Leveraging Demonstrations for Learning the Structure and Parameters of Hierarchical Task Networks

Philippe Hérail, Arthur Bit-Monnot

LAAS-CNRS, Université de Toulouse, INSA, CNRS, Toulouse, France

philippe.herail@laas.fr, abitmonnot@laas.fr

Abstract

Hierarchical Task Networks (HTNs) are a common formalism for automated planning, allowing to leverage the hierarchical structure of many activities. While HTNs have been used in many practical applications, building a complete and efficient HTN model remains a difficult and mostly manual task.

In this paper, we present an algorithm for learning such hierarchical models from a set of demonstrations. Given an initial vocabulary of tasks and accompanying demonstrations of possible ways to achieve them, we present how each task can be associated with a set of methods capturing the knowledge of how to achieve it. We focus on the algorithms used to learn the structure of the model and to efficiently parameterize it, as well as an evaluation in terms of planning performance.

Introduction

Hierarchical Task Networks (HTNs) (Erol, Hendler, and Nau 1994) are an approach to automated planning that combine a declarative action-based model for describing the primitive actions achievable by a system with procedural knowledge describing how those primitives can be combined to achieve high-level tasks. Even though these hierarchical models are a formalism that allows to plan more efficiently while remaining interpretable by human engineers, it is cumbersome to design such models from scratch. This difficulty stems from the quickly exploding number of possible contexts that need to be considered when carrying out even basic tasks in a simple environment. To address this issue, we intend to allow the agent to learn such HTN models from previously observed execution traces, and in particular the ones resulting from a tutor's demonstration.

The goal of such a learning system would be to be able to solve any previously demonstrated tasks through a solution of at least equivalent quality to the demonstrated one. It should also be able to generalize the demonstrations to solve new unseen tasks, or previously demonstrated tasks in a new environment. This is done by learning, for any given task in the considered domain, a set of methods that achieve the high-level objectives associated to the task. This set of methods should cover all possible ways of achieving this task with the exception of clearly suboptimal ways. Any

method should be associated with a validity scope that defines whether it is *applicable* in a given state. When applicable, it should achieve the task.

Intuitively, if a learned hierarchical planning model has these desirable properties, an automated planner facing a task to achieve could greedily pick any applicable method and have the guarantee that it will fulfill the objectives associated to the task. While this might lead to suboptimal behavior, classical search mechanism would allow an automated planner to derive an optimal solution.

The objective of this paper, extending our previous work (Hérail and Bit-Monnot 2022), is to present a method for building parameterized hierarchical planning models based on a set of demonstrations, each demonstration associating a high-level task to an *execution trace*: a sequence of primitive actions achieving it.

Related Work

Over the years, several approaches have been developed to learn hierarchical planning models.

HTN-MAKER (Hogg, Muñoz-Avila, and Kuter 2008) and HTNLearn (Zhuo, Muñoz-Avila, and Yang 2014) are both approaches aiming at learning parameterized HTNs from demonstrations, the latter of which was later extended to support partial and disordered input traces (Zhuo, Peng, and Kambhampati 2019). Both these methods require as input, in addition to the demonstrations, demonstrated tasks and subtasks annotated with preconditions and effects. The first method learns the models through goal regression while the second formulates the problem as one of maximal constraint satisfaction.

Other approaches exist that do not need subtasks as input, such as the work by Li et al. (2014) or CircuitHTN (Chen et al. 2021). These methods, however, learn HTNs structures without parameters and method preconditions, limiting their ability to generalize to new environments.

Recently, the learning of Hierarchical Goal Networks (HGNs) structure instead of HTNs, has been proposed as a preliminary work by Fine-Morris and Muñoz-Avila (2019), leveraging a vector representation of the states and unsupervised learning procedures to learn such networks while limiting the burden of annotating demonstration data. The most recent work in this area focuses on numeric goals and preconditions (Fine-Morris et al. 2022)

Due to their similarities with HTNs, some work aiming at learning grammars is relevant and in particular the work on learning Combinatory Categorial Grammars (CCGs) for plan and goal recognition (Geib and Goldman 2011; Kantharaju, Ontañón, and Geib 2019). While the learned CCGs are not practically usable for planning, the authors propose several ideas for extracting interesting patterns from a set of execution traces.

