Teaching an HTN Learner

Ruoxi Li[1], **Mark Roberts**[2], **Morgan Fine-Morris**[2,3], **Dana Nau**[1]

[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
[3]Department of Computer Science, Lehigh University, Bethlehem, PA 18015, USA
rli12314@cs.umd.edu, mark.roberts@nrl.navy.mil, nau@umd.edu, mof217@lehigh.edu

Abstract

We describe Teachable-HTN-Maker, a modified version of the well-known HTN-Maker algorithm that learns Hierarchical Task Network (HTN) methods. Instead of learning methods from all subsequences of a solution plan as HTN-Maker does, Teachable-HTN-Maker learns from a curriculum consisting of examples that are presented in a meaningful order. We compare Teachable-HTN-Maker against HTN-Maker in two planning domains, and observe that it learns fewer methods and better ones.

1 Introduction

Curriculum learning (Bengio et al. 2009) is a training strategy for machine learning. It was inspired by the observation that humans and animals learn much better when the examples are not randomly presented but organized in a meaningful order that starts by illustrating simple concepts and gradually introduces more complex ones. In this paper we investigate how to apply this strategy to improve HTN-Maker (Hogg, Muñoz-Avila, and Kuter 2016).

HTN-Maker learns methods for annotated tasks from subsequences of a solution plan. However, it tries to learn methods for *all* of the annotated tasks from *all* of the subsequences. Many of the methods learned by are not useful because they have undesirable preconditions or decomposition strategies. As a result, an HTN planner that uses methods learned by HTN-Maker may not perform efficiently.

In this paper we make the following contributions:

- We describe Teachable-HTN-Maker, a modified version of HTN-Maker. Instead of examining every subsequence of a solution plan, Teachable-HTN-Maker examines only the subsequences that we tell it to examine in an order that we specify. This modification makes it possible for Teachable-HTN-Maker to learn from curricula.
- We compare Teachable-HTN-Maker and HTN-Maker on two sets of planning problems. One is to move a stack of n blocks in the Blocks World, maintaining their order (which requires moving the stack twice). The other is to deliver n packages in the Logistics domain. In our experiments, Teachable-HTN-Maker learned fewer methods and better ones than HTN-Maker, and did so with less running time. A planner using the methods learned by Teachable-HTN-Maker solves more problems with lower runtime than with the methods learned by HTN-Maker.

2 Background and Related Work

Automated planning systems typically require that a domain expert provide knowledge about the dynamics of the planning domain. In classical planning, the domain knowledge includes semantic descriptions of actions. In Hierarchical Task Networks (HTNs), the domain knowledge includes structural properties and potential hierarchical problem-solving strategies. A significant knowledge engineering burden for a domain expert is required to write HTN decomposition methods. HTN-Maker (Hogg, Muñoz-Avila, and Kuter 2016) overcomes this burden, in part, by learning HTN methods after analyzing the semantics of a solution plan for planning problems.

Several other works have investigated ways to learn HTN methods (Lotinac and Jonsson 2016; Zhuo, Munoz-Avila, and Yang 2014; Xiao et al. 2020). Furthermore, Choi and Langley (2005) have investigated how to learn hierarchical logic programs that are analogous to HTN methods. However, none of those investigations used curricula.

Algorithm 1 describes the high-level operation of HTN-Maker. Its input includes the domain $\mathcal{D}$, initial states from a planning problem $\mathcal{P}$ in a planning domain, an execution trace $\mathcal{E}$ (which can be a plan produced by a planner), a set $\mathcal{T}$ of *annotated tasks* to be accomplished, and the Boolean choice *p* of whether pruning is enabled. Each task's annotations include preconditions that need to be true to accomplish the task, and effects that must be true after accomplishing the task (see Figures 2 and 3 for examples).

During the learning process, if *pruning* is enabled, newly learned methods from the following two categories will be pruned by (i.e., removed from the set of learned methods): 1) subsumed methods, where method m_1 subsumes method m_2 if there exists a substitution that may be applied to m_2 such that both have the same head and subtasks and the precondition of the m_2 implies the precondition of m_1; and 2) unneeded methods: if the preconditions of a subtask are fulfilled, the subtask could just be called directly.

The procedure *LearnMethods* performs the analysis for τ on the subtrace $\mathcal{E}[start, end]$. HTN-Maker analyzes all $O(k^2)$ subtraces for an $\mathcal{E}$ of length *k*, and often learns many methods with undesirable preconditions or decomposition strategies. To address these issues, we modify HTN-Makerf (distribution version *ch-htn-tools-1.1*) to use a curriculum to guide the learning process.

