Hypothesizing Method Structure for HTN Learning from Demonstration

Michael Leece MLEECE@UCSC.EDU
Arnav Jhala AHJHALA@NCSU.EDU

Abstract

One of the key capabilities of a cognitive system is the ability to plan and reason at some abstract level. Hierarchical Task Networks are one potential planning framework that provides this capability, but are severely limited by the knowledge acquisition bottleneck. There are many components required in learning HTN models from demonstration, but one of the first steps must be generating a method structure to build on. Hypothesizing this decomposition has frequently been simple, thus pushing the responsibility of dealing with an inaccurate model to components further down the learning pipeline, or ignored, by assuming the decomposition structure is included with the input. We introduce two methods for hypothesizing method structure and evaluate them for coverage and signal against a greedy approach used in prior work. Finally, we discuss ways for these methods to be extended and improved in the future.

1. Introduction

Hierarchical Task Networks (HTN) are a popular planning framework that allow great power and flexibility in encoding abstract knowledge. By searching the plan space at various levels of abstraction, they are more quickly able to find effective plans than traditional search using primitive operators only.

However, HTNs have a significant drawback that has limited their use in practice. In particular, they suffer from a heavy knowledge engineering bottleneck. In order to create a functional HTN planner for a given problem area, a domain expert must encode all of the tasks that could be requested of the planner, as well as the ways each of those tasks can break down into sub-tasks, under which conditions which decomposition should be applied, and then the same for each of those subtasks, and so on. In a complex domain, this engineering effort quickly becomes a daunting task. This direct encoding is further complicated by the fact that for some domains, human experts experience trouble consciously thinking of the reasons for the decisions they make, as demonstrated in Johnson (1983). In the meantime, humans have the ability to learn this structure from observation, by identifying patterns and abstractions over multiple demonstrations.

This bottleneck is a well-known problem, and has motivated research into automated retrieval of the knowledge required. Some work has focused on building up a domain model incrementally by solving iteratively harder problems, but this disregards the fact that for many domains, there exists a wealth of expert demonstrations, though not encoded expert knowledge. Other work has worked toward the goal of learning HTN domains from demonstration, but generally assume that there is some amount of expert labeling still occurring.

In short, much of the existing work in HTN learning has assumed *what* tasks are being pursued with which actions is given, and has focused on learning *how* to achieve those tasks: when each decomposition is applicable, ordering constraints, etc. From a knowledge engineering reduction standpoint, this is understandable—it is much easier for someone to label a trace by simply saying "This subset of actions was me taking the bus to work" than including every factor that went into that decision: car was out of gas, it was raining, and the umbrella is broken, leaving those to be learned by the system through observation.

However, we wish to reach the point where systems can learn a full domain model purely from observation of primitive actions. On the other hand, if we wish to leverage the existing systems, we can decouple the components of the learning problem, and attempt to hypothesize a method structure for downstream components that assume it as given. Prior work that has taken this approach has used simple algorithms for hypothesizing method structures that lack flexibility, and struggle to deal with interleaved task expansions. We present two new algorithms for duplicating the human cognitive ability of acquiring planning structure, and attempt to characterize their strengths and weaknesses.

2. Related Work

Yang et al. (2007) presents an Expectation Maximization approach to method learning that introduces the HTN Task Cluster Model (HTCM), modeling HTN tasks as Markov chains. This allows them to calculate probabilities of chains belonging to a given task, with the additional assumption that the input includes which tasks are achieved by which traces.

Li et al. (2009a) uses a greedy approach to structure hypothesizing while attempting to learn probabilistic HTNs to simulate user preferences. This took the most common pair of actions across all traces and replaced it with an abstract task, and repeated, with a special case check for recursive structures. This generates a grammar in Chomsky Normal Form, allowing them to use pCFG learning techniques for learning method expansion probabilities.

Leece & Jhala (2014) present the beginnings of a pattern mining approach to learning method structures for the real-time strategy video game domain. However, they only analyze the first level of generated methods, which qualitatively appear to be accurate.