Learning Problem

Hierarchical Planning Model

We define a hierarchical planning model H as a lifted HTN structure which can be written as a tuple $H = (T, A, M)$ where T is a set of abstract tasks, A a set of primitive actions and M a set of possible methods decomposing the tasks $t \in T$ into subtasks $\{t_d \mid t_d \in \{T \cup A\}\}$. Figure 1 shows a simple task hierarchy as an illustration.

A primitive task (or action) $a \in A$ models the basic acting capabilities of the agent, and represents directly executable primitives. They are represented using an identifying symbol and a set of parameters, such as $a = \text{action\_name}(\text{arg}_1, \dots, \text{arg}_n)$. Actions are associated with preconditions and effects that allow verifying the validity of a plan.

An abstract (or non-primitive) task $t \in T$ is associated with a set of methods M_t that allow decomposing it and possible postconditions representing the predicates that must hold after executing t for it to be considered a success. Similar to actions, they are represented using an identifying symbol and a set of arguments.

A method $m \in M_t$ is a tuple $m = (Pre_m, N_m)$, where Pre_m are the preconditions of the method, and N_m is a sequence of subtasks in $\{T \cup A\}$, representing a possible decomposition of t. This totally-ordered task network represents a way to achieve the task t and is only applicable in the current state if its preconditions Pre_m hold.

For a given a planning domain H, a planning problem is an initial task network N_p, representing the activity that must be carried out and a initial state s_0 described by a set of boolean state variables. We consider that at any instant, the current state s is fully observable and that it only evolves when a primitive action is executed (i.e. there are no exogenous events).

Learning of a Planning Model

Inputs to the Learning Problem For the learning problem itself, we consider as input a fixed set A of primitive actions as well as a vocabulary of non-primitive tasks T_I. For each primitive action, the learner knows its symbol and parameters but is *not* given any knowledge of its preconditions of effects.

For each task $t_I \in T_I$, the agent is given a set D_{t_I} of demonstration traces from the tutor. Each trace $d \in D_{t_I}$ is an alternating sequence of states and tasks (either primitive or non-primitive), starting from a given initial state and ending in a final state in which the task t_I has been successfully achieved. d is considered optimal and maximally abstract
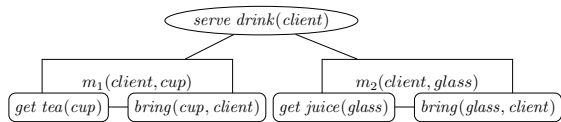

Figure 1: Structure of a simple task hierarchy with two alternative methods. Method preconditions omitted for clarity.

with regard to the initial task vocabulary: for every demonstrated task, no other more abstract task from the initial vocabulary T_I may be used to abstract a subsequence of d, and each demonstration is optimal according to a chosen metric. For a case where actions are uniform in cost, one may naturally consider the total number of primitive actions required to achieve t_I as the optimality metric.

Learner Objectives The primary objective of the learning process is to produce a hierarchical model that is capable of solving planning problems that were not part of the demonstrated set. This completeness property is intrinsic to the model and defines whether the model is theoretically capable of solving any possible problem of the domain at hand.

The quality of a planning model H is however tightly coupled with the ability of an automated planner to exploit it to quickly derive solution plans. In particular, for any given planner we are aiming at maximizing the efficiency of the planner for solving a problem given H, which is typically measured as the runtime of the planner. This leads us to define the *coverage* of a learned model as the ratio of solved problems by a given planner under computational limits.

Approach to Model Learning

Requirements of Model Selection

Let us now give an initial intuition about the shape of models that could be learned and the implication for the learning process. Figure 2 presents several possible models (figures 2b-2e) that could be generated based on two example sequences (figure 2a).

The first one (2b) allows the choice of any of the four primitive actions $\{a, b, c, d\}$, each placed in a specific method. This model relies on a recursive call to t to re-propose the same choice until the task's postconditions are achieved. While this model allows building any sequence of actions it does not help the agent towards a meaningful sequence based on demonstrations. The second model (2c) takes the opposite approach and records each known trace into a method. This model is obviously strongly tied to the demonstration set and would fail to generalize to new problems. In between these two extremes, we have the models (2d) and (2e) that present different options to abstract common subsequences. The former encodes the repeated $a\ b$ sequence in a single method and relies on the recursive call to complete the sequence. The latter delays the choice between c and d to after the execution of a and b, using a synthetic task t_s.

