Automatically Learning HTN Methods from Landmarks

Ruoxi Li[1], Dana Nau[1], Mark Roberts[2], Morgan Fine-Morris[2]

[1]Dept. of Computer Science and Institute for Systems Research, Univ. of Maryland, College Park, MD, USA
[2]Navy Center for Applied Research in AI, Naval Research Laboratory, Washington, DC, USA
rli12314@cs.umd.edu, nau@umd.edu, mark.roberts@nrl.navy.mil, morgan.fine-morris.ctr@nrl.navy.mil

Abstract

Hierarchical Task Network (HTN) planning usually requires a domain engineer to provide manual input about how to decompose a planning problem. Even HTN-MAKER, a well-known method-learning algorithm, requires a domain engineer to annotate the tasks with information about what to learn. We introduce CURRICULAMA, an HTN method learning algorithm that completely automates the learning process. It uses landmark analysis to compose annotated tasks and leverages curriculum learning to order the learning of methods from simpler to more complex. This eliminates the need for manual input, resolving a core issue with HTN-MAKER. We prove CURRICULAMA's soundness, and show experimentally that it has a substantially similar convergence rate in learning a complete set of methods to HTN-MAKER.

1 Introduction

Automated planning systems require a domain expert to provide knowledge about the dynamics of the planning domain. In Hierarchical Task Networks (HTNs), expert-provided knowledge includes structural properties and potential hierarchical problem-solving strategies in the form of HTN decomposition methods. Writing these methods is a significant knowledge engineering burden. Some techniques (Hogg, Muñoz-Avila, and Kuter 2016; Hogg, Muñoz Avila, and Kuter 2008) partially overcome this burden by learning HTN methods after analyzing the semantics of a solution plan for planning problems. But these techniques still require input from the human designer. We overcome the need for human input by combining two insights to produce an algorithm, CURRICULAMA, that leverages planning landmarks and curriculum learning. Unlike other HTN learning algorithms, CURRICULAMA doesn't require a human to construct a curriculum because it constructs its own curriculum by analyzing the landmarks in planning problems.

Curriculum learning (Bengio et al. 2009) is a training strategy that improves learning performance by presenting training examples in increasing order of difficulty. We apply curriculum learning to the problem of learning HTN methods by emphasizing learning simpler methods before learning gradually more complex methods that incorporate previously learned methods.

Landmarks (Hoffmann, Porteous, and Sebastia 2004) are facts that must appear in every solution to a planning problem. In the context of learning hierarchical knowledge, methods that achieve landmarks provide a backbone for solving a planning problem. More critically, landmarks also provide a natural way to structure methods automatically.

We develop an approach that builds curricula to learn methods that achieve landmarks. This paper makes the following contributions:

- We introduce CURRICULAMA, which uses landmarks to generate curricula for constructing HTN methods. This approach obviates HTN-MAKER's need for manual annotation of tasks.

- We prove that the methods learned by CURRICULAMA can be used by a hierarchical planner to solve an HTN planning problem that is equivalent to the classical planning problem from which the methods were learned.

- Our experimental results show that CURRICULAMA has a similar convergence rate to HTN-MAKER in learning a complete set of methods to solve all the test problems.

2 Background

HTN-MAKER. Our work builds on the HTN method learning mechanism of the HTN-MAKER algorithm, which learns hierarchical planning knowledge in the form of decomposition methods for HTNs. HTN-MAKER takes as input the initial states from a set of classical planning problems in a planning domain and solutions to those problems, as well as a set of semantically-annotated tasks to be accomplished. The algorithm analyzes this semantic information in order to determine which portions of the input plans accomplish a particular annotated task and constructs HTN methods based on those analyses. Formally, each annotated task has a head, preconditions, and goals. For example, here is the Make-Clear annotated task for the Blocks World domain:

```
1      (:task
2      :head Make-Clear(?a - block)
3      :preconditions ()
4      :goals ((clear ?a)))
```

What we call goals HTN-MAKER calls effects. It does not cause the goals to be true; instead, the goals specify what is needed to be true after performing the annotated task.

Landmarks. *Landmarks* are a natural way to subdivide a planning solution. A landmark (Hoffmann, Porteous, and Sebastia 2004) for a planning problem is a fact that is true at some point in every plan that solves the problem. A landmark graph is a directed graph where the nodes are landmarks and the edges are orderings. There are four types of orderings among landmarks: natural, necessary, greedy-necessary, and reasonable (see Appendix for formal definitions [1]).