Algorithm 1: A high-level description of HTN-Maker.

Input: domain $\mathcal{D}$, problem $\mathcal{P}$, solution trace $\mathcal{E}$, Annotated tasks $\mathcal{T}$, Pruning enabled p
Output: A set of HTN methods $\mathcal{M}$

```
M = ∅
for end ← 1 to |E| do
    for start ← end down to 1 do
        for τ in T do
            LearnMethods(start, end, τ, D, P, E, M, p)
return M
```

Algorithm 2: Teachable-HTN-Maker.

Input: domain $\mathcal{D}$, problem $\mathcal{P}$, solution plan $\mathcal{E}$, curriculum $\mathcal{C}$
Output: A set of HTN methods $\mathcal{M}$

```
M = ∅
for (start, end, τ) in C do
    LearnMethods(start, end, τ, D, P, E, M, p)
return M
```

3 Teachable HTN-Maker

Suppose we want to teach an HTN method learner how to solve some task τ. A curriculum would start by teaching the learner how to solve very simple subtasks of τ, then increasingly complicated subtasks, until we teach it how to solve τ itself. If the learner learns from plan traces, then the plan traces for the subtasks of τ will be subtraces of the plan trace for τ. More specifically, if $\mathcal{E}$ is a plan trace for τ, then the plan trace for each subtask τ_i is a subtrace $\mathcal{E}[start_i, end_i]$ of $\mathcal{E}$. Thus we can represent our curriculum as a sequence of triples of the form $(start_i, end_i, \tau_i)$.

Teachable-HTN-Maker is a modified version of HTN-Maker that takes such triples as input, and analyzes only these triples rather than analyzing every subsequence of $\mathcal{E}$. The pseudocode is in Algorithm 2.

4 Experimental Setup

To examine whether a curriculum can improve upon the methods learned by HTN-Maker, we compared Teachable-HTN-Maker and HTN-Maker in the Blocks World domain and the Logistics domain from the 2nd International Planning Competition (IPC-2). Although these domains are conceptually simple, large problems remain a challenge for planners. For our comparisons, we measured the total number of methods each system learned (with or without pruning) and the time they took to learn those methods, and we evaluated the methods' planning performance.

Blocks World Domain The first domain includes a number of blocks sitting on a table (possibly on top of each other), and a robotic hand that can grasp one block at a time. The objective is to learn methods to move a stack of *n* blocks using the robotic hand, keeping the top-to-bottom order of the blocks the same as in the original stack.

For example, let τ_1 be the task of moving a stack of two blocks (A and B) from block C onto the table, while maintaining their order (i.e. A on B). Let π_1 be the following plan for that task:

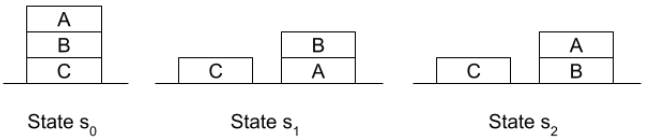

Figure 1: To move two blocks A and B from block C to the table while maintaining their order, the plan π_1 inverts their order (state s_1), then inverts it again (state s_2).

```
(:task Make-1Pile
 :parameters
   (?a)
 :precondition
   (and)
 :postcondition
   (and
     (on-table ?a)
     (clear ?a)))
```

```
(:task Make-2Pile
 :parameters
   (?a ?b)
 :precondition
   (and)
 :postcondition
  (and (on-table ?b)
       (on ?a ?b)
       (clear ?a)))
```

Figure 2: Example annotated tasks in the Blocks World.

Action 1: unstack(A,B)	Action 5: unstack(B,A)
Action 2: putdown(A)	Action 6: putdown(B)
Action 3: unstack(B,C)	Action 7: pickup(A)
Action 4: stack(B,A)	Action 8: stack(A,B)

Figure 1 shows what the plan does.

To teach how to accomplish τ_1, we can use the curriculum shown below. It consists of seven subplans of π_1, starting with simpler ones and combining them into progressively harder ones. For each subplan, the curriculum includes the annotated task (see Figure 2 for examples) that the subplan accomplishes.

