Learning Hierarchical Task Networks with Preferences from Unannotated Demonstrations

Kevin Chen* **Nithin Shrivatsav*** **David Kent***

Harish Ravichandar **Sonia Chernova**

Georgia Institute of Technology
{*kchen367, nithinshri, dekent, harish.ravichandar, chernova*}*@gatech.edu*

Abstract: We address the problem of learning Hierarchical Task Networks (HTNs) from unannotated task demonstrations, while retaining action execution preferences present in the demonstration data. We show that the problem of learning a complex HTN structure can be made analogous to the problem of series/parallel reduction of resistor networks, a foundational concept in Electrical Engineering. Based on this finding, we present the CircuitHTN algorithm, which constructs an action graph representing the demonstrations, and then reduces the graph following rules for reducing combination electrical circuits. Evaluation on real-world household kitchen tasks shows that CircuitHTN outperforms prior work in task structure and preference learning, successfully handling large data sets and exhibiting similar action selection preferences as the demonstrations.

Keywords: Hierarchical Task Modeling, Learning from Demonstration, Preference Learning

1 Introduction

Hierarchical planning leverages the hierarchical structure inherent in complex tasks to facilitate greater scalability and task decomposition compared to traditional planning techniques. One of the most widely used hierarchical planning representations are *hierarchical task networks (HTNs)* [1], which have been applied in robotics in the context of navigation [2], manipulation [3, 4], partially observable planning [5], and human-robot collaborative tasks [6, 7, 8].

Until recently, HTN models have been largely hand-crafted [9, 10]. However, hand coding an HTN for complex tasks is difficult, as it requires significant domain knowledge and engineering effort. Recent work has sought to address this problem through automated learning methods, though such approaches often require significant hand-labeling task hierarchy in the form of task segmentation or non-primitive task annotation [11, 12, 13]. Approaches that learn hierarchical structure from unannotated data are less common, and either do not learn alternative decompositions [14] or preferences [3], or learn highly rigid task structures that impede model interpretability [15].

In this work, we introduce *CircuitHTN*, a novel approach for learning hierarchical task structure from unannotated task demonstrations (i.e. sequences of state-action pairs obtained from observations of a task, with no segmentation or hierarchical action annotation). Our approach surpasses prior work by producing sound task models capable of reproducing all demonstrated approaches for completing a task. Additionally, our approach learns to imitate action selection preferences reflected within the demonstrations, a capability that is key for applications that require modeling an agent's likely behavior, such as action prediction and cooperative tasks.

A key insight of our work, which enables the above benefits, is that we can learn HTN structures through an iterative directed acyclic graph reduction approach by analogy to the Electrical Engineering problem of reduction of resistor networks. Specifically, we show that sequential plan actions can be modeled as series circuits, and decision points within plans can be modeled as parallel circuits. This unique reformulation of the problem enables our approach to learn more accurate and scalable

*Denotes equal contribution

4th Conference on Robot Learning (CoRL 2020), Cambridge MA, USA.

task models by identifying and abstracting groups of actions as one can abstract groups of resistors into a simpler equivalent structure. Certain tasks with extensive optional execution strategies, however, produce densely connected action graphs that are too complex to model as combination circuits. For such cases, we provide an extension to CircuitHTN that converts densely connected action graphs into equivalent graphs suitable for series/parallel reduction.

We evaluate our approach in two domains: i) a simulated robot performing a table setting task that contains both hard and soft action ordering constraints, and ii) a complex partial-order dataset consisting of 50 demonstrations obtained from motion capture data of users preparing a salad (CMU 50Salads dataset [16]). Our results show that CircuitHTN surpasses prior state of the art [15, 3] with respect to scalability, preference alignment, and soundness of the learned models.

2 Related Work

Hierarchical task modeling encodes complex robotic behaviors in transparent, human-readable representations, such as HTNs [17, 4], behavior trees [18], and AND/OR graphs [19]. We focus specifically on the HTN learning problem. Mohseni-Kabir et al. [4] address the annotation problem using an interactive teaching process, in which a human teacher works with a robot to specify a hierarchical task. Their approach combines top-down and bottom-up reasoning to learn single-task networks, including multiple decompositions and optional steps. Our work instead aims to learn hierarchical task structure from *unannotated* task demonstrations, while also representing execution preferences inherent to the provided demonstrations. Hayes and Scassellati [3] present algorithms for learning particular execution structures in collaborative robot assembly tasks, called cliques and chains, from unannotated state-action sequences. French et al. [18] demonstrate unsupervised behavior tree learning from demonstration, evaluated on household robotics tasks. Both unsupervised approaches focus on learning sound task models, but do not represent demonstrator preferences within the task.