These four models are just a handful of examples among the many possible models that could be generated. Denoting as Θ the set of possible models, the objective of a learning

$$t \rightarrow a\ b\ c$$
$$t \rightarrow a\ b\ d$$

(a) Available demonstrations, showing that t was once achieved with the $a\ b\ c$ action sequence and once with the $a\ b\ d$ action sequence. Intermediate states (also available in the demonstration) are omitted.
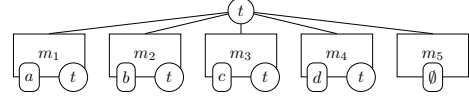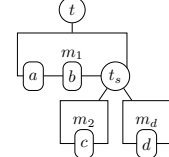
(b) Generic model where the planner might pick any of the primitive actions and rely on the recursive call to t to continue if needed.
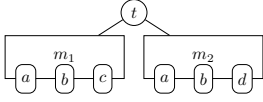
(c) Model where each demonstration is fully encoded into a dedicated method.
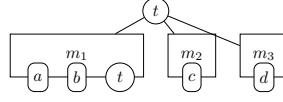
(d) Intermediate model the common $a\ b$ sequence is grouped. It relies on the recursive call to t in m_1 to produce a full sequence.

(e) Model where the $a\ b$ sequence is shared, requiring a synthetic task t_s

Figure 2: Illustration of the possible structures of the learned model for a simple learning task with two demonstration of how to perform a task t. Note that for conciseness the parameters and preconditions or the task and methods are omitted.

system is to find, or at least approach, the optimal model $\theta^* \in \Theta$

$$\theta^* = \underset{\theta \in \Theta}{\arg\min}\ cost(\theta)$$

where $cost(\theta)$ is a function that measures the cost of a particular model and should typically account for the size of the model as well as its capacity to solve both demonstrated and unseen problems. With this in mind we now turn our attention to the characterization of the set of possible models Θ. In a later section, we will propose a cost function to evaluate the models.

Generation of Candidate Planning Models

At a high level, the goal of the learning problem is to generate a model where some subtasks group together common behaviors, with a sensible parameterization of methods depending on the current task, as well as reasonable preconditions to limit the search effort of the planning engine.

The overview of our process for achieving this goal from a set D of demonstrations is presented in algorithm 1.

Algorithm 1 Planning Model Search

1: $h \leftarrow$ GENERATE BASE MODEL
2: **while** QUALITY(h) improves **do**
3: $\quad H_c \leftarrow$ GEN CANDIDATE MODELS STRUCTS(h, D)
4: $\quad$ **for all** $h_c \in H_c$ **do**
5: $\quad\quad h_c \leftarrow$ EXTRACT MODEL PARAMETERS(h_c, D)
6: $\quad h \leftarrow$ FIND BEST MODEL($H_c \cup \{h\}$)

Structure Generation

Model structures are generated through the exploration of the neighborhood of the current best model, using the following operators.

In order to quickly progress towards a useful structure, one of the operators is implemented using a procedure similar to the one described in HTN-MAKER (Hogg,

Muñoz-Avila, and Kuter 2008). As we consider only optimal demonstrations and totally ordered subtasks, we do not need to consider a task's postconditions, and consider that each subsequence going up to the end of a demonstration of a task t is a way to achieve t. Furthermore, as the parameters will be extracted in the next step, we only consider the demonstration as a sequence of task *symbols*, removing the need for complex method subsumption detection techniques, replacing it with a removal of duplicate symbol sequences.

While this procedure does provide methods that will always be useful in some case for achieving the task for which it was learned, it does not provide multiple hierarchy levels. Therefore, we designed a new operator to allow grouping some behaviors into new tasks. In order to focus the search on relevant parts of the search space, we use frequent pattern mining to generate the candidate subtasks. Frequent patterns are extracted in a greedy fashion, using a procedure inspired by the GoKrimp algorithm (Lam et al. 2014), which is based on the Minimum Description Length (MDL) (Grünwald 1996) concept, incrementally finding the patterns that most compress the sequence dataset.

