The learner ran out of time.

4.2 Post-competition Fixes and Improvements

We observed the following issues in the submission of Vanir to the competition.

- In the competition, the learning system spawned 32 threads to solve the propositional theory, but only 1 CPU core was available. After the competition, we reduced the number of threads to 1.
- In the competition, the planning system crashed before running the backoff planner LAMA (Richter and Westphal 2010) in some cases where no domain control knowledge was generated. After the competition, we allow the backoff planner to run if no domain control knowledge was generated.

In addition to the issues above, we plan to integrate the following improvements in the future.

- For incomplete hierarchical policies such as the one learned in rovers, running IW(k) search in a leaf node n with width $w(\mathcal{Q}_n)$ equal to k is possible. This change allows also using incomplete hierarchical policies for solving planning tasks.

4.3 Reevaluation of the Planning Step

Table 1 shows the results after reevaluation of Vanir after fixing the issues from above. We used the same resource limits and ran our experiments on the same hardware, i.e., 24 hours and 32 GiB memory limits for learning, and 30 minutes and 8 GiB memory limits for planning. Note that Vanir learns a valid hierarchical policy for the training instances in Ferry and Miconic. In Ferry, we observe that Vanir receives a much higher quality score (76.3) than the other competitors. In Miconic, Vanir solves all instances but produces worse plans and obtains a slightly lower quality score (75.2) than the other competitors. In the other domains, we get similar results as the baseline LAMA because this is the backoff planner. Finally, Vanir receives the highest total quality score among all competitors (519.9).

References

Bonet, B., and Geffner, H. 2021. General policies, representations, and planning width. In *Proc. AAAI 2021*, 11764–11773.

Drexler, D.; Francès, G.; and Seipp, J. 2022. Description logics state features for planning (DLPlan). https://doi.org/10.5281/zenodo.5826139.

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

Drexler, D.; Seipp, J.; and Geffner, H. 2023. Learning hierarchical policies by iteratively reducing the width of sketch rules. In *Proc. KR 2023*.

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

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2019. Multi-shot ASP solving with clingo. *Theory and Practice of Logic Programming* 19:27–82.

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

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