The field of AI and hierarchical planning provides more extensive coverage of HTN learning approaches, including top-down and bottom-up approaches for learning hierarchical structure, task preconditions, and action models from *annotated* task data. Garland and Lesh [20] generalize non-primitive actions to new problems in a planning domain. Yang et al. [11] use clustering to segment annotated demonstrations into non-primitive actions, though the machine-learned segmentations may be inaccurate. Neural task programming [21] identifies and composes hierarchical programs from execution traces, enabling end-to-end hierarchical task generalization. Algorithms such as CaMeL [22] focus on efficient precondition learning. Other approaches [12, 23, 24, 13] generalize examples from multiple problems to fully learn HTN planning domains. In all of the above work, annotations take the form of either labeled and segmented execution traces, lists of desired non-primitive tasks, or pre-determined non-primitive preconditions and effects.

We focus specifically on learning hierarchical task structure only, as there is already extensive prior work on learning action models from task demonstrations (e.g., [25]). While less common, there are HTN domain learning approaches designed to learn non-primitive action decompositions from unannotated data [26, 14], but they do not capture tasks with multiple valid recipes. An alternative approach is to model task hierarchy through probabilistic grammars [27, 15, 28]. Most similar to our work is that of Li et al. [15], which uses context free grammars (CFGs) to learn hierarchical task structure with probabilistic preferences for valid alternative task decompositions. However, the CFG formulation imposes strict constraints on the network structure of the resulting HTN, sacrificing the model transparency that would make HTNs beneficial for robotics applications.

3 Learning HTNs from Demonstration

We define the HTN learning problem as follows: Given a set of unannotated demonstrations, $D = \{\{s_k^i, a_k^i\}_{k=0}^{n_i}\}_{i=0}^{m}$, where (s_k, a_k) represents a state-action pair from m total task demonstration sequences of length n_i, our objective is to learn a probabilistic hierarchical task model that captures both valid task plans and the user's preferences concerning action selection. Specifically, we learn an HTN representation extended with concepts from AND/OR graphs [19] and behavior trees [18]. HTNs consist of primitive tasks, which correspond to directly executable robot actions, and non-primitive tasks, which can be decomposed into additional primitive and non-primitive tasks [1]. We include preference information by introducing two types of non-primitive nodes into the HTN: *decision nodes* and *sequence nodes*. Sequence nodes decompose into a fully-ordered set of primitive

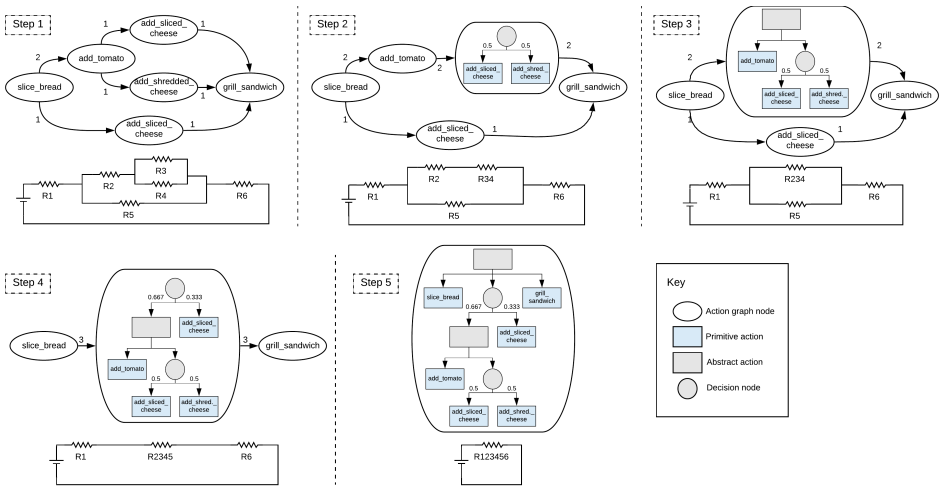


Figure 1: An action graph for making a grilled cheese sandwich, incrementally reduced into an HTN using the Naïve CircuitHTN algorithm. Analogous circuits are shown below each step.

and non-primitive actions, analogous to `AND` nodes in `AND/OR` graphs and sequence nodes in behavior trees. Decision nodes decompose into a set of task alternatives, analogous to `OR` nodes in `AND/OR` graphs and selector nodes in behavior trees. Further, we add a probability to each alternative of a decision node, denoting a preference for how often the alternative is executed.

We learn an HTN from a set of available demonstrations such that it contains a root non-primitive task and its full decomposition to primitive tasks. We assume a fully observable state space and deterministic actions, and that all demonstrations correctly perform the task. Actions can be repeated; for instance, the action add_salt may appear several times in a single cooking demonstration. We assume monotinicity of tasks, i.e. actions cannot undo the effects of other actions, and thus states are not repeated within a single demonstration. As in the works of Hayes and Scassellati [3] and Li et al. [15] we assume our demonstrations consist of grounded states and actions.