In our case, we extract patterns from a set of demonstrations D through the function described in algorithm 2, with three parameters $l, k, n_c \in \mathbb{N}$. We consider again demonstrations as sequences of task symbols, and try to extract patterns as regular expressions (regexps). An example of mapping from a a regexp pattern to a hierarchical representation is presented in figure 3b. $len(p)$ is defined as the number of symbols in p, excluding any regexp operator.

Patterns are extracted by incrementally building a set P_f of patterns ($|P_f| \leq k$), each iteration extracting the most compressing pattern p such that $len(p) \leq l$ and then replacing all the matches of p in D (see an example figure 3a).

The pattern generation function is detailed in algorithm 3. We define the COMPRESSED SIZE(D, p) function as the function that returns the size of the demonstration set D compressed using pattern p as in the work of Lam et al. (2014), and the function CONCAT(p_1, p_2) as the one that generates a new pattern by appending p_2 to p_1.

Algorithm 2 GENERATE PATTERNS(D, k, l, n_c)

```
1: P_f ← ∅
2: while |P_f| ≤ k do
3:    p ← MOST COMPRESSING PATTERN(D, l, n_c)
4:    P_f ← P_f ∪ {p}
5:    SUBSTITUTE PATTERN(D, p)
6: return P_f
```


$$D: \begin{cases} d_1: \mathbf{a}\ \mathbf{b}\ c\ d \\ d_2: \mathbf{a}\ \mathbf{b}\ d \\ d_3: \mathbf{a}\ \mathbf{b}\ c \\ d_4: d\ \mathbf{a}\ \mathbf{a}\ \mathbf{b} \end{cases} \underset{p}{\mapsto} \begin{cases} d_1: \mathbf{p}\ c\ d \\ d_2: \mathbf{p}\ d \\ d_3: \mathbf{p}\ c \\ d_4: d\ \mathbf{p} \end{cases}$$

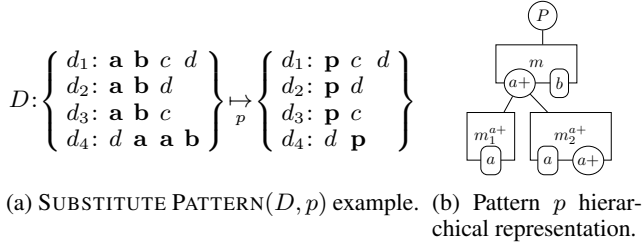(a) SUBSTITUTE PATTERN(D, p) example. (b) Pattern p hierarchical representation.

Figure 3: Pattern substitution and hierarchical representation example for $p = a+b$.

Pattern generation starts by initializing a set P with the task symbols in the demonstrations, the previously generated patterns and a set of choice patterns, as well as a set R containing the standard regexp operators $\{?, *, +\}$. The choice patterns are generated by taking a random set of n_c existing patterns and combining them together using the $|$ regexp operator. P is used to initialize the set of candidate patterns $\mathcal{P}$.

The set of candidate patterns is then extended by adding possible regexp patterns (line 8), and this new set is then extended again by adding potential following task or patterns (line 11). The best compressing pattern of the set is then kept and the process is repeated until either the compression stops improving anymore or the pattern length reaches the limit l.

Algorithm 3 MOST COMPRESSING PATTERN(D, l, n_c)

```
1: P_a ← TASK SYMBOLS(D)
2: P_p ← EXISTING PATTERNS(D)
3: P_c ← GENERATE CHOICE PATTERNS(P_p, n_c)
4: P ← P_a ∪ P_p ∪ P_c
5: 𝒫 ← P, R ← {?, *, +}
6: repeat
7:    𝒫' ← 𝒫
8:    for all (p, r) ∈ 𝒫 × R do
9:       𝒫' ← 𝒫' ∪ {CONCAT(p, r)}
10:   𝒫'' ← 𝒫'
11:   for all (p, p_e) ∈ 𝒫' × P do
12:      𝒫'' ← 𝒫'' ∪ {CONCAT(p, p_e)}
13:   p* ← arg min_{p∈𝒫''} COMPRESSED SIZE(D, p)
14:   𝒫 ← {p*}
15: until COMPRESSED SIZE(D, p*) stops improving OR
       len(p*) ≥ l
16: return p*
```

Model Structure Demonstration Matching

In order to use the demonstrations to parameterize a candidate model structure C, we need to find a hierarchical mapping between a given demonstration d and a top level task t_d, through a decomposition of this task as defined in C. This matching is done through an adaptation of the technique using HTN planning for plan verification developed by Höller et al. (2021).

