IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 21, 2019, accepted January 2, 2020, date of publication January 13, 2020, date of current version January 21, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2965930

Sequential Association Rule Mining for Autonomously Extracting Hierarchical Task Structures in Reinforcement Learning

BEHZAD GHAZANFARI[1], **FATEMEH AFGHAH**[1], **AND MATTHEW E. TAYLOR**[2]
[1]School of Informatics, Computing, and Cyber Security, Northern Arizona University, Flagstaff, AZ 86011, USA
[2]School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA 99163, USA

Corresponding author: Behzad Ghazanfari (bg697@nau.edu)

ABSTRACT Reinforcement learning (RL) techniques, while often powerful, can suffer from slow learning speeds, particularly in high dimensional spaces or in environments with sparse rewards. The decomposition of tasks into a hierarchical structure holds the potential to significantly speed up learning, generalization, and transfer learning. However, the current task decomposition techniques often cannot extract hierarchical task structures without relying on high-level knowledge provided by an expert (e.g., using dynamic Bayesian networks (DBNs) in factored Markov decision processes), which is not necessarily available in autonomous systems. In this paper, we propose a novel method based on *Sequential Association Rule Mining* that can extract *Hierarchical Structure of Tasks in Reinforcement Learning* (*SARM-HSTRL*) in an autonomous manner for both Markov decision processes (MDPs) and factored MDPs. The proposed method leverages association rule mining to discover the causal and temporal relationships among states in different trajectories and extracts a task hierarchy that captures these relationships among sub-goals as termination conditions of different sub-tasks. We prove that the extracted hierarchical policy offers a hierarchically optimal policy in MDPs and factored MDPs. It should be noted that *SARM-HSTRL* extracts this hierarchical optimal policy without having dynamic Bayesian networks in scenarios with a single task trajectory and also with multiple tasks' trajectories. Furthermore, we show theoretically and empirically that the extracted hierarchical task structure is consistent with trajectories and provides the most efficient, reliable, and compact structure under appropriate assumptions. The numerical results compare the performance of the proposed *SARM-HSTRL* method with conventional HRL algorithms in terms of the accuracy in detecting the sub-goals, the validity of the extracted hierarchies, and the speed of learning in several testbeds. The key capabilities of *SARM-HSTRL* including handling multiple tasks and autonomous hierarchical task extraction can lead to the application of this HRL method in reusing, transferring, and generalization of knowledge in different domains.

INDEX TERMS Association rule mining, extracting task structure, hierarchical reinforcement learning.

I. INTRODUCTION

Reinforcement learning (RL) is known as a commonly used approach for planning and sequential decision making in artificial intelligence (AI) systems, where the agents gradually learn and optimize their actions from delayed rewards through a trial-and-error mechanism. However, one of the main challenges of RL approaches is scalability to high-dimensional or sparse rewards state spaces [1]. Hierarchical reinforcement learning (HRL) methods are known to reduce the computational complexity of RL approaches by temporal and state abstraction in the form of decomposing the learning problem to a hierarchy of several sub-problems. Sub-goals refer to the local target states that not only provide easy access or high reinforcement gradients, but also must be visited frequently [2], [3]. These sub-goals can help an agent to accelerate the learning process, particularly in high dimensional spaces. In [4], an HRL decomposition method called MAXQ is proposed based on the assumption of having an expert with the knowledge of task structures to provide a correct hierarchy, however such assumption can restrict the application of this method in

The associate editor coordinating the review of this manuscript and approving it for publication was Jenny Mahoney.

autonomous systems where a limited expert's understanding is available [5].

In the absence of an expert, several HRL techniques have been reported for task decomposition, in which a number of sub-goals that are correlated with the successful policies are utilized as the required states to decompose the learning task [2], [3], [6]. However, extracting these states in an autonomous manner is still a challenging problem to reach the expected detection accuracy or since they rely on advance knowledge [7]. One important missing piece in the majority of existing HRL methods is that the potential hidden correlations among the sub-goals to achieve the ultimate goal have been overlooked. Therefore, most of these previously reported methods cannot extract the hierarchical structure and the relation among sub-goals or temporally extended actions [8]. The key contribution of this work is to propose an HRL method based on the idea of sequential association rule mining *(SARM)* that extracts hierarchical knowledge from the hidden correlations among the extracted sub-goals and uses this knowledge to decompose the tasks into multiple sub-tasks.

The rest of this paper is organized as follows: In Section II, a comprehensive review of current HRL techniques in different domains is presented. Section III provides a review of the required concepts and notations used throughout the paper. In Section IV, the proposed SARM-HSTRL method is described, followed by multiple examples that explain the method step by step. The theoretical analysis including theorems and proofs of the properties, and the time complexity of this method are shown in Section V. The performance of the proposed method is evaluated for different simulation scenarios in Section VI. The concluding remarks are presented in Section VII.