HTN-MAKER, developed in Hogg & Munoz-Avila (2007) and Hogg et al. (2008) is another example of a system for learning HTNs from demonstration. While it does not assume the hierarchical structure of traces as given, it does take the abstract tasks of the domain, in the form of pre- and post-conditions, from which it learns method decompositions to achieve these tasks by searching for sequences of actions in demonstrations that achieve these tasks.

Nejati et al. (2006) approaches a similar problem of learning HTNs from demonstration without hierarchical structure, but assume that some information about the final goal is provided, from which they construct their learned methods incrementally.

Within the greater cognitive architecture research fields, the major architectures–which are not explicitly utilizing HTNs, but are using similar planning processes–have also addressed this issue.

Könik & Laird (2006) and van Lent & Laird (2001) both explore learning from experts in the context of the Soar architecture. However, it is a more interactive process that our problem, with the expert annotating traces generated by the planner, or analyzing and annotating the agent's self-annotations. Li et al. (2009b) confront a very similar problem to ours within the context of the ICARUS architecture.

Finally, there are a number of other systems that perform HTN learning, but most presume structure as a given. Garland et al. (2001) proves soundness and completeness for learning bindings, parameters, and constraints, given that the hierarchical decomposition for all traces is provided as input. Zhuo et al. (2009) approaches the problem of dealing with partial state observations when learning preconditions and an action model, but also assume that decomposition tree information is given.

3. Hierarchical Task Networks

Terminology: An HTN *domain* $\mathcal{H}$ consists of a 3-tuple: $\mathcal{H} = \langle \mathcal{A}, \mathcal{T}, \mathcal{M} \rangle$. $\mathcal{A}$ is the set of *primitive actions* in the domain, $\mathcal{T}$ is the set of abstract *tasks*, and $\mathcal{M}$ is the set of *methods* that achieve the elements of $\mathcal{T}$. A method has four components: a unique identifier, which abstract task it achieves, a decomposition into subtasks, and a set of constraints, which can consist of preconditions, postconditions, and conditions that must hold throughout the execution of the method. An HTN *planning problem*, then, consists of an initial state, a set of tasks to be achieved, and an HTN domain. It is then the goal of the HTN planner to find a valid plan that achieves all of the tasks using the decompositions provided in the domain methods, while maintaining all constraints imposed by those methods. Learning these decompositions is the target of this work.

4. Algorithm Descriptions

4.1 Baseline

Our baseline is the Structure Hypothesizer algorithm from Li et al. (2009a), which we will summarize briefly for the reader.

Given a library of plans for achieving some high-level task, first search for evidence of recursive expansion rules. This evidence is of the form of long chains of a repeated action, followed or preceded by some other action. For example, if the pattern $aa \ldots ab$ occurs frequently in the library, it would imply the recursive rule $B \rightarrow aB; B \rightarrow b$. If the frequency and average length of a recursive pattern are both above some user-defined threshold, add the recursive expansion to the HTN method library and replace each instance of it with a new abstract symbol.

If no recursive pattern exceeds the thresholds, find the most frequent pair of adjacent actions in the plan library. Add a task to the HTN that has a single expansion corresponding to this pair of actions, and replace all instances of the pair in the plan library with the newly added task.

This process is repeated until all traces in the plan library have been compressed to a single task, the original high-level task.

4.2 EM Algorithm

For our first algorithm, we begin by making the weak assumption that the decompositions for a given task in an HTN may be modeled as a Markov chain. This is valid for simple hierarchies—for example, where each task has only one method decomposition with non-overlapping subtasks—and it even allows for some complexities in expansions, allowing a probabilistic choice between multiple method expansions, so long as they do not overlap. On the other hand, while it may be possible to set up a Markov chain to model an HTN, this particular view of task decomposition is clearly not able to model the full complexity of HTN tasks. A simple demonstration of this would be two methods for a task which pass through the same subtask. It would be possible for the Markov chain to switch tracks from the first method to the second, even if this led to an invalid plan.