Model Parameterization

For each method in the model, we need to identify the parameters that should be passed to its subtasks. Furthermore, for synthetic tasks, we need to determine the tasks' parameters themselves.

The parameterization process is then as follows:

1. Identify a superset of the possible parameters for synthetic tasks and methods from the primitive tasks'.

2. Express the parameterization problem into one of MAX-SMT (Nieuwenhuis and Oliveras 2006) and solve it, with the objective of minimizing the number of parameters in the final model.

3. Add a new step to remove parameters not used to constrain methods instantiations, to reduce the search space during the planning phase.

Parameter Generation To extract the set of possible parameters for a model, we start by splitting it into sub-models, each comprised of a top-level non-primitive task, its methods and their direct subtasks (primitive and non-primitive). Such a sub-model is presented in Figure 4a, where t is a synthetic task for which two methods m_1 and m_2 were learned.

Algorithm 4 PROPAGATE ARGS UPWARDS(h_{sub})

```
1: for all m ∈ M_t do
2:    for all p̂ ∈ args(SUBTASKS(m)) do
3:       args(m) ← args(m) ∪ {p̂}
4:       if m ∉ M̂_p then
5:          p̂' ← (p, M̂_p ∪ {m})
6:          args(t_h) ← args(t_h) ∪ {p̂'}
```

To extract the arguments for the set of subhierarchies H_{subs} corresponding to an HTN H, arguments are propagated upwards from the subtasks to the top level task for each subhierarchy, as described in algorithm 4. Non-primitive subtasks in the subhierarchies are then updated to keep a consistent signature. The process is repeated until a fixed point is reached.

To enforce termination of the algorithm in the case of recursive task definitions, we augment each parameter p with the set of methods $\hat{M}_p$ it has been propagated through, defining $\hat{p} = (p, \hat{M}_p)$. We also define a condition to allow methods to act as filters (alg. 4 line 4), preventing methods from crossing twice a given method boundary. This behavior is illustrated in the example figure 4, where $*$ and $\dagger$ superscripts (associated with m_1 and m_2 respectively) are used to visualize the set $\hat{M}_p$ for each parameter p. In figure 4d, $A_{1,t}$ and
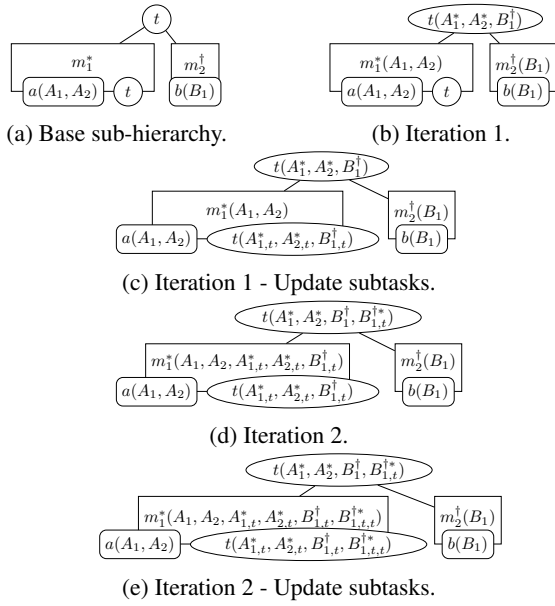
(a) Base sub-hierarchy.　　　　(b) Iteration 1.

(c) Iteration 1 - Update subtasks.

(d) Iteration 2.

(e) Iteration 2 - Update subtasks.

Figure 4: Parameter generation example.

(a) Demonstration set D.　　(b) Partial constraints extracted from d_3.

(c) Model simplified from D.

Figure 5: Parameter deduplication for the model figure 4e.

$A_{2,t}$ are not propagated upwards from m_1 as they are originating from A_1 and A_2 via method m_1.