	Subplan of π_1	Annotated Task
1.	Actions 1 and 2	Make-1Pile
2.	Actions 3 and 4	Make-2Pile
3.	Actions 1 through 4	Make-2Pile
4.	Actions 5 and 6	Make-1Pile
5.	Actions 7 and 8	Make-2Pile
6.	Actions 5 through 8	Make-2Pile
7.	Actions 1 through 8	Make-2Pile

The preconditions of the methods learned from this curriculum include cases where part of the stack has already been moved, but not cases where a block is held in the robot hand.

Logistics Domain The objective in the Logistics domain is to move packages among locations in various cities, using trucks within cities and airplanes between cities. Let τ_2 be the task of moving three packages p1, p2, p3 within a city, from locations p1s, p2s, p3s to p1d, p2d, p3d, respectively. Let π_2 be the following plan for τ_2:

Action 1: drive-to(p1s)	Action 7: drive-to(p2d)
Action 2: load(p1)	Action 8: unload(p2)
Action 3: drive-to(p1d)	Action 9: drive-to(p3s)
Action 4: unload(p1)	Action 10: load(p3)
Action 5: drive-to(p2s)	Action 11: drive-to(p3d)
Action 6: load(p2)	Action 12: unload(p3)

Here is a curriculum to teach how to perform τ_2. As before, each curriculum entry includes a subplan of π_2 and an annotated task (see Figure 3) that the subplan accomplishes:

	Subplan of π_2	Annotated Task
1.	Actions 1 through 4	Deliver-1Pkg
2.	Actions 5 through 8	Deliver-1Pkg
3.	Actions 1 through 8	Deliver-2Pkg
4.	Actions 9 through 12	Deliver-1Pkg
5.	Actions 1 through 12	Deliver-3Pkg

Methods First, we randomly generate problems with corresponding solution traces for moving a stack of *n* blocks in the Blocks World domain and delivering *n* packages in the Logistics domain. Then we compare the average number of methods learned (with and without pruning) as well as the time (ms) taken by HTN-Maker and our Teachable-HTN-Maker. After we learn HTN methods for each problem, we evaluate the methods by using them to solve the planning problems using an HTN planner: HTN-Maker's implementation of the SHOP (Nau et al. 1999) planning algorithm. Finally, we compare the average length of the plan generated by the HTN planner as well as the running time (ms) for those problems. For each stage of experiments (method learning or planning with the learned methods), for each problem domain (Blocks World or Logistics), as well as for each configuration of learning approaches (HTN-Maker or Teachable-HTN-Maker) and pruning strategies (with or without pruning), we allow a limit of 2 hours of running time on each test suite.

5 Results and Discussion

The results of the method learning experiments (Figure 4a) show that Teachable-HTN-Maker learns significantly fewer methods in less time than HTN-Maker (both with and without pruning). HTN-Maker's run time increases with problem size, such that it cannot solve problems with larger than a certain amount of blocks or packages. The evaluation results (Figure 4b) show that the HTN planner takes significantly less time to solve more problems using the methods learned by Teachable-HTN-Maker (both with and without pruning).

In the Logistics domain, the number of methods learned increases exponentially with the number of packages. This is caused by the existence of alternative ways to bind variable names to object names when learning methods for DelivernPkg tasks. More specifically, when learning methods for the task Deliver-2Pkg from a solution plan that first has package A then package B delivered to the destination, there are 3 possible ways to bind the object names in the domain to the variable names in the annotated task: 1) o1 to A and o2 to B, 2) o1 to B and o2 to A, or 3) both o1 and o2 to B. We have not yet implemented a solution to prevent unwanted name binding or to prune the unwanted methods caused by unwanted name binding. Nevertheless, Teachable-HTN-Maker still learns fewer methods with the same deficiency.

HTN-Maker nondeterministically chooses subtask groupings to form methods when there are several possibilities. The implementation tested in the evaluations caused the algorithm to make deliberate choices when a method decompositions is learned from right to left, such that the right

```
(:task
    Deliver-1Pkg
  :parameters
  (?o - obj
   ?d - location)
  :precondition
  (and)
  :effect
  (and (at ?o1 ?d)))
```

```
(:task Deliver-2Pkg
  :parameters
  (?o1 - obj
   ?o2 - obj
   ?d - location)
 :precondition
 (and)
  :effect
 (and (at ?o1 ?d)
(at ?o2 ?d)))
```

Figure 3: Example annotated tasks in the Logistics domain.

subtask (if any) always corresponds to a previously learned method instance that extends over the largest subplan (if there are multiple ones). For the Blocks World example problem described previously (Figure 1), the method learned from the 7th curriculum step (Figure 5) effectively moves a stack of blocks *b* and *a* from above *c* onto table while maintaining the order by dividing the task into two subtasks that respectively reach state s_1 and s_2. The learned method has `((MAKE-2PILE ?a ?b) (MAKE-2PILE ?b ?a))` as subtasks (Figure 5). Respectively, the original HTN-Maker learns a method that has `((UNSTACK ?a ?b) (MAKE-2PILE ?b ?a))` as subtasks, where the subtask `(MAKE-2PILE ?b ?a)` takes the remaining 7 out of 8 total actions. The method learned with the curriculum is conceptually more desirable.