3 Learning HTN methods with Curricula Generated from Landmarks

Suppose we want to teach an HTN learner to learn methods for solving some task τ. An ideal curriculum would focus the learner on the simplest subtasks of τ first, then build more and more complex subtasks until all of τ is learned. Learning HTN methods from plan traces is a common approach, and it follows that traces for subtasks of τ will also be subtraces of a plan trace for τ. Specifically, if π is a plan trace for solving τ, then the plan trace for solving each subtask τ is a subtrace $\pi[b,e]$ of π, where $[b,e]$ indicates the beginning and ending indices of the subtrace. Thus we can represent a k-step curriculum that learns subtraces as a sequence of triples of the form (b_i, e_i, τ_i) where $i \in \mathbb{N}_k$.

Definition 1. A *classical planning problem* P is a triple (Σ, s_0, g), where Σ is the classical planning domain description, s_0 is the initial state and g is the goal (Ghallab, Nau, and Traverso 2016).

Definition 2. Given a plan trace π, a *curriculum* C is a sequence of k curriculum steps of the form (b_i, e_i, τ_i), where b_i and e_i specify the starting and ending indices of the subtrace to analyze, and τ_i specifies the annotated task to learn from the subtrace for step $i \in \mathbb{N}_k$.

For example, consider a Blocks World classical planning problem with 4 blocks A, B C and D stacked on each other (see Figure 1) and the goal is to have block A's top clear. Formally: initial state s_0={(on-table A), (on B A), (on C B), (on D C), (clear D), (hand-empty)}; and goal g={(clear A)}. A possible solution π_{clearA} is to remove the blocks above A one by one through 5 actions, which are prefixed with the character '!' to distinguish them from predicates. Here is a 3-step curriculum for learning methods of π_{clearA}:

Step	Begin	End	Annotated Task
a	1	1	Make-Clear
b	1	3	Make-Clear
c	1	5	Make-Clear

We want step a to learn a method *m1* for annotated tasks Make-Clear from the first action in plan P, this method tries to clear a block under one block. Then we want step b to learn a method *m2* for annotated tasks Make-Clear
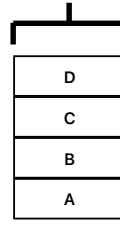
Action 1: (!Unstack D C)
Action 2: (!Putdown D)
Action 3: (!Unstack C B)
Action 4: (!Putdown C)
Action 5: (!Unstack B A)

Figure 1: A Blocks World problem in which the initial state is a stack of 4 blocks. The goal is to make the bottom block A clear. The plan to achieve the goal is shown on the right.

Algorithm 1 CURRICUGEN: Curriculum Generation from Landmarks

Input: a classical planning problem P, a possibly empty set of HTN methods M
Output: an updated set of HTN methods M

1: $(\Sigma, s_0, g) \leftarrow P$
2: $(V, E_V) \leftarrow$ extract landmark graph from (Σ, s_0, g)
3: add reasonable orders to (V, E_V)
4: $C \leftarrow \langle \rangle$ {initialize the curriculum steps}
5: $\pi \leftarrow \langle \rangle$ {initialize the plan trace}
6: $s \leftarrow s_0$ {initialize the current state}
7: $i \leftarrow 0$ {initialize the plan length}
8: **while** $V \neq \emptyset$ **do**
9:     select and remove a vertex v in (V, E_V) that has no predecessors
10:     $\pi' \leftarrow$ CLASSICALPLANNER(Σ, s, v)
11:     $s \leftarrow \gamma(s, \pi)$
12:     $i \leftarrow i + length(\pi')$
13:     concatenate π' to π
14:     $t \leftarrow$ MAKEANNOTATEDTASK(v)
15:     **for** k from i to 1 **do**
16:         append (k, i, t) to C
17: $M =$ CURRICULEARN(Σ, s_0, π, C, M)
18: **return** M

---

from the first to the third action in plan P, this method tries to clear a block under two blocks, and would presumably contain a subtask Make-Clear that is related to *m1* previously learned from the first action. Then step c learns a new method *m3* for Make-Clear that subsumes *m2*.

CURRICULAMA, has two subroutines: CURRICUGEN generates curricula from landmarks while CURRICULEARN learns HTN methods from the curricula.

3.1 CURRICUGEN

Since a landmark must be true at some point in every solution to a planning problem, we hypothesized that it would be useful to learn methods that reach landmarks. Our algorithm, CURRICUGEN (Algorithm 1) extracts landmarks from a planning problem, then generates a curriculum from those landmarks.