However, this assumption does let us calculate the probability of a task T decomposition generating a sequence of n actions l, given the prior and transition distributions Θ_T for that task:

$$P(l|\Theta_T) = P(l_0|\Theta_T) * \prod_{i=1}^{n} P(l_i|l_{i-1}, \Theta_T) \tag{1}$$

The original EM method-learning algorithm presented in Yang et al. (2007) presumes that all abstract tasks formulated in a given trace would be included, in order, with the input. Thus, the primary goal for the algorithm was determining break points in the chain of primitive actions. With these, the provided tasks could be overlaid on the appropriate subsequences and learned from the collection of appearances of that task in the trace library.

Our relaxation of the input assumptions, in which the algorithm receives only the top-level task and the trace of primitive actions, eliminates this approach. We introduce a set of latent abstract tasks Y (line 4 of Algorithm 2), representing the HTN tasks that the top-level task decomposes into, but which are not explicitly provided for us. As mentioned, these are modeled as Markov chains, and so are represented with a prior and transition distribution ($P(a_0|\Theta_T)$ and $P(a_t|a_{t-1}, \Theta_T)$, respectively).

Lines 6 and 7 represent the Expectation-Maximization loop of the algorithm. In the terminology of EM, we first calculate the most likely explanation for which tasks generated which action sequences using our current model parameters (stored in $taskSequences$), then we update our model parameters based on the actions that have been assigned to them.

To be more precise, we use a Viterbi-like dynamic programming algorithm to calculate the most probable assignment of primitive actions to tasks for each trace in the library. In the classic Viterbi algorithm, we calculate the probability of the most likely trace ending in a particular state and time step by calculating the probability of the most likely traces to each state in the time step prior, and taking the best of each of those when multiplied with the associated transition probability.

A similar approach for us, if our goal is to assign actions in some trace l to tasks, rather than states, would result in the following recursive definition:

$$OPT(i,t) = \max(OPT(i-1,t) * P(l_i|l_{i-1}, Y_t), \max_{u \in Y - \{t\}}(OPT(i-1,u) * P(l_i|Y_u))) \tag{2}$$

where $OPT(i,t)$ is the probability of the most likely assignment that assigns the i-th action to task t.

However, this definition will fail when given ground out traces of partially-ordered plans. Namely, it assumes that an action either belongs to the same task as the previous action, or is the start of a new task. This corresponds to the first-order Markov assumption in the Viterbi algorithm.

Our solution is to introduce a parameter k that determines the maximum number of intermediate actions allowed before two actions may no longer be considered as part of the same task. The higher this parameter, the more capable the algorithm is of dealing with mixed action groundings. However, there is a run-time tradeoff as well, limiting k in practice.

This changes the formulation of the recursive definition to the following:

$$OPT(i, t, s) = \max_{u \in T}(OPT(i-1, s[-1], [u] + s[:-1]) * nextTaskProb(l_i, t, [u] + s[:-1]) \quad (3)$$

where

- s is a suffix of length k representing the task assignments of the previous k actions.
- $nextTaskProb$ is a function that takes an action a, a task t, and a suffix s, and calculates the probability of that action being assigned to that task given that it follows the suffix. This is $P(a|t)$ if none of the actions in the suffix are assigned to t, and otherwise is the transition probability from the latest t-assigned action to a.

Using standard backtracking techniques, we can then calculate the most probable assignment of actions to tasks for every trace. With these, we update the parameters of our model Y to maximize the probability of each task generating the action subsequences that it has been assigned according to Equation 1 (line 7 of Algorithm 2).

Once the EM loop has converged, and the parameters of the model Y are no longer changing (or changes are below some threshold), we take the task model in Y with the lowest weighted entropy, representing the simplest learned model. In the extreme case of zero entropy, this corresponds to a sequence of actions that always occurs in a set order, which corresponds to a single-method task. We add this task to the task library, and generate a set of methods to achieve it, using one of two approaches. The first is to simply add every sequence assigned to this latent task in the final expectation step; that is, every instance of this task in the most probable assignment of actions to tasks (hereafter referred to as the Most Probable Explanation (MPE) approach). The second is finding all walks in the Markov chain with probability greater than some user-defined threshold Θ_m. Finally, we replace the subsequences in L that were assigned to this task with an abstract symbol for the task, clean up L (adding any single-task traces as expansions for the top-level task), and reiterate.