The CircuitHTN learning approach consists of two steps: *i)* converting the unstructured demonstrations to an action graph, and *ii)* converting the action graph to an HTN. The action graph is a non-hierarchical representation of all valid combinations of demonstration subsequences that complete the task. Converting an action graph to a tree-structured HTN is challenging due to its often large and interconnected structure; for example, an action graph constructed from ten demonstrations from the 50Salads dataset (see Section 4.3) consists of 138 nodes connected by 155 edges. The key insight of our work is that the problem of converting an action graph to an HTN can be made analogous to the problem of series/parallel reduction of resistor networks. In the following sections, we first describe action graph construction, and then present a naïve variant of CircuitHTN that only utilizes traditional circuit rules. We then describe a case in which naïve CircuitHTN can fail, and present an automated approach for identifying it and restructuring the action graph to again make the problem solvable through the electrical circuit analogy.

3.1 Action Graph Construction

We construct the action graph from demonstrations as shown in the `create_action_graph` function of Algorithm 1. For a set of demonstrations D, an action graph is a Weighted Directed Acyclic Graph (DAG) whose vertices are the set of state-action pairs observed in D (line 5), with directed edges connecting sequential state-action pairs (lines 6, 9). The edge weights represent frequentist probabilities according to how often action sequences occur in D (lines 7, 10). We provide a full example of an action graph for making a grilled cheese sandwich in the top-left graph in Fig. 1, constructed from the following demonstration set (states omitted for brevity):

- $[slice\_bread, add\_sliced\_cheese, grill\_sandwich]$
- $[slice\_bread, add\_tomato, add\_sliced\_cheese, grill\_sandwich]$
- $[slice\_bread, add\_tomato, add\_shredded\_cheese, grill\_sandwich]$

Algorithm 1 CircuitHTN Algorithms

```
function CREATE_ACTION_GRAPH(demos)
    G ← empty DAG
    for demo in demos do
        for s_i, a_i, s_{i+1}, a_{i+1} in demo do
            G.V ∪{(s_i, a_i), (s_{i+1}, a_{i+1})}
            if G.E ∋ ((s_i, a_i), (s_{i+1}, a_{i+1})) then
                G[s_i, a_i][s_{i+1}, a_{i+1}].weight ++
            else
                G.E ∪{((s_i, a_i), (s_{i+1}, a_{i+1}))}
                G[s_i, a_i][s_{i+1}, a_{i+1}].weight = 1
    return G

function NAÏVE_CIRCUIT_HTN(demos)
    htn_graph ← create_action_graph(demos)
    while |htn_graph.V| > 1 do
        combine_nodes_parallel(htn_graph)
        combine_nodes_series(htn_graph)
    return htn_graph.V[0]

function EXTENDED_CIRCUIT_HTN(demos)
    htn_graph ← create_action_graph(demos)
    while |htn_graph.V| > 1 do
        while is_reducible(htn_graph) do
            combine_nodes_parallel(htn_graph)
            combine_nodes_series(htn_graph)
        if |htn_graph.V| > 1 then
            restructure_graph(htn_graph)
    return htn_graph.V[0]

function RESTRUCTURE_GRAPH(G)
    candidates ← FCS(G)
    for src, sink, successors in candidates do
        if !check_valid(G, src, sink, succs) then
            candidates.remove_candidate()
    parallelize(lowest_order(candidates))

function CHECK_VALID(G, src, sink, succs)
    H ← graph(DFS(G, src, sink, succs))
    H.E \ {e | from(e) = src, to(e) ∈ succs}
    H.E \ sink.in_edges
    H.V \ src, sink
    return is_isolated(H, G \ H)

function PARALLELIZE(G, src, sink, successors)
    H ← empty DAG
    for s in successors do
        H.add_disjoint_subgraph(G, s, sink)
    G.remove_nodes(H.V)
    G.add(H)
    G.add_edges(src, successors)
```

3.2 HTN Generation

We generate an HTN from an action graph based on the process of reducing a combination circuit by finding the equivalent resistance of a set of resistors. Combination circuits are iteratively solved by combining two resistors in series or in parallel, using the sum of the resistances or the harmonic mean of the resistances, respectively. This process is repeated until a single equivalent resistor remains. Two resistors are in parallel if the following three conditions hold true:

1. The indegree and outdegree of both resistors is 1.
2. Both resistors share the same predecessor.
3. Both resistors share the same successor.

Two resistors are in series if the following three conditions hold true:

1. The outdegree of one resistor is 1. Label this resistor R1
2. The indegree of the other resistor is 1. Label this resistor R2.
3. R1's successor is R2.