CURRICUGEN takes as input a classical planning problem. It first generates a landmark graph for P using h^m Landmarks (Line 2) from the landmark generation method introduced by Keyder, Richter, and Helmert (2010); then it
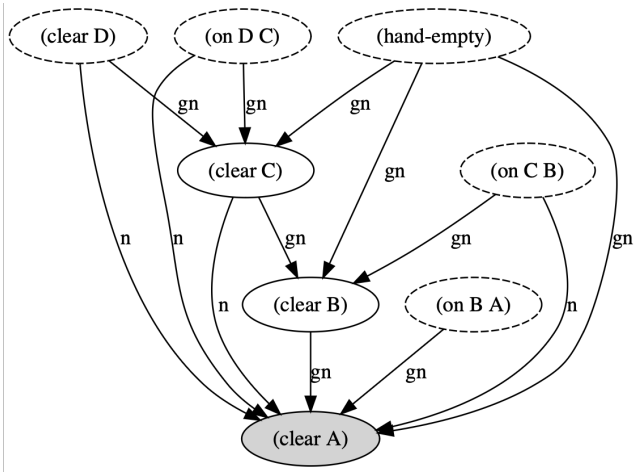
Figure 2: A landmark graph for clearing block A from blocks B, C and D above in the Blocks World domain. The circled nodes are landmarks, where the dashed nodes are the landmarks that are satisfied in the initial state, and the filled node is the goal. The edges are orderings among the landmarks, where 'gn' stands for greedy necessary ordering, and 'n' stands for natural ordering.
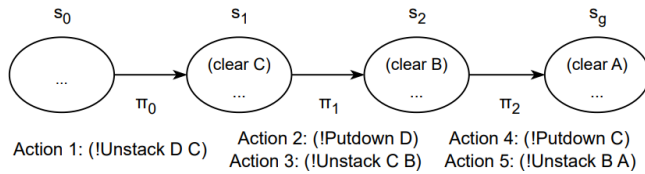

Figure 3: The subplans generated from the landmarks.

adds reasonable orders to the landmark graph (Line 3) from the method described by Hoffmann, Porteous, and Sebastia (2004);[2] and then it iterates through the landmarks by their orderings. For each landmark, CURRICUGEN iteratively obtains a solution trace from the current state using a classical planner and updates the current state by applying the solution plan (Lines 10 and 11). MAKEANNOTATED-TASK (Line 14) takes as input the current landmark and produces an annotated task that has a task name, empty preconditions, and the landmark as its goals. Given the annotated task produced from the landmark, CURRICUGEN generates curriculum steps (Lines 15 and 16) that progressively trace backward to the beginning of the plan to learn methods.

Example. In the Blocks World problem in Figure 1, the initial state has 4 stacked blocks A, B, C, and D, and the goal is to clear block A. Excluding the initial state, Figure 2 shows the landmark graph from CURRICUGEN for (s_0, g), which consists of 3 landmarks: (clear C) $\prec$ (clear B) $\prec$ (clear A).

CURRICUGEN generates subplans to achieve the first,

second and the third landmarks (Figure 3). For each landmark, it creates an annotated task (in this case, Make-Clear) and curriculum steps in which the final indices correspond to the action that achieves the landmark, and the beginning indices go back to the plan trace's start:

Step	Begin	End	Annotated Task
a	1	1	Make-Clear
b	3	3	Make-Clear
c	2	3	Make-Clear
d	1	3	Make-Clear
e	5	5	Make-Clear
f	4	5	Make-Clear
g	3	5	Make-Clear
h	2	5	Make-Clear
i	1	5	Make-Clear

The curriculum comprises nine steps, labeled from a to i. Each step is defined by a specific segment of π, delineated by its beginning and ending indices, along with the name of an annotated task. The curriculum is structured to initiate simpler tasks, gradually progressing to more complex ones. The regressive sequencing of the beginning indices aims at learning methods with varying preconditions for the same annotated tasks.

3.2 CURRICULEARN

CURRICULAMA's CURRICULEARN subroutine learns HTN methods from curricula, shown in Algorithm 2. The input to CURRICULEARN includes a domain description Σ, an initial states s_0, an execution trace π (which can be a plan produced by a planner given a goal), a curriculum C, and a possibly empty set of HTN methods M. For each curriculum step in C, it uses the algorithm LEARN-METHOD[3] to perform the analysis on the subtrace $\pi[b, e]$ and learns some new methods for τ. It also keeps a set of indexed method instances X to identify and reuse previously learned methods as subroutines in a new method that is being synthesized. Specifically, each method is indexed by the beginning index of the corresponding subtrace b, and the ending index of the corresponding subtrace e.