In the Blocks World domain, the plans produced by the planner using methods learned by HTN-Maker are slightly shorter than the plans produced by the planner using methods learned by Learnable-HTN-Maker. For example, to move a stack of 2 blocks A and B (over C) onto the table while maintaining the order (Figure 1), it takes 6 actions: unstack A from B, put down A, unstack B from C, put down B, pick up A, and stack A on B. On the contrary, the plan produced by the planner using methods learned by Teachable-HTN-Maker has 8 actions. As the number of blocks increases, the length difference between the plans decreases by percentage. However, it takes significantly longer time to find the plans using methods learned by HTN-Maker.

In the Logistics domain, the planner couldn't solve problems with more than 3 packages using the methods learned by HTN-Maker, and with the pruned methods couldn't solve any of the problems. In contrast, when using methods learned by Teachable-HTN-Maker (with or without pruning), the planner always found a solution. In Figure 4(b), notice that when the HTN planner used Teachable-HTN-Maker's methods (without pruning) for 5 packages, its average running time was relatively large. In another set of experimental runs (not shown here) on the same problems, its average running time was much smaller. We believe the variation in running time was because the HTN planner randomized its choices among applicable HTN methods. We will further investigate this in the future.

6 Conclusions

We have described Teachable-HTN-Maker, a modified version of HTN-Maker learns using a curriculum. Our prelim-

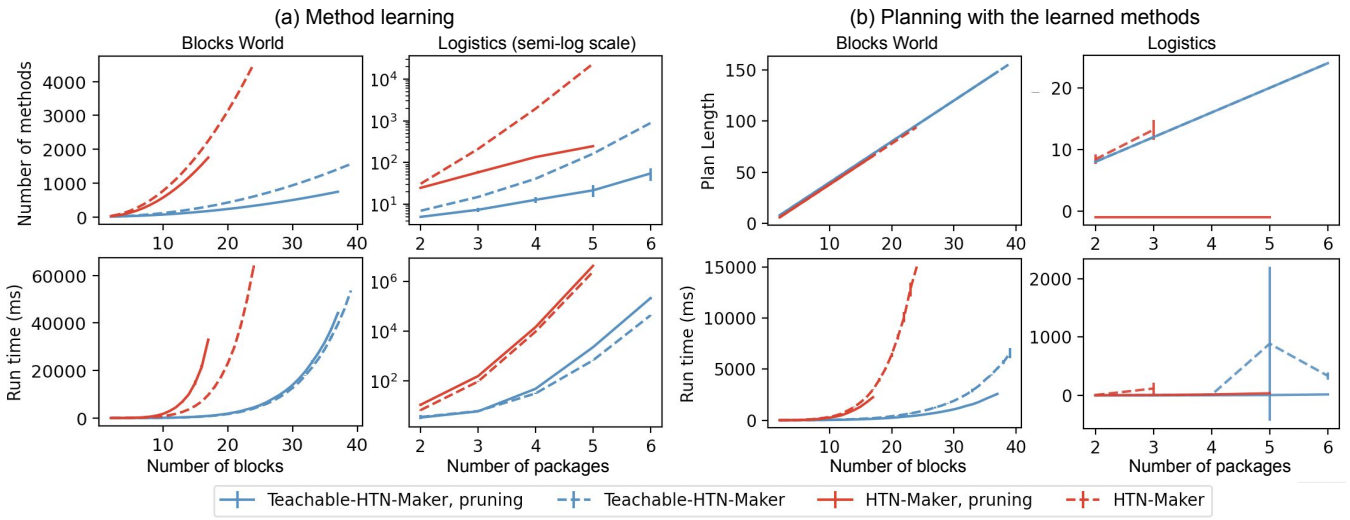

Figure 4: The plots show (a) average number of methods learned and running time for HTN-Maker and Teachable-HTN-Maker, both with and without pruning; and (b) the HTN planner's average plan length and running time on the same problems using the learned methods. In the Blocks World problems, each problem is to move a stack of n blocks, maintaining their order; and each data point is the average of 20 randomly generated problems of size n ($2 \leqslant n \leqslant 40$). In the Logistics problems, each problem is to deliver n packages, and each data point is the average of 5 randomly generated problems of size n ($2 \leqslant n \leqslant 10$).