An action graph can be viewed as a resistor circuit, wherein a resistor represents a graph node and a wire between two resistors represents a graph edge. A voltage source connects to the initial action resistor, and the terminal action resistor connects to a ground[2]. Any valid path along which current can flow from the power source to the ground represents a valid action sequence for solving the task.

Our implementation of the above process for HTN learning, which we call CircuitHTN, is given in the `naïve_circuit_htn` function of Algorithm 1. We reduce the action graph by combining nodes in parallel and in series when the above rules hold until a single node[3], representing the final HTN, remains. We combine two nodes in parallel with a decision node, using the action graph's edge weights to calculate a probability representing a preference for each decision node child. When combine two nodes in series with a sequence node. Fig. 1 shows a full example of iteratively constructing an HTN using CircuitHTN, with the equivalent combination circuit.

[2]Dummy initial/terminal actions can be used if the task has multiple possible start/end actions.

[3]Throughout this work, we use *node* interchangeably with the term *task* from traditional HTN literature.

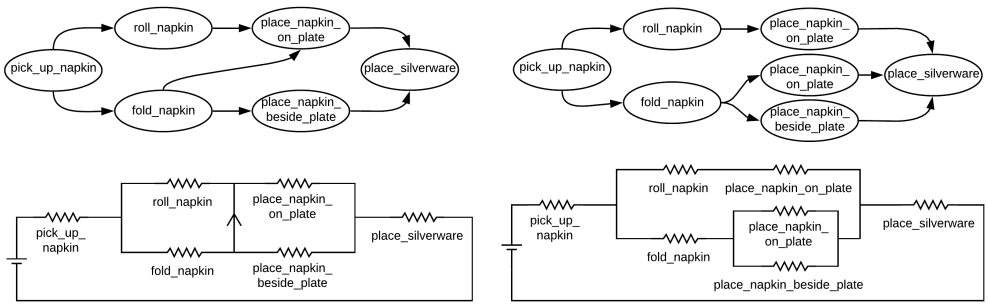


Figure 2: (Left) Example of a task graph, and its circuit representation, where no actions can be combined in series or parallel. (Right) The equivalent graph and circuit representation after restructuring with the graph `parallelize` algorithm, which can be further reduced using the series-parallel rules.

3.3 Extended Graph Reduction

A limitation of the Naïve CircuitHTN algorithm is that some demonstration sets can produce action graphs containing actions neither in series nor in parallel. This situation occurs when multiple sets of divergent action paths intersect, as shown in Fig. 2 (left) for napkin placement during a table setting task. In the resulting circuit representation, no two resistors can be combined. Further, the circuit analogy breaks down: a circuit cannot accurately represent that current must flow only from $fold\_napkin$ to $place\_napkin\_on\_plate$, and not from $roll\_napkin$ to $place\_napkin\_beside\_plate$. Under such conditions, we can restructure the graph to a reducible series-parallel structure, as shown in Fig. 2 (right).

We extend the Naïve Circuit-HTN algorithm to detect and resolve non-series-parallel graph structures, and present it in the `extended_circuit_htn` function of Algorithm 1. Whenever an action graph cannot be further reduced by the series-parallel rules, we restructure it (lines 26-27). We first detect all sets of re-converging paths in the graph using the `FCS` (first common successors) algorithm (line 31). A set of paths between two nodes a_1 and a_2 are re-converging paths if a_2 is the first common successor for the direct successors of a_1, where the first common successor of a set of k nodes $\{b_i\}_{i=1}^{k}$ is the shortest-distance node reachable from all b_i. The subgraph constructed of all simple paths from the source node, a_1, to the sink node, a_2, forms a potential candidate for restructuring.

We check the validity of each candidate (lines 32-33) as follows. A subgraph containing all simple paths between the source and sink node that pass through each direct successor (line 38, constructed using depth first traversal) is a valid candidate for restructuring if it is isolated from the remaining vertices of the action graph after removing the source and sink nodes (lines 39-41). Note that changing the structure of an isolated subgraph will not affect the rest of the action graph's structure.

With valid candidate subgraphs identified, we restructure the subgraph with the fewest vertices, using the graph `parallelize` algorithm (line 35). For each direct successor, the algorithm creates an edge-disjoint subgraph containing all paths from the successor to the sink node. This involves duplicating some nodes, but the result is a now parallel structure for each successor, and the accuracy of the task model is unchanged. We then repeat the process of series/parallel node combination, until the graph either needs additional restructuring or contains only the final HTN node.