Algorithm 2 A high-level description of CURRICULEARN.

Input: classical domain description Σ, initial state s_0, solution trace π, curriculum C, and a possibly empty set of HTN methods M
Output: an updated set of HTN methods
1: initialize $X \leftarrow \emptyset$
2: let $\vec{S}$ be the state trajectory generated from $\gamma(s_0, \pi)$
3: **for** $(b, e, \tau) \in C$ **do**
4:    $M' \leftarrow$ LEARN-METHOD$(\pi, \vec{S}, \tau, X, b, e)$
5:    $M \leftarrow M \cup M'$
6:    **for** $m \in M'$ **do**
7:       $X \leftarrow X \cup \{(m, b, e)\}$
8: **return** M

---

[2]We use the implementation of h^m landmark generation and reasonable order extraction in the Fast Downward planning system (https://www.fast-downward.org/), configured to only allow for single-atom (conjunctive) landmarks.

[3]LEARN-METHOD performs hierarchical goal regression over a plan trace. It is the same procedure that HTN-MAKER uses as a subroutine (Hogg, Muñoz-Avila, and Kuter 2016, Algorithm 3) to learn preconditions and subtasks of HTN methods.

CURRICULAMA takes as input a planning problem and outputs a set of learned HTN methods. It does this by using CURRICUGEN to generate curricula from landmarks, and CURRICULEARN to acquire HTN methods from these curricula. This obviates HTN-MAKER's need for manual annotation of tasks and corresponding plan subtraces.

4 Theoretical Analysis

We prove that the methods learned by CURRICULAMA from a classical planning problem can be applied to solve the equivalent hierarchical problem. First, we need to go through the process of CURRICUGEN given a classical planning problem and define the hierarchical planning problem.

Given a classical planning problem $P = (\Sigma, s_0, g)$ as a training example, CURRICUGEN produces a solution trace π and a curriculum C. Given π, C, and a possibly empty set of HTN methods M, CURRICULEARN will add newly learned methods to M. Let τ be an annotated task that has g as its goals. Then $P_h = ((\Sigma, M), s_0, \langle \tau \rangle)$ is the *hierarchical planning problem* (see Definition 3) equivalent to the classical planning problem P.

Definition 3. A *hierarchical planning problem P_h* is a triple $(\Sigma_h, s_0, \langle \tau \rangle)$ where Σ_h is the hierarchical planning domain description, s_0 is the initial state and $\langle \tau \rangle$ is the task list. A hierarchical planning domain description Σ_h is a tuple (Σ, M), where Σ is the classical planning domain description and M is the set of HTN methods.

Now we can show how the methods learned by CURRICULAMA can be used to solve the hierarchical problem, which is equivalent to the classical problem from which the methods were learned.

Proposition 1. *Given $P = (\Sigma, s_0, g)$, π, τ, M, and $P_h = ((\Sigma, M), s_0, \langle \tau \rangle)$, π is a solution to P_h as a result of hierarchically decomposing g using the methods in M.*

Proof Sketch If π is empty or g is satisfied in s_0, then CURRICULEARN will learn a trivial method for τ that has empty subtasks, which is sufficient to solve the problem. Otherwise, since g has to be the final landmark in the landmark graph of P, the final curriculum step in C is $(1, len(\pi), \tau)$. Therefore, CURRICULEARN will learn at least one method from curriculum step $(1, len(\pi), \tau)$. This method must be applicable to s_0 in P_h because its preconditions were computed by regressing g through the actions of π (Hogg, Muñoz-Avila, and Kuter 2016, the LEARN-METHOD precedure), which is applicable to s_0. Furthermore, the goal regression procedure guarantees that whenever the preconditions of a method are satisfied there must be some way to reduce the subtasks of that method using other methods learned from π, because the subtasks of that method were chosen from the indexed instances of other methods. Eventually, g in P_h will be hierarchically decomposed into the action sequence π. $\square$

Therefore, CURRICULAMA is sound for the original problem for which it learned methods. That is, methods learned by CURRICULAMA from a classical planning problem P will solve the equivalent hierarchical problem P_h.