Parameter Deduplication The parameter generation process aims at being exhaustive and covering all possible cases, regardless of the demonstrations traces at hand. In many cases, this results in many more parameters than actually needed to regenerate the traces. We cast the problem of deduplicating arguments as MAX-SMT with equality logic and uninterpreted functions, under the objective of maximizing the number of unifications permitted by the demonstration traces. At a high level, we have three kinds of constraints on the sub-hierarchies' arguments, extracted from the sub-hierarchies and examples:

- Structural hard constraints, to enforce consistency between a task reference definition (when considered as a top level one) and its occurrences as methods' subtasks. These constraints are defined as in the equation below:
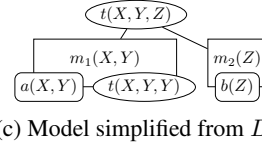
$$\forall (i,j) \in |\operatorname{args}(t)|, \operatorname{arg}_i(t) = \operatorname{arg}_j(t)$$
$$\Rightarrow \operatorname{arg}_i(t_{sub}) = \operatorname{arg}_j(t_{sub})$$

- Inequality hard constraints, added when the demonstrations show that two parameters must be distinct in a given instantiation.

- Equality soft constraints, extracted from all the examples that show two parameters that are identical in a given example.

Figure 5 shows some constraints that can be extracted through a combination of the structure of the hierarchy presented figure 4e and the information contained in a given set of demonstrations. Figure 5c shows the extracted model after simplification using the demonstrations set D presented figure 5a and solving the associated constraint system. Some constraints are presented in figure 5b as an example.

In a next step, we post-process the resulting model to remove task arguments that do not permit unifying part of the instantiation of its subtasks, or that do not provide instantiation information from some parent task in the hierarchy.

After extracting the tasks' and methods' parameters, preconditions are extracted by taking the lifted intersection of the states preceding the instantiation of each method m, considering only those that can be totally parameterized with the parameters of m.

Model Quality Evaluation

To evaluate the quality of the learned model, we use the metric described in our previous work (Hérail and Bit-Monnot 2022), based on the MDL principle (Grünwald 1996). This metric exploits data compression as a way to drive the model search towards abstracting redundant parts in the demonstrations. The quality of a given planning model H is defined as the weighted sum of the model size ($L_{\mathrm{model}}(H)$) and the demonstration dataset size, reconstructed using H ($L_{\mathrm{dem}}(\mathcal{D}|H)$), as presented below:

$$L(H, \mathcal{D}) = \alpha L_{\mathrm{model}}(H) + L_{\mathrm{dem}}(\mathcal{D}|H) \quad (1)$$

At this point we have generated several models, each of which has been parameterized. The metric above allows choosing a single model among the candidates. This selected model will be used as the baseline for the next iteration, where it will typically be extended with new tasks and methods, to cover more demonstrations traces. The process stops once this metric shows no improvement, meaning that the model modifications stop improving the abstractions of the demonstrations.

Learned Models Evaluation

In order to assess the validity of our approach we tested our learner on a variety of planning domains:

CHILDSNACK A simple domain whose reference model only contains one task with two alternative methods with only primitive subtasks. The learning set consists of 50 demonstrations of the serving task.

TRANSPORT A standard deliver-with-trucks scenario. The learning set for this domain consists of 20 demonstrations of the delivery task.

(a) CHILDSNACK domain.
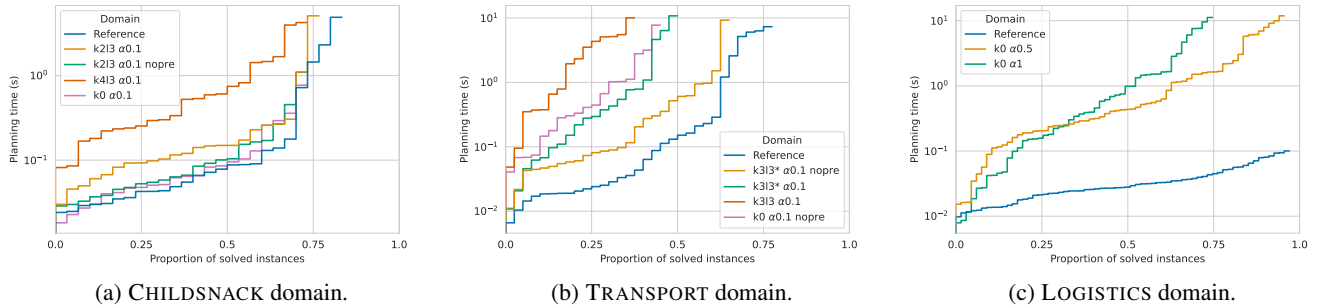
(b) TRANSPORT domain.