4 Evaluation

We evaluate our CircuitHTN approach against two prior HTN learning algorithms designed to learn from unannotated data. Our experiments compare the following four techniques:

- **Clique/Chain HTN (CC-HTN)** - structure from action space topology [3]
- **Probabilistic HTN (pHTN)** - structure and preferences learned as CFG [15]
- **Naïve CircuitHTN** - action graph reduction using series and parallel combination rules
- **Extended CircuitHTN** - restructuring complex action graphs to fit the series-parallel rules

We first provide a representative example comparing the structure of the task models produced by each approach. We then report on two experiments with increasingly complex tasks: a table setting task learned from human demonstration using a simulated robot manipulator, and a salad making task learned from human observations. We evaluate each approach over four primary metrics:

- *Structure* - number of primitive tasks in the learned models

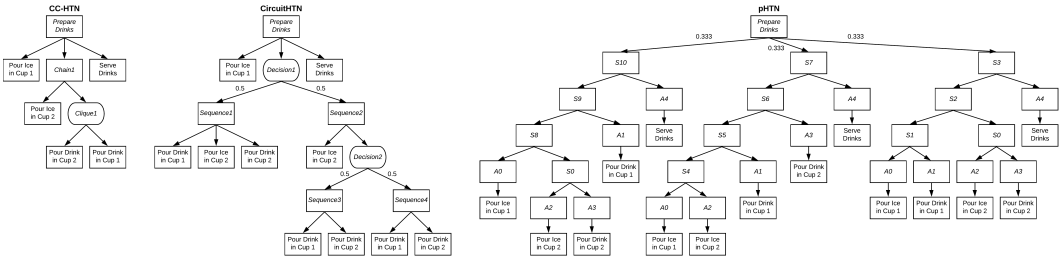
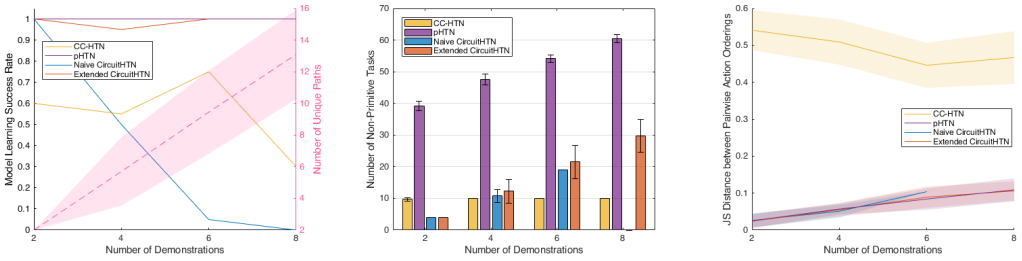


Figure 3: Example of learned task models for each unsupervised HTN learning method.



(a) Completeness with respect to increasing demonstration set size and number of unique action paths.

(b) Number of non-primitive tasks in learned HTN models (± 1 SD).

(c) Jensen-Shannon distance (± 1 SD) between sampled pairwise action ordering distributions.

Figure 4: Evaluation of completeness, structure, and preference alignment for table setting.

- *Completeness* - success rate of learning models over increasing demonstration set sizes
- *Soundness* - rate of reaching valid goal states using plans produced from learned models
- *Preference Alignment* - Jensen-Shannon distance between action preference distributions sampled from the demonstration sets and the produced plans

4.1 Structural Comparison

The structure of a learned HTN reveals how it balances expressivity with compactness to accurately represent the demonstrations while supporting efficient reasoning and human-readability. To compare the CC-HTN, pHTN, and CircuitHTN algorithms, consider a robot with three demonstrations of a drink pouring task. We select this example as it highlights differences between the algorithms, while being compact enough to present in its entirety. The robot's observations consist of the following demonstrations, in which ice must be placed into a glass before pouring a drink:

1. *pour_ice1, pour_drink1, pour_ice2, pour_drink2, serve*
2. *pour_ice1, pour_ice2, pour_drink1, pour_drink2, serve*
3. *pour_ice1, pour_ice2, pour_drink2, pour_drink1, serve*

Fig. 3 presents a visualization of the models produced by each method. The CC-HTN model (Fig. 3 left) creates a compact representation of the task, but it does not capture every demonstrated approach (e.g. demonstration 1, where each cup is filled individually), or preferences between them. Both the CircuitHTN model[4] and the pHTN model (Fig. 3 middle and right, respectively) correctly capture the full structure of the task, including probabilistic decompositions. The limitations imposed by the pHTN's grammar induction result in generating a large number of non-primitive actions, creating a large HTN model. By comparison, CircuitHTN is highly expressive, fully representing the demonstration approaches and preferences among them, while remaining compact.