However, we also want to know how rapidly it can learn a complete set of methods from the training problems. In the next section, we will empirically evaluate CURRICULAMA to show that it has a comparable convergence rate in learning a complete set of methods to HTN-MAKER.

5 Empirical Study

We have evaluated CURRICULAMA (and compared it to HTN-MAKER) experimentally in five IPC (International Planning Competition) domains: Logistics, Blocks World, Rover, Satellite, and Zeno Travel. These domains are used for evaluation in the original papers on HTN-Maker. Due to space limitations, we present results for the Blocks World and Logistics; results are similar across domains and can be seen in the Appendix. We assess the efficiency of CURRICULAMA in learning Hierarchical Task Network methods and the effectiveness in solving hierarchical planning problems using the learned methods.

Our evaluation considered how well the methods learned from an incremental set of training problems can solve a set of static testing problems. A single trial for a domain used PDDL-Generators (Seipp, Torralba, and Hoffmann 2022) to generate 150 random training problems and 50 testing problems from the same distribution of parameters. Starting with an empty set of methods M, the procedure iterated through the training problems $(1, 2, .., 150)$, augmented M using either CURRICULAMA or HTN-MAKER, and used HTN-MAKER's version of the SHOP planner (Nau et al. 1999) to solve the 50 test problems with the current set M. We repeated five trials in each of the five domains and reported on the following metrics: (1) convergence, (2) average plan length, (3) average planning time, (4) method generation, (5) running time in learning. All experiments are run on AMD EPYC 7763 (2.45GHz).

Figure 4 shows that CURRICULAMA's method learning exhibits a similar convergence rate and results in plan lengths and planning time comparable to HTN-MAKER, all while achieving a significant advantage: it *completely eliminates* the need for expert-provided annotated tasks.

CURRICULAMA's planning mechanism may cause it to learn extraneous methods in some domains (e.g., the Logistics, Satellite and Rover domain). While it's possible that this may be an indication of overfitting, we believe this is more likely a result of partial orders in the landmark graph. The landmark generation algorithms used by CURRICULAMA (Algorithm 1, lines 2 and 3) return only a partial ordering among landmarks given the additional reasonable orders. All reasonable orderings are not determined because determining whether a reasonable order exists between two given landmarks is a PSPACE-complete problem (Hoffmann, Porteous, and Sebastia 2004). Then, CURRICULAMA enforces a total ordering to formulate a sequence of subgoals, which is not necessarily the optimal strategy (see Appendix for an example). This often results in CURRICULAMA's derivation of additional methods from extended plan traces, as those methods cover more potential (and suboptimal) paths to the goal. Improving CURRICULAMA's strategy for ordering landmarks by incorporating more so-
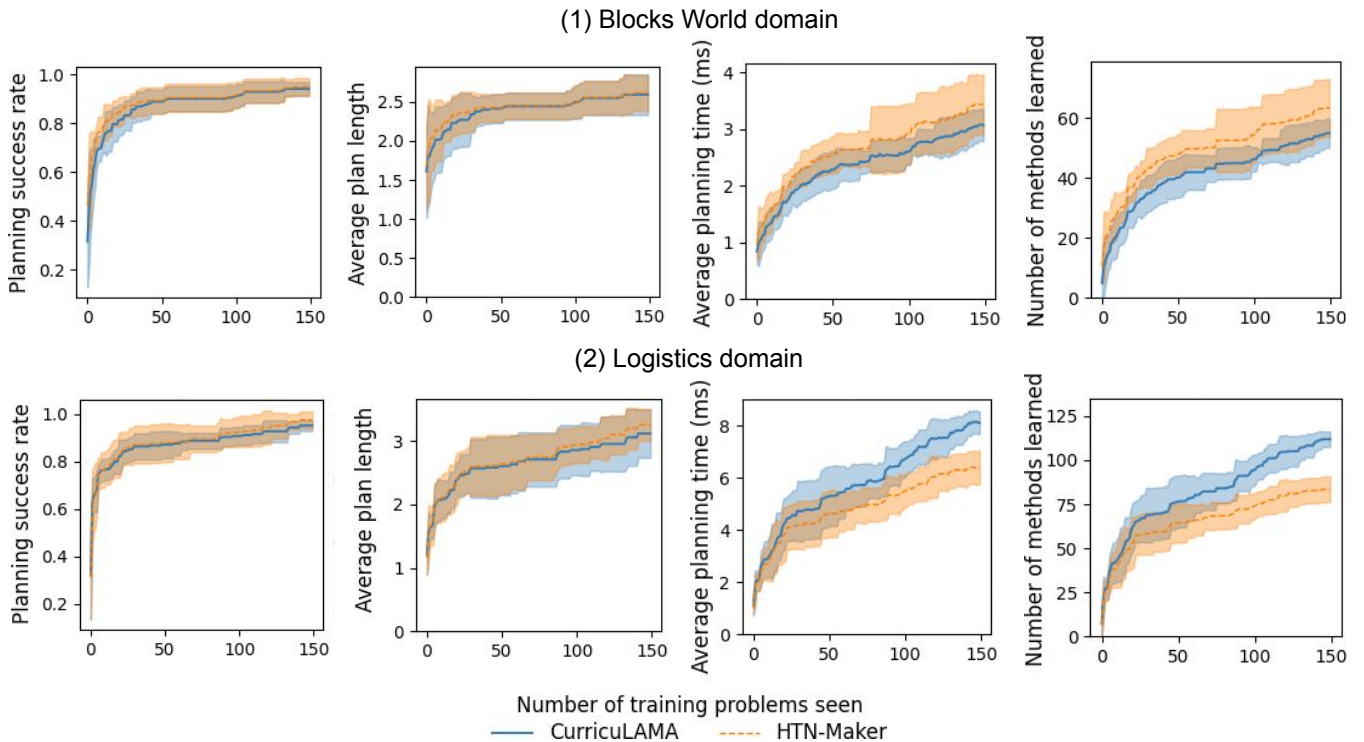
Figure 4: *Experimental results* in (1) the Blocks World domain and (2) the Logistics domain. From left to right, the subfigure's y-axis shows (a) the fraction of problems that the planner could successfully solve using the methods that each learning algorithm learned; (b) the average length of the plans that the planner produced using the learned methods; (c) the average planning time over the 50 test problems; and (d) the total number of methods learned. In each of the subfigures, the x-axis shows the number of training problems (0-150) from which the methods were learned. The blue line displays the results for CURRICULAMA and the orange dashed line displays the results for HTN-MAKER. The shaded areas indicate the variance in the number of methods learned across five trials.