(c) LOGISTICS domain.

Figure 6: Planning time distribution for the different domains using the Lilotane planner.

LOGISTICS A variant of the TRANSPORT domain, extended with several cities connected via airplanes, only able to move to specific airport locations. This is the version that was used in the development of HTN-MAKER (Hogg, Muñoz-Avila, and Kuter 2008), modified to use typing instead of predicates. The learning set consists of 60 demonstrations of the delivery task.

For each domain, the models were learned from traces extracted from a random subset of the 2020 International Planning Competition (IPC) (Behnke, Höller, and Bercher 2021) instances or HTN-MAKER's training set when applicable. Trace selection was biased towards simple problems in order to reduce learning times, which were all well under one hour (wall-clock time) using 6 threads on a portable workstation, equipped with 32 GB of RAM and an Intel Core i7-10850H CPU (6/12 @ 2.70GHz). Demonstration traces used for learning only decompose one single task, while the test instances may require several instantiations of the learned tasks to be solved to be considered a success.

Figure 6 shows planning times for unseen problems for several learned models with the Lilotane (Schreiber 2021) planner. Each model is a result of a different parameterization of the learner, included in the graphs' legend: k is the maximal number of patterns to extract, l is their maximal length and α is the weight in the metric formula (eq. 1). A star, as in $k \cdot l \cdot *$, means that repeating patterns were allowed during the search while the *nopre* text indicates that no method preconditions were learned. For each domain, we include the performance of the Lilotane planner with a handwritten reference domain.

These results show that for the simplest domain, CHILDSNACK, the best learned models solve a number of instances similar to the reference IPC model. Analyzing the learned model structures (not presented here due to space constraints), they correspond closely to the IPC domain's, being identical for the model without abstraction (k=0). This shows our learning algorithm is able to extract alternative methods for tasks. The extracted preconditions do appear to be relevant and provide guidance to the planner while not over constraining the possibilities. A simple grouping of tasks appear beneficial, which we conjecture is from a reduction of the number of some methods' parameters.

Considering the TRANSPORT domain, a model providing

some abstraction stays in line with the performance of the reference model. However, it appears that the extracted preconditions are not specific enough to guide the search, leading to worse planning performance than without. The best learned model has a multi-level structure that is similar to the reference domain, showing the relevance of using a combination of pattern mining and regression through the actions.

The learned models on the LOGISTICS domain do not reach the performance of an expertly handcrafted model in terms of planning speed, but one is still able to solve as many instances, given the right learning parameters. Models with synthetic subtasks performed poorly and are not represented on the graph for clarity. This is likely due to reaching local minima early in the search process, leading to overly complex models poorly capturing the domain structure.

Conclusion & Future Work

The proposed approach shows that it is possible to learn parameterized HTN domains that are close to handcrafted ones in terms of planning performance on a subset of the domains from the IPC competition.

Compared to other approaches such as HTN-MAKER (Hogg, Muñoz-Avila, and Kuter 2008) or HTNLearn (Zhuo, Muñoz-Avila, and Yang 2014), the work needed from the tutor is limited, requiring only a limited number of demonstrations and no annotated intermediate tasks. These intermediate tasks are learned by abstracting common behaviors detected through frequent pattern mining, which our approach is then able to parameterize sensibly using the proposed MAX-SMT approach.

Furthermore, the nature of the search process allows to improve a learned model if new knowledge becomes available, by reusing the current model as the new starting point.

Despite the very limited information exploited by our algorithm, our experiments show that the learned models are already competitive with handwritten ones. Future work shall focus on improving the addition of the frequent patterns to the candidate structures during the search process in order to extract efficient ones in more complex domains. The parameter extraction process is also a planned improvement avenue, as considering the surrounding states during the refinement of a task could provide useful insights for detecting the most relevant parameters.

References

Behnke, G.; Höller, D.; and Bercher, P., eds. 2021. *Proceedings of the 10th International Planning Competition: Planner and Domain Abstracts – Hierarchical Task Network (HTN) Planning Track (IPC 2020)*.