4.2 Table Setting Task

In this experiment, we consider a table setting task with human demonstrations provided on a simulated robot platform. We evaluate the effectiveness of each approach at successfully learning task models that capture and reproduce human demonstrations and preferences. The task goal is to use a robot to set a table with dishware, food, and a drink. Actions available to the robot include picking

[4]In this case both Naïve CircuitHTN and Extended CircuitHTN produce the same results.

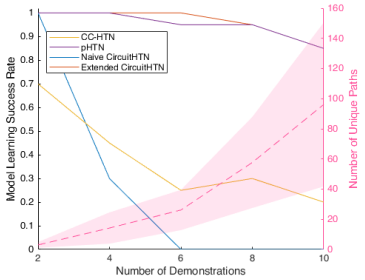
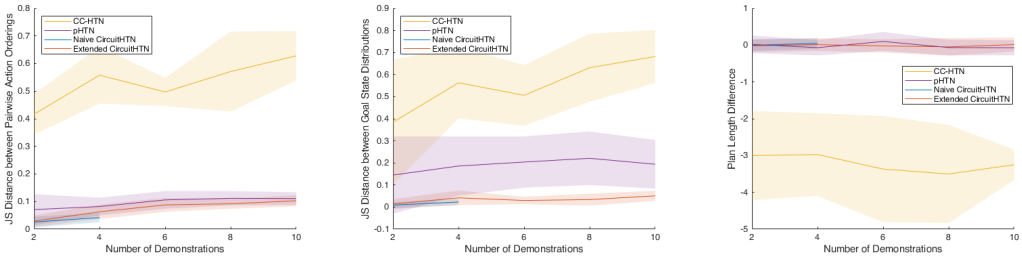


Figure 5: Salad task completeness.

Demo set size	2	4	6	8	10
Naïve CircuitHTN	1.0	1.0	n/a	n/a	n/a
Ext. CircuitHTN	1.0	1.0	1.0	1.0	1.0
CC-HTN	1.0	1.0	1.0	1.0	1.0
pHTN	0.92	0.89	0.88	0.89	0.88

Table 1: Soundness of plans produced by learned salad making HTNs.



(a) Jensen-Shannon distance between sampled pairwise action ordering distributions.

(b) Jensen-Shannon distance between goal state distributions.

(c) Difference between HTN-generated plan lengths and demonstration sequence lengths.

Figure 6: Evaluation of preference learning for the 50Salads task. All plots show mean ± 1 SD.

and placing silverware, plates, fruit, cups, ice, and a bottle of water, as well as pouring water from the bottle. The task has a single goal state, with both hard execution ordering constraints (e.g. plates must be placed before food), as well as soft execution ordering constraints that act as preferences (e.g. silverware may be placed at any time). We collected 10 demonstrations of demonstrators (university students, 7 male/3 female, 4 roboticists/6 non-roboticists) controlling a robot to set a table, using a tabletop Franka Panda arm simulated in RLBench [29]. We created 80 datasets for our evaluation, each containing a unique combination of either 2, 4, 6, or 8 of the collected demonstrations. The combinatorial nature of multiple demonstrations quickly creates large networks; for example, a set of 8 demonstrations produces an action graph with an average of 96 vertices and 104 edges.

Fig. 4 shows comparisons across the four approaches for the table setting task. As the demonstration set size increases, the action graph becomes increasingly complex. Fig. 4a shows the necessity of the graph restructuring component of the Extended CircuitHTN algorithm, as Naïve CircuitHTN quickly becomes unable to reduce the denser action graphs. Additionally, the CC-HTN algorithm fails more often with larger demonstration set sizes, as it becomes more difficult to identify clique and chain structures, while the Extended CircuitHTN and pHTN algorithms are able to learn models effectively across larger numbers of demonstrations. Fig. 4b shows that Naïve and Extended CircuitHTN learn preference-aligned models with compact structure, producing HTNs with significantly fewer non-primitive tasks than the pHTN models for all demonstration set sizes.

To evaluate preference alignment, we sample a set of 100 ground truth plans by traversing the action graph, and sample an additional 100 plans from each learned HTN. We then create preference distributions for the ground truth and learned plans by constructing a discrete frequency distribution over all pairwise orderings between actions (e.g. how many times does *place_plate* appear before *place_fork*, etc.), and measure their difference with Jensen-Shannon distance (where lower values are better). As shown in Fig. 4c, Naïve CircuitHTN (when it succeeds), Extended CircuitHTN, and pHTNs closely reproduce the demonstrated preferences, while CC-HTNs do not.

4.3 Salad Making Task

Our second task learning experiment focuses on a more complex task. We evaluate each approach's ability to learn preferences in salad making, using observations of people making salads from the 50Salads dataset [16]. Similar to table setting, this dataset represents a real-world task with a mix of both ordering constraints and un-ordered (i.e.preference-based) action sequences. The task is more

challenging than table setting for two reasons: (1) the task has multiple goal states, as the human-provided demonstrations show different types of salads according to the demonstrator's preference, and (2) demonstrations are of varying length, as actions can be partially completed and returned to according to the demonstrator's preference. Further, we note that this data was collected independently and without biases towards our learning approach.