phisticated heuristics could help reduce the creation of these redundant methods and is a topic for future work.

While CURRICULAMA may learn slightly more methods in some domains due to suboptimal landmark orderings, this does not appear to have a detrimental impact on planning success rates or plan lengths. Notably, in domains where CURRICULAMA successfully captures all required landmark orderings, it learns fewer methods than HTN-MAKER, which results in relatively shorter planning time.

It's also worth highlighting that CURRICULAMA's additional computational time (Figure 5), averaging between 0.2 to 0.8 seconds per problem, is negligible when compared to the overall planning time or the time required from a domain expert. Thus, CURRICULAMA is an efficient and promising alternative to acquiring planning methods.

5.1 Future Work

The number of methods and planning time keep increasing without convergence for both algorithms in the Blocks World, Logistics and Rover domain. We believe that given enough training problems, they will eventually converge. To verify that, we will expand the training set in our future experiments.

We are also interested in an empirical study that compares manually annotated task and automatically annotated tasks when directly applied to HTN-MAKER without any curricula. This will give an idea of the quality of the tasks generated by CURRICUGEN.

Last but not least, we will theoretically and empirically analyse CURRICULAMA's time complexity as some measure of task domain problem or solution complexity increases.

6 Related work

Several researchers have investigated ways to learn structural knowledge, including HTNs, though they did not use curricula generated from landmarks to guide learning.

The algorithm by Lotinac and Jonsson (2016) uses invariant analysis to construct an HTN from the PDDL description of a planning domain and a single representative instance. Learn-HTN (Zhuo et al. 2009) learns HTN-method preconditions and action preconditions and effects under partial observability. It receives task decomposition trees whose leaves are all primitive actions. It solves constraints using MAX-SAT among subtask-pairs in a task decomposition tree to learn action models and method preconditions. Similar to Learn-HTN, HTNLearn (Zhuo, Munoz-Avila, and Yang 2014) learns HTN methods and action mod-
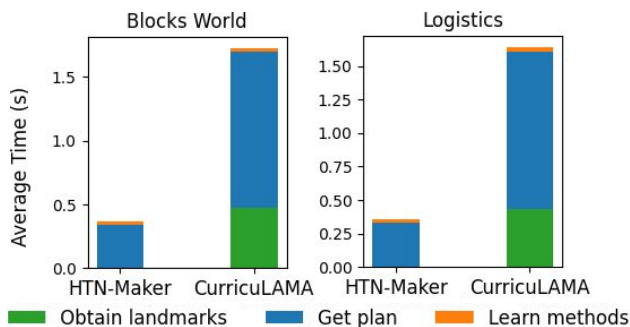
Figure 5: *Running time needed to learn methods.* The bars represent the average time that each learning algorithm spent on different parts of the learning process. Green represents the time to obtain landmarks (Alg 1, Line 2 and 3), blue indicates the time to obtain the plan (Alg 1, Line 8 to 16) , and orange shows the time to learn methods.