II. RELATED WORKS

HRL methods can potentially address two common issues of RL approaches related to high-dimensional state spaces and the sparse rewards by decomposing the tasks into several sub-tasks. Thus, HRL methods provide temporal and state abstraction rather than only learning, which make them different from the methods that are based on representation learning such as deep RL methods. Furthermore, the extracted hierarchical structure can be used for transfer learning and generalization of knowledge while the optimality of solutions can be satisfied. It is worth noting that HRL and deep RL methods are different areas of research with different objectives. Deep RL techniques use function approximation methods or representation learning methods to provide practical solutions for applications where the state spaces are continuous or include images. Moreover, deep RL methods still suffer from sparse rewards issues. HRL methods have been also used in deep RL domain to address sparse reward issues and to facilitate transfer learning and generalizing of knowledge [9]–[13].

In general, HRL methods can be categorized depending on the domains that they are applied to. These categories include: 1) Markov decision processes (MDPs) for discrete state and action spaces; 2) Factored Markov decision processes (FMDPs); and 3) Continuous domains and deep RL. Since we want to distinguish between MDPs in discrete from continuous and image state spaces, we refer to them as continuous domains and deep RL. Deep RL methods are not commonly used in FMDPs since the classical RL methods in MDPs and FMDPs lead to better results and the optimal solutions can be acquired. However, there is no guarantee that the deep RL methods can provide optimal solutions. Thus, the HRL methods are often developed based on the domain characteristics and none of the existing methods can work well in all domains— i.e., the no-free-lunch theorem. For example, the methods proposed in FMDPs can extract the task structure in several levels, hierarchical forms, since FMDPs can be described in a high-level representation that makes it feasible to extract such knowledge–the details related to the performance of HRL methods in each domain are described in the following subsections.

The main contribution of our proposed method is to decompose the hierarchical structure of tasks in an autonomous manner in the most efficient way in MDPs (discrete) and FMDPs. Moreover, this method offers the following advantages compared to the state-of-the-art techniques in the field including: 1) The proposed method can be applied in both of MDPs and FMDPs, not limited to just one of them; 2) The proposed method extracts the hierarchical optimal policy without having dynamic Bayesian networks (DBNs) in factored MDPs in scenarios with a single task trajectory and also with multiple tasks' trajectories; and 3) The proposed method is able to extract the hierarchical task structures of several tasks' trajectories while the most reported techniques focus on one successful trajectory. We also prove that the extracted structure is the most efficient, reliable, and the compact hierarchical structure for both MDPs (discrete) and FMDPs.

A. HRL METHODS IN MDPS (DISCRETE STATE AND ACTION SPACES)

The HRL methods based on extracting the sub-goals [2], [3], or the ones based on bottlenecks extraction [14]–[16] can only extract a flat hierarchy, (i.e., one level) which means that these methods only find the sub-goals or the bottlenecks, rather than a hierarchical structure of them. Since these methods often use the paths or the sub-graphs of the agent, or the shortest paths among the nodes of a graph to calculate their required measures such as betweenness [16], their performance considerably degrades in scenarios with a large state space, or when the number of actions to reach the goal-states increases. They also usually require prior knowledge about the measures that helps to partition the state space to parts that are connected densely inside, but sparsely to each other [8].

Next, we provide a review of the common methods for extracting the sub-goals or bottlenecks. In summary, both types of these HRL methods are based on clustering or classification approaches to separate the sub-goals or

the bottlenecks. The frequency of visiting the states is considered as the key factor in the detection of the sub-goals in [2] and other extensions of this work including [3]. The visitation frequencies in such pure forms are prone to error and involve significant time-consuming computations for sub-goal extraction. In [6], any state is considered as a sub-goal if it has two conditions of a high frequency of state occurrence and large reinforcement gradients. Thus, this method may not show good performance in a domain with sparse rewards. We should note that these conditions are necessary but not enough [8].

A metric to detect the bottleneck states from a local perspective, called *"relative novelty"* was introduced in [15], where the novelty of the next visited states is compared to the novelty of the states which were visited before. Such methods need some characteristics of the sub-goals to be able to classify the states as sub-goals. The proposed method in [17] learns the value function by using image processing techniques. This method suffers from the same issues as the frequent-based methods and it is limited to two-dimensional spaces [8]. The graph-based methods are another common way to find the bottleneck states, i.e., the states which separate a graph into sections that are densely connected within themselves but sparsely connected to each other. These graph-based methods receive the MDP state transition graph as an input. For example, the Max-Flow/Min-Cut algorithm proposed in [18] is performed using a clustering method based on the graph density, in which the topology clustering has been used for detecting the bottlenecks to split a state-space into multiple clusters. The performance of this method depends on two functions that are defined based on the graph density: one for calculating the clustering quality and the other to indicate the separation quality between the clusters. The authors in [14] proposed an approach by using a similarity value function along with graph partitioning to extract the value bottlenecks.

As described in [8], an spectral clustering algorithm was introduced in [19] as an approximation of