Chen, K.; Srikanth, N. S.; Kent, D.; Ravichandar, H.; and Chernova, S. 2021. Learning Hierarchical Task Networks with Preferences from Unannotated Demonstrations. In *Proceedings of the 2020 Conference on Robot Learning*, 1572–1581. PMLR.

Erol, K.; Hendler, J.; and Nau, D. 1994. HTN Planning: Complexity and Expressivity. In *AAAI Conference on Artificial Intelligence*.

Fine-Morris, M., and Muñoz-Avila, H. 2019. Learning domain structure in HGNs for nondeterministic planning. In *Proceedings of the 2nd ICAPS Workshop on Hierarchical Planning (HPlan 2019)*, 22–30.

Fine-Morris, M.; Floyd, M. W.; Auslander, B.; Pennisi, G.; Gupta, K.; Roberts, M.; Heflin, J.; and Muñoz-Avila, H. 2022. Learning decomposition methods with numeric landmarks and numeric preconditions. In *Proceedings of the 5th ICAPS Workshop on Hierarchical Planning (HPlan 2022)*, 29–37.

Geib, C. W., and Goldman, R. P. 2011. Recognizing plans with loops represented in a lexicalized grammar. In Burgard, W., and Roth, D., eds., *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. AAAI Press.

Grünwald, P. 1996. A minimum description length approach to grammar inference. In Wermter, S.; Riloff, E.; and Scheler, G., eds., *Connectionist, Statistical and Symbolic Approaches to Learning for Natural Language Processing*, 203–216. Berlin, Heidelberg: Springer Berlin Heidelberg.

Hérail, P., and Bit-Monnot, A. 2022. Learning Operational Models from Demonstrations: Parameterization and Model Quality Evaluation. In *ICAPS Hierarchical Planning Workshop (HPlan)*.

Hogg, C.; Muñoz-Avila, H.; and Kuter, U. 2008. HTN-MAKER: Learning HTNs with minimal additional knowledge engineering required. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 2*, AAAI'08, 950–956. Chicago, Illinois: AAAI Press.

Höller, D.; Wichlacz, J.; Bercher, P.; and Behnke, G. 2021. Compiling HTN plan verification problems into HTN planning problems. In *Proceedings of the 4th ICAPS Workshop on Hierarchical Planning (HPlan 2021)*, 8–15.

Kantharaju, P.; Ontañón, S.; and Geib, C. W. 2019. Extracting CCGs for plan recognition in RTS games. In Guzdial, M.; Osborn, J. C.; and Snodgrass, S., eds., *Proceedings of the 2nd Workshop on Knowledge Extraction from Games Co-Located with 33rd AAAI Conference on Artificial Intelligence, KEG@AAAI 2019, Honolulu, Hawaii, January 27th, 2019*, volume 2313 of *CEUR Workshop Proceedings*, 9–16. CEUR-WS.org.

Lam, H. T.; Mörchen, F.; Fradkin, D.; and Calders, T. 2014. Mining Compressing Sequential Patterns. *Statistical Analysis and Data Mining: The ASA Data Science Journal* 7(1):34–52.

Li, N.; Cushing, W.; Kambhampati, S.; and Yoon, S. 2014. Learning Probabilistic Hierarchical Task Networks as Probabilistic Context-Free Grammars to Capture User Preferences. *ACM Transactions on Intelligent Systems and Technology* 5(2):32.

Nieuwenhuis, R., and Oliveras, A. 2006. On SAT Modulo Theories and Optimization Problems. In Biere, A., and Gomes, C. P., eds., *Theory and Applications of Satisfiability Testing - SAT 2006*, Lecture Notes in Computer Science, 156–169. Berlin, Heidelberg: Springer.

Schreiber, D. 2021. Lilotane: A Lifted SAT-based Approach to Hierarchical Planning. *Journal of Artificial Intelligence Research* 70:1117–1181.

Zhuo, H. H.; Muñoz-Avila, H.; and Yang, Q. 2014. Learning hierarchical task network domains from partially observed plan traces. *Artificial Intelligence* 212:134–157.

Zhuo, H. H.; Peng, J.; and Kambhampati, S. 2019. Learning Action Models from Disordered and Noisy Plan Traces. *arXiv:1908.09800 [cs]*.