els from partially observed plan traces annotated with potentially empty partial decomposition trees that capture task-subtask decompositions.

The HTN Learning System (Yang, Pan, and Pan 2007) uses partitioning to learn HTN methods. Other algorithms learn method preconditions using the hierarchical relationships between tasks, the action models, a complete description of the intermediate states using case-based reasoning (Xu and Muñoz-Avila 2005) and version-space learning (Ilghami et al. 2005). The algorithm by Li, Kambhampati, and Yoon (2009) uses techniques similar to probabilistic context-free grammar learning and learns probabilistic HTNs. It can learn recursive methods on repeated structures.

Word2HTN (Gopalakrishnan, Munoz-Avila, and Kuter 2016) combines semantic text analysis techniques and subgoal learning to create HTNs. Plan traces are viewed as sentences where a plan trace consists of actions with their grounded preconditions and effects. Each word in the sentence is an atom or an action in the plan trace. This work uses Word2Vec to convert each word into a vector representation and applies agglomerative hierarchical partitioning on the learned vectors to learn methods with binary-subtask decompositions. As an extension to their approach, (Fine-Morris et al. 2020) approximates landmarks using solution traces and learns methods with symbolic and numeric preconditions that initially decompose problems using two or more landmarks, and then finish the decomposing using (primitive task, complex task) right-recursion.

CC-HTN (Hayes and Scassellati 2016) and Circuit-HTN (Chen et al. 2021) translate execution traces into multi-trace graphical representations where primitive tasks comprise vertices and edges indicate sequential tasks. They apply bottom-up consolidation techniques to group simple tasks into progressively larger ones. Circuit-HTN treats the graph as a circuit of resistors, using techniques from the reduction of parallel and in-series resistors to discover new decompositions. CC-HTN similarly uses clique- and chain-detection for structure-learning and also learns the method parameters.

Hérail and Bit-Monnot (2023) iteratively learns HTNs

using bottom-up pattern-recognition and compression techniques on common sequences of subtasks in traces, which get replaced with a task symbol. Segura-Muros, Pérez, and Fernández-Olivares (2017) learn HTNs using process and data mining by converting execution traces into event-logs and extracting the preconditions and effects for each task using a fuzzy rule-based learning algorithm.

HPNL (Langley 2022) is a system that learns new methods for Hierarchical Problem Networks (Shrobe 2021) by analyzing sample hierarchical plans, using violated constraints to identify state conditions, and ordering conflicts to determine goal conditions.

The hierarchical plan recognition algorithm by Geib (2009) uses Combinatory Categorial Grammars (CCGs) as part of the ELEXIR framework. It requires a hand-authored CCG representing structure of plans done by agents. Lex_{Learn} (Geib and Kantharaju 2018) learns CCGs by enumerating CCG categories for a set of plan traces from templates. Lex_{Greedy} (Kantharaju 2021) employs a greedy approach to improve the scalability of CCG learning. Lex_{Greedy}^{T} (Kantharaju 2021) learns CCGs in domains with type trees as an extension to Lex_{Greedy}.

Teleoreactive Logic Programs (TLPs) are a framework for encoding knowledge using ideas from logic programming, reactive control, and HTNs. This work includes ways to learn recursive TLPs from problem solution traces (Choi and Langley 2005), a learning method that acquires recursive forms of TLP structures from traces of successful problem solving (Langley et al. 2006), and an incremental learning algorithm for TLPs based on explanation-based learning (Nejati, Langley, and Konik 2006).

7 Conclusion

CURRICULAMA generates curricula from landmarks and uses them to acquire HTN methods according to these curricula. We have proved that the methods that CURRICULAMA learns from classical planning problems enable an HTN planner to solve equivalent HTN planning problems. In our experiments CURRICULAMA learned comparably good methods to those learned by HTN-MAKER for the same problems, with no requirement for a human to provide methods, curricula, or annotations of tasks. The idea that landmarks are useful for structural knowledge learning, and that curricula can be constructed from those landmarks, may apply to other structural knowledge learning techniques.