On some of the larger problems, no results are shown for HTN-Maker or the HTN planner using the HTN-Maker methods because our 2-hour time limit (see "Methods" in the main text) was exceeded before reaching those problems.

In (b), the zero values for HTN-Maker with pruning in the Logistics problems mean that the HTN planner could not solve the problems using the learned methods.

```
(:method MAKE-2PILE
 :parameters (?b - BLOCK ?a - BLOCK)
 :vars (?c - BLOCK)
 :precondition(and (not (= ?b ?a))
 (ON ?b ?c) (ON ?a ?b) (not (= ?a ?c))
 (not(= ?b ?c)) (CLEAR ?a) (HAND-EMPTY))
 :subtasks ((MAKE-2PILE ?a ?b)
   (MAKE-2PILE ?b ?a)))
```

Figure 5: Final method learned for Figure 1.

inary experiments in the Blocks World domain and Logistics domain show that it learns significantly fewer and better methods with less computational effort.

Future Work In future work, we will evaluate our approach in more sophisticated problems with more varieties of annotated tasks and in more domains, e.g., the Depots, Zeno Travel, and Satellite domains, and domains from the 2020 IPC for Hierarchical Planning (Höller et al. 2020). It is both a challenge and an opportunity to figure out how to generate optimal curriculum as the planning problem gets more sophisticated. We will also consider evaluating how different curricula influence the performance of Teachable-HTN-Maker, and evaluate the methods in different problems of the same kind instead of only evaluating them in the exactly same problem where they were learned.

The annotated tasks input to HTN-Maker do not specify any precondition, this allows HTN-Maker to learn methods for the same annotated task from subtraces with different starting positions. The learned methods would respectively have different preconditions, including the ones we are not interested in. A straightforward augmentation to HTN-Maker is to specify preconditions in the annotated tasks and let those preconditions to be checked when learning methods from a subtraces. We would like to compare our approach with the augmented version.

Method instances learned from different subtraces can be generalized into the same lifted method (e.g., the method instances learned from step 1 and 4 of Blocks World curriculum). However, HTN-Maker cannot recognize the semantic equivalence among those subtraces, and therefore spends unnecessary computational resources on learning the same lifted methods from such subtraces. We hope to significantly strengthen the system by making it learn each *lifted* method only once. This is analogous to DreamCoder (Ellis et al. 2020), which learns *concepts* incrementally.

Acknowledgments. At UMD, this work has been supported in part by ONR grant N000142012257 and NRL grants N0017320P0399 and N00173191G001. At NRL, Mark Roberts thanks ONR and NRL for funding this research. The information in this paper does not necessarily reflect the position or policy of the funders, and no official endorsement should be inferred.

References

Bengio, Y.; Louradour, J.; Collobert, R.; and Weston, J. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, 41–48.

Choi, D.; and Langley, P. 2005. Learning teleoreactive logic programs from problem solving. In *International Conference on Inductive Logic Programming*, 51–68. Springer.

Ellis, K.; Wong, C.; Nye, M.; Sable-Meyer, M.; Cary, L.; Morales, L.; Hewitt, L.; Solar-Lezama, A.; and Tenenbaum, J. B. 2020. Dreamcoder: Growing generalizable, interpretable knowledge with wake-sleep bayesian program learning. *arXiv preprint arXiv:2006.08381*.

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

Höller, D.; Behnke, G.; Bercher, P.; Biundo, S.; Fiorino, H.; Pellier, D.; and Alford, R. 2020. HDDL: An Extension to PDDL for Expressing Hierarchical Planning Problems. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI 2020)*, 9883–9891. AAAI Press.

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

Nau, D.; Cao, Y.; Lotem, A.; and Munoz-Avila, H. 1999. SHOP: Simple hierarchical ordered planner. In *Proceedings of the 16th international joint conference on Artificial intelligence-Volume 2*, 968–973.

Xiao, Z.; Wan, H.; Zhuo, H. H.; Herzig, A.; Perrussel, L.; and Chen, P. 2020. Refining HTN Methods via Task Insertion with Preferences. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 10009–10016.

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.