We would like to make a note that this algorithm is a hard EM algorithm, using the most probable assignment as labels when doing the maximization step, rather than a probability distribution. A soft EM approach may be feasible as well, using an algorithm similar to the forward-backward algorithm. However, the complexity would increase, as we would need a posterior probability distribution for transitions, rather than simply assignments, and this is complicated by the interleaving of assignments.

4.3 Pattern Mining Algorithm

The second approach we present for hypothesizing structure is a generalization of the greedy approaches used in prior work. The underlying assumption is that subsequences of actions that are both common and closely linked in the time domain are the best candidates for HTN methods. When taken to the extreme, this results in taking the most frequently occurring pair of actions as a task method, replacing them with an abstract task, and repeating until all traces have condensed to a single task.

However, this approach is not robust when presented with traces from partially-ordered plans. In particular, it may require an exponential number of pairs to explain plans in which tasks are overlapped.

Our solution to this problem is to use the Generalized Sequential Pattern (GSP) algorithm developed by Srikant & Agrawal (1996), in replacement of simple pair-matching. GSP is a pattern mining algorithm that takes a database of traces and searches for common subsequences. These common subsequences act as our HTN method candidates, and we can replace them with an abstract task in L and iterate to find higher-level tasks (lines 7-10 of Algorithm 1). In addition, it includes two parameters that will be useful to us:

- Minimum Support - The minimum amount of traces a subsequence must appear in to be considered a frequent pattern
- Maximum Gap - The maximum gap allowed between elements of a subsequence. If 0, the subsequence must be contiguous. If the length of the trace, any subsequence is allowed.

By relaxing these two parameters (line 13), we can identify additional subsequences, and consequently more HTN method candidates. Our evaluation tests two different approaches to this relaxation strategy.

5. Evaluation

5.1 Synthetic Domains

For evaluation, we used randomly generated synthetic planning domains with a single high-level task, for two primary reasons. First, it provides an oracle planner that can be exhaustively searched to generate all valid plans. Second, it allows us to run a suite of tests and therefore achieve stronger confidence in our results than if we were confined to a single domain.

Some domain characteristics for the randomly generated domains used in these evaluations are as follows:

- Depth of plan decomposition tree: Between 3 and 8. Median depth of 5.
- Method expansions per task: Randomly chosen between 1, 2 and 3
- Subtasks per method: Randomly chosen between 2 and 3

Algorithm 1: GSP Structure Hypothesizer algorithm

```
Input  : $L$ : library of action traces for the same top-level task
Output: $T$ : A set of HTN tasks
        $M$ : A library of method expansions for $T$
1  $T \leftarrow \emptyset$
2  $M \leftarrow \emptyset$
3  while $|L| > 0$ do
4      while GSP(L) $\neq \emptyset$ do
5          $patterns \leftarrow GSP(L)$
6          foreach $p \in patterns$ do
7              $t \leftarrow nextAbstractSymbol()$
8              $T \leftarrow T + \{t\}$
9              $M \leftarrow M + \{(t, p)\}$
10             $L \leftarrow replaceTask(p, t)$
11         end
12     end
13     Relax Minimum Support or Maximum Gap
14 end
```

Algorithm 2: EM Structure Hypothesizer algorithm

```
Input  : $L$ : library of action traces for the same top-level task
Output: $T$ : A set of HTN tasks
        $M$ : A library of method expansions for $T$
1  $T \leftarrow \emptyset$
2  $M \leftarrow \emptyset$
3  while $|L| > 0$ do
4      Initialize latent tasks $Y$ randomly
5      while Y not converged do
6          $taskSequences \leftarrow viterbiBacktrack(L, Y)$
7          $Y \leftarrow updateTasks(taskSequences)$
8      end
9      $t_r \leftarrow lowestEntropy(Y)$
10     $T \leftarrow T + \{t_r\}$
11     $M \leftarrow M + generateMethods(t_r)$
12     $L \leftarrow replaceTask(t_r)$
13 end
```

In truth, this only explores there a small fraction of the potential space of HTN planning domains, and it is possible that different results would be obtained in domains with different characteristics. However, we feel that the synthetic domains were sufficiently complex to present a challenge to learning systems.