Acknowledgements

References

Bengio, Y.; Louradour, J.; Collobert, R.; and Weston, J. 2009. Curriculum learning. In *ICML*, 41–48.

Chen, K.; Srikanth, N. S.; Kent, D.; Ravichandar, H.; and Chernova, S. 2021. Learning Hierarchical Task Networks with Preferences from Unannotated Demonstrations. In *CoRL*, 1572–1581.

Choi, D., and Langley, P. 2005. Learning teleoreactive logic programs from problem solving. In *ILP*, 51–68.

Fine-Morris, M.; Auslander, B.; Floyd, M. W.; Pennisi, G.; Muñoz-Avila, H.; and Gupta, S. K. M. 2020. Learning hierarchical task networks with landmarks and numeric fluents by combining symbolic and numeric regression. In *ACS*.

Geib, C., and Kantharaju, P. 2018. Learning combinatory categorial grammars for plan recognition. In *AAAI*, 3007–3014.

Geib, C. W. 2009. Delaying commitment in plan recognition using combinatory categorial grammars. In *IJCAI*, 1702–1707.

Ghallab, M.; Nau, D.; and Traverso, P. 2016. *Automated planning and acting*. Cambridge University Press.

Gopalakrishnan, S.; Munoz-Avila, H.; and Kuter, U. 2016. Word2HTN: Learning task hierarchies using statistical semantics and goal reasoning. In *IJCAI Workshop on Goal Reasoning*.

Hayes, B., and Scassellati, B. 2016. Autonomously constructing hierarchical task networks for planning and human-robot collaboration. In *ICRA*, 5469–5476.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *JAIR* 22:215–278.

Hogg, C.; Muñoz Avila, H.; and Kuter, U. 2008. HTN-MAKER: Learning HTNs with minimal additional knowledge engineering required. In *AAAI*, 950–956.

Hogg, C.; Muñoz-Avila, H.; and Kuter, U. 2016. Learning hierarchical task models from input traces. *Computational Intelligence* 32(1):3–48.

Hérail, P., and Bit-Monnot, A. 2023. Leveraging Demonstrations for Learning the Structure and Parameters of Hierarchical Task Networks. In *FLAIRS*, volume 36.

Ilghami, O.; Munoz-Avila, H.; Nau, D. S.; and Aha, D. W. 2005. Learning approximate preconditions for methods in hierarchical plans. In *ICML*, 337–344.

Kantharaju, P. 2021. *Learning Decomposition Models for Hierarchical Planning and Plan Recognition*. Ph.D. Dissertation, Drexel University, Philadelphia, Pennsylvania, USA.

Keyder, E.; Richter, S.; and Helmert, M. 2010. Sound and complete landmarks for and/or graphs. In *ECAI*, 335–340.

Langley, P.; Choi, D.; Olsson, R.; and Schmid, U. 2006. Learning recursive control programs from problem solving. *Journal of Machine Learning Research* 7(3).

Langley, P. 2022. Learning hierarchical problem networks for knowledge-based planning. In *ILP*.

Li, N.; Kambhampati, S.; and Yoon, S. 2009. Learning probabilistic hierarchical task networks to capture user preferences. In *IJCAI*, 1754–1759.

Lotinac, D., and Jonsson, A. 2016. Constructing hierarchical task models using invariance analysis. In *ECAI*. 1274–1282.

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

Nejati, N.; Langley, P.; and Konik, T. 2006. Learning hierarchical task networks by observation. In *ICML*, 665–672.

Segura-Muros, J. A.; Pérez, R.; and Fernández-Olivares, J. 2017. Learning HTN domains using process mining and data mining techniques. In *ICAPS Workshop on Generalized Planning*.

Seipp, J.; Torralba, Á.; and Hoffmann, J. 2022. PDDL generators. https://doi.org/10.5281/zenodo.6382173.

Shrobe, H. E. 2021. Hierarchical problem networks for knowledge-based planning. In *ACS*.

Xu, K., and Muñoz-Avila, H. 2005. A domain-independent system for case-based task decomposition without domain theories. In *AAAI*, 234–240.

Yang, Q.; Pan, R.; and Pan, S. J. 2007. Learning recursive htn-method structures for planning. In *ICAPS Workshop on AI Planning and Learning*.

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 *IJCAI*, 1804–1809.

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