Similar to the table setting experiment, we learn HTN models with each approach over datasets of various size. We created 100 datasets, each containing a unique combination of either 2, 4, 6, 8, or 10 of demonstrations from the 50Salad task demonstrations. Depending on the size of the demonstration set, each approach learned task models with differing success rates, as shown in Fig. 5. Performance for all of the approaches degraded as demonstration set size increased, although as with the table setting task, Extended CircuitHTN and pHTN could still construct HTNs for a majority (85%) of large demonstration sets. However, Table 1 shows a considerable difference in soundness between Extended CircuitHTNs and pHTNs. We define a sound plan as one that reaches a valid goal state. Based on the 50Salads dataset, we define a valid goal state as a salad that includes chopped lettuce, tomato, cheese, and a dressing consisting of oil, vinegar, salt, and pepper (adding cucumbers and mixing dressing into the salad are optional steps). The graph-based algorithms (CircuitHTN and CC-HTN) always produce valid salads, but the stochastic generative rules defined by the pHTN's learned grammar can generate plans that produce invalid results.

We evaluate preference alignment in the same manner as in our table setting experiment. For the more complex salad making task we also evaluate preference alignment over goal states (as Jensen-Shannon distance between goal state distributions) and action sequence lengths (as the difference between demonstration and generated plan lengths). All of these results are shown in Fig. 6.

Both Extended CircuitHTN and pHTNs again learned action order preferences that are close to the action orderings present in the demonstration sets. Extended CircuitHTN was more effective than all other methods, however, at learning preferences over multiple goal states. Fig. 6c shows that CC-HTNs consistently produce shorter paths than the demonstrations, showing that they learn efficient task policies. This is advantageous for applications that require optimal task planning, but can be detrimental for applications involving behavior modeling, where preference learning is necessary.

4.4 Summary

Extended CircuitHTN is the only approach capable of learning HTNs for large demonstration sets while also producing consistently sound plans. pHTN, as a state-agnostic method that generates action sequences from probabilistic decompositions, was the only approach to generate invalid plans. All graph-based methods produced valid task plans, as their learned hierarchical structures account for state-based pre-conditions. CC-HTN, while capable of learning correct task structure for certain cases, frequently failed to learn modles as the size of the demonstration set increased. While clique structures are a useful and compact means of representing un-ordered actions, they require many demonstrations for tasks with large sets of optional orderings. In contrast, our graph parallelization and reduction rules enable Extended CircuitHTN to account for a wider range of structures, resulting in consistent performance without requiring an exhaustive set of demonstrations.

Our results demonstrate that Extended CircuitHTN effectively and consistently learns action execution preferences that are representative of the task demonstrations. CC-HTN results in low preference alignment, indicating that demonstration information can be lost when not explicitly considering action selection preferences. In comparison to pHTN, Extended CircuitHTN learns preferences with comparable effectiveness while producing significantly smaller models, making them more relevant to human-robot collaborative tasks, which require transparent task models [19, 17].

5 Conclusions and Future Work

We introduced CircuitHTN, a circuit-inspired graph reduction approach to learning hierarchical task structure with action selection preferences from unannotated data. CircuitHTN outperforms prior HTN learning approaches both by accurately representing action execution preferences, and by successfully reducing complex task graphs produced from large and complex demonstration sets to compact models. As preference learning for hierarchical task structures has rarely been studied, we present CircuitHTN as an initial approach toward learning complete HTNs that capture realistic task execution behavior. Future work can extend our approach to recognize additional compact structures such as cliques, learn preconditions and effects, and support parameterized and stochastic actions.

Acknowledgments

This work is supported in part by Hitachi America and an Early Career Faculty grant from NASA's Space Technology Research Grants Program.

References

[1] K. Erol, J. A. Hendler, and D. S. Nau. Umcp: A sound and complete procedure for hierarchical task-network planning. In *AIPS*, volume 94, pages 249–254, 1994.

[2] T. Belker, M. Hammel, and J. Hertzberg. Learning to optimize mobile robot navigation based on htn plans. In *IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, volume 3, pages 4136–4141. IEEE, 2003.

[3] B. Hayes and B. Scassellati. Autonomously constructing hierarchical task networks for planning and human-robot collaboration. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5469–5476. IEEE, 2016.

[4] A. Mohseni-Kabir, S. Chernova, and C. Rich. Collaborative learning of hierarchical task networks from demonstration and instruction. In *AAAI Fall Symposium Series*, 2014.