Using these domains, we generated primitive action traces by sampling the plan space for partially-ordered plans and their groundings, using a simple probabilistic check ($p = 0.1$ in our evaluation) when expanding a task to decide whether or not to interleave its subtasks' actions when grounding. These formed a demonstration library of primitive action traces that could be fed to the algorithms for hypothesizing method structure.

We introduce two metrics that we will be using to evaluate the effectiveness of the hypothesized method structures.

- *Coverage* — A measurement of how well the hypothesized structures cover the valid plans of the oracle. The fraction of valid plans generated by the oracle that exist as a grounding of a partially-ordered plan in the learned structure.

- *Signal* — A measurement of how much noise is in the plans of the hypothesized structure. The fraction of partially-ordered plans generated by the learned structure that can be ground to a valid oracle plan.

Note that these correspond loosely to the precision and recall measures from classification, adjusted to fit our problem.

As the envisioned first step in a learning pipeline, it is clear that our first target is high coverage, with a secondary goal of high signal. This is because future learning steps will be pruning the method structure via learning applicability conditions, and in doing so therefore remove large numbers of potential plans that do not correspond to valid oracle plans. On the other hand, we can see that a simple greedy approach such as our baseline will maximize signal, as it only builds structure that can expand into exact traces that it has observed. However, this hurts its generalization ability in being able to generate unseen plans.

5.2 Method Selection for EM

Table 1 shows the average coverage and signal of a number of parameter configurations for the EM algorithm over 12 different randomly generated domains. The two different configurations refer to the approaches used to generate methods for the abstract task once EM has been run and the lowest-entropy task selected. MPE uses all action subsequences assigned to the task in the final Expectation step as methods, while Threshold uses all walks through the final Markov chain that are above a given probability Θ_m.

MPE has the advantage of having one less parameter to tune (we discuss α in the following section), and appears to be consistently stronger than the baseline. The inherent risk of this approach is of a task's Markov chain acquiring responsibility for explaining some extraneous action sequences that match no task well during the EM process, and these being added as a method expansion while not reflecting the main task the representation has learned. However, this risk is mitigated, though not entirely removed, by choosing the lowest entropy task.

Algorithm			**Coverage**	**Signal**
Baseline			0.491 ± 0.101	1.00 ± 0.00
EM MPE	α			
	1		0.662 ± 0.045	0.043 ± 0.03
	0.1		0.695 ± 0.055	0.027 ± 0.012
	0		0.662 ± 0.048	0.072 ± 0.017
EM Threshold	α	Θ_m		
	1	0.05	0.420 ± 0.082	0.662 ± 0.070
	1	0.01	0.390 ± 0.147	0.138 ± 0.049
	0	0.1	0.525 ± 0.052	0.450 ± 0.086
	0	0.05	0.620 ± 0.094	0.052 ± 0.024
	0	0.01	0.827 ± 0.083	0.024 ± 0.019
GSP			0.532 ± 0.090	0.983 ± 0.02

Table 1. Coverage and Signal for a range of algorithm configurations over 12 random domains, with a demonstration library of 200 plans. α is the additive smoothing parameter for Laplace estimation in Maximization-step updates, and Θ_m is the probability threshold for paths in the Markov chain to be added as methods.

Thresholding removes this risk, as it will only take the major action paths that have been learned (though it retains the risk of a task learning to represent two non-colliding oracle tasks and thus merging them incorrectly). However, it performed much less consistently, responding drastically to changes in the threshold parameter. In a domain with no oracle to test against, it may be difficult to determine whether one is using the correct parameter values.

Note that if no walk exceeded the threshold probability, the highest probability walk was used as the only method expansion for that task. In the $\Theta_m = 0.1$ case, this behavior led the algorithm to a structure much closer to the baseline's single method per task expansion, resulting in a higher signal, at the cost of a mediocre coverage.

A third possibility for generating task methods from a Markov chain is using Monte Carlo rollouts. In testing, we discovered that this approach performed significantly worse than either of the prior two. Due to the nature of the EM algorithm, each latent task absorbs some amount of 'noise', whether it be true noise or the result of higher-level structures that can't currently be modeled. Choosing the lowest-entropy task means that if the rollout *does* leave the common paths, it tends to meander among actions randomly until reaching a common chain again. This generates a very long and low-quality method.

5.3 Smoothing

During the Maximization step of the EM algorithm, we perform an update to each task's Markov chain representation based on the action sequences assigned to it by the previous Expectation step. We tested the performance of the algorithm with and without additive smoothing during this update. When $\alpha = 0$, the update is a maximum likelihood update, and has a tendency to converge more rapidly and rigidly (if an action transition is not present in a step's subsequences, it will never be assigned to this task again). When $\alpha = 1$, we include a 'pseudo-observation' of each action transition, a slightly more forgiving approach.

We observe in Table 1 that a small amount of smoothing was marginally (though not statistically significantly) beneficial to the MPE version of the algorithm, but it was severely detrimental to the coverage of the Thresholding configuration. We believe the reason for this is that HTN tasks, once they have been expanded into methods, are in fact fairly rigid processes. As a result, leaving them flexible is an inaccurate representation of the underlying mechanic. One avenue for further exploration is to reduce α over the course of the EM algorithm, to allow flexibility in the beginning when latent tasks are finding their identity, and rigidity toward the end, when they are solidifying which action sequences they account for.

5.4 GSP

Two different procedures for relaxing the minimum support and maximum gap parameters of the GSP algorithm were tested. Using an approach similar to grid search, one parameter was slowly relaxed until a defined limit was reached, then tightened again and the other parameter relaxed by a single increment. Changing which parameter belonged to which 'dimension' of the search resulted in two different learned method structures.

Unfortunately, due to a lack of branching factor introduction to the hypothesized methods, each structure was only able to reproduce the traces provided to it, and so their metrics were identical (though the expansions to reproduce those traces were different). As a result, the GSP algorithms were only able to slightly outperform the baseline approach, likely due to their ability to deal with interleaved actions due to searching for subsequences rather than contiguous pairs.

5.5 Size of Plan Library

Figure 1 shows the progression of coverage as the algorithms receive greater numbers of demonstration plans from the oracle. As expected, both the Baseline and GSP are relatively linear, with low generalization ability. On the other hand, the EM algorithms both improve very quickly, implying that they are learning substructures across demonstrations that can be used to generate the valid plans that have not yet been observed.

6. Future Work

As mentioned, these algorithms provide only one component of a full HTN learning system. The true evaluation of their efficacy is as part of a complete unit that performs learning from start (unlabeled primitive traces) to finish (full HTN model with tasks, methods, preconditions, etc.).

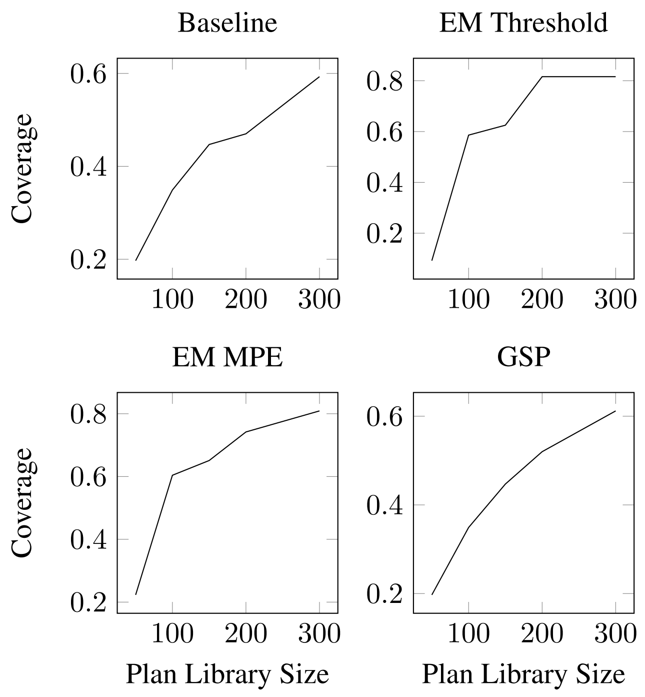

Figure 1. Change in coverage as the size of the provided plan library increases. The total number of valid oracle plans was 400.