[5] M. Weser, D. Off, and J. Zhang. Htn robot planning in partially observable dynamic environments. In *IEEE International Conference on Robotics and Automation*, pages 1505–1510. IEEE, 2010.

[6] L. De Silva, R. Lallement, and R. Alami. The hatp hierarchical planner: Formalisation and an initial study of its usability and practicality. In *IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 6465–6472. IEEE, 2015.

[7] Z. Kasap and N. Magnenat-Thalmann. Towards episodic memory-based long-term affective interaction with a human-like robot. In *19th International Symposium in Robot and Human Interactive Communication*, pages 452–457. IEEE, 2010.

[8] W. Wang, R. Li, Z. M. Diekel, and Y. Jia. Robot action planning by online optimization in human–robot collaborative tasks. *International Journal of Intelligent Robotics and Applications*, 2(2):161–179, 2018.

[9] D. Nau, T.-C. Au, O. Ilghami, U. Kuter, D. Wu, F. Yaman, H. Munoz-Avila, and J. W. Murdock. Applications of shop and shop2. *IEEE Intelligent Systems*, 20(2):34–41, 2005.

[10] I. Georgievski and M. Aiello. Htn planning: Overview, comparison, and beyond. *Artificial Intelligence*, 222:124–156, 2015.

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

[12] C. Hogg, H. Munoz-Avila, and U. Kuter. Htn-maker: Learning htns with minimal additional knowledge engineering required. In *AAAI*, pages 950–956, 2008.

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

[14] C. Nguyen, N. Reifsnyder, S. Gopalakrishnan, and H. Munoz-Avila. Automated learning of hierarchical task networks for controlling minecraft agents. In *IEEE Conference on Computational Intelligence and Games (CIG)*, pages 226–231. IEEE, 2017.

[15] N. Li, W. Cushing, S. Kambhampati, and S. Yoon. Learning probabilistic hierarchical task networks as probabilistic context-free grammars to capture user preferences. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(2):29, 2014.

[16] S. Stein and S. J. McKenna. Combining embedded accelerometers with computer vision for recognizing food preparation activities. In *ACM international joint conference on Pervasive and ubiquitous computing*, pages 729–738. ACM, 2013.

[17] A. Roncone, O. Mangin, and B. Scassellati. Transparent role assignment and task allocation in human robot collaboration. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1014–1021. IEEE, 2017.

[18] K. French, S. Wu, T. Pan, Z. Zhou, and O. C. Jenkins. Learning behavior trees from demonstration. In *International Conference on Robotics and Automation (ICRA)*, pages 7791–7797. IEEE, 2019.

[19] R. A. Knepper, D. Ahuja, G. Lalonde, and D. Rus. Distributed assembly with and/or graphs. In *Workshop on AI Robotics at the Int. Conf. on Intelligent Robots and Systems (IROS)*, 2014.

[20] A. Garland and N. Lesh. Learning hierarchical task models by demonstration. *Mitsubishi Electric Research Laboratory (MERL), USA*, 2003.

[21] D. Xu, S. Nair, Y. Zhu, J. Gao, A. Garg, L. Fei-Fei, and S. Savarese. Neural task programming: Learning to generalize across hierarchical tasks. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8, 2018.

[22] O. Ilghami, H. Munoz-Avila, D. S. Nau, and D. W. Aha. Learning approximate preconditions for methods in hierarchical plans. In *Proceedings of the 22nd international conference on Machine learning*, pages 337–344. ACM, 2005.

[23] C. Hogg, U. Kuter, and H. Muñoz-Avila. Learning hierarchical task networks for nondeterministic planning domains. In *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

[24] H. H. Zhuo, D. H. Hu, C. Hogg, Q. Yang, and H. Munoz-Avila. Learning htn method preconditions and action models from partial observations. In *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

[25] Q. Yang, K. Wu, and Y. Jiang. Learning action models from plan examples using weighted max-sat. *Artificial Intelligence*, 171(2-3):107–143, 2007.

[26] N. Nejati, P. Langley, and T. Konik. Learning hierarchical task networks by observation. In *Proceedings of the 23rd international conference on Machine learning*, pages 665–672. ACM, 2006.

[27] K. Tu, M. Pavlovskaia, and S.-C. Zhu. Unsupervised structure learning of stochastic and-or grammars. In *Advances in neural information processing systems*, pages 1322–1330, 2013.

[28] X. Xie, H. Liu, M. Edmonds, F. Gaol, S. Qi, Y. Zhu, B. Rothrock, and S.-C. Zhu. Unsupervised learning of hierarchical models for hand-object interactions. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9. IEEE, 2018.

[29] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison. Rlbench: The robot learning benchmark & learning environment, 2019.