Therefore, the most immediate task is to integrate this output into a pipeline that leverages existing components from prior research.

Beyond that, we would like to develop a mathematically principled approach for tuning the maximum gap width and minimum support threshold for the GSP algorithm, rather than the grid search used in this work. Additionally, GSP suffers from a similar problem as the greedy algorithms, namely, that tasks other than the high-level one tend to have exactly one method expansion. This leads to high signal, but low coverage, which we would like to address in the future.

One of the original motivations for this work was for learning from human demonstrations in complex domains, in which a lack of labeled data, interleaved tasks and noise all cause significant problems for the existing HTN learning systems. In light of this, we would like to evaluate our approach with human demonstrations, though we would need to rely on domain performance as evaluation, with no access to an oracle to compare against.

7. Conclusion

In conclusion, we have presented two algorithms for hypothesizing method structure for an HTN domain that rely purely on primitive action traces. One slightly outperforms the greedy baseline as a generalized version better able to handle interleaved actions, while the other significantly outperforms it in learning unseen plans via substructures.

This comes at the cost of introducing a number of possible plans that are not valid, and so in order to be effective, these algorithms must be filling a role as the start of a learning pipeline, rather than standalone. However, if they are, they will provide a structure from which more domain information can be extracted and pruned.

References

Garland, A., Ryall, K., & Rich, C. (2001). Learning hierarchical task models by defining and refining examples. *Proceedings of the 1st international conference on Knowledge capture* (pp. 44–51). ACM.

Hogg, C., & Munoz-Avila, H. (2007). Learning hierarchical task networks from plan traces. *Proceedings of the ICAPS-07 Workshop on AI Planning and Learning*.

Hogg, C., Munoz-Avila, H., & Kuter, U. (2008). HTN-MAKER: Learning HTNs with minimal additional knowledge engineering required. *AAAI* (pp. 950–956).

Johnson, P. E. (1983). What kind of expert should a system be? *Journal of Medicine and Philosophy*, *8*, 77–97.

Könik, T., & Laird, J. E. (2006). Learning goal hierarchies from structured observations and expert annotations. *Machine Learning*, *64*, 263–287.

Leece, M., & Jhala, A. (2014). Sequential pattern mining in starcraft: Brood war for short and long-term goals.

van Lent, M., & Laird, J. E. (2001). Learning procedural knowledge through observation. *Proceedings of the 1st international conference on Knowledge capture* (pp. 179–186). ACM.

Li, N., Kambhampati, S., & Yoon, S. W. (2009a). Learning probabilistic hierarchical task networks to capture user preferences. *IJCAI* (pp. 1754–1759).

Li, N., Stracuzzi, D. J., Langley, P., & Nejati, N. (2009b). Learning hierarchical skills from problem solutions using means-ends analysis. *Proceedings of the 31st Annual Meeting of the Cognitive Science Society. Amsterdam, Netherlands: Cognitive Science Society, Inc.*

Nejati, N., Langley, P., & Konik, T. (2006). Learning hierarchical task networks by observation. *Proceedings of the 23rd international conference on Machine learning* (pp. 665–672). ACM.

Srikant, R., & Agrawal, R. (1996). *Mining sequential patterns: Generalizations and performance improvements*. Springer.

Yang, Q., Pan, R., & Pan, S. J. (2007). Learning recursive HTN-method structures for planning. *Proceedings of the ICAPS-07 Workshop on AI Planning and Learning*.

Zhuo, H. H., Hu, D. H., Hogg, C., Yang, Q., & Munoz-Avila, H. (2009). Learning HTN method preconditions and action models from partial observations. *IJCAI* (pp. 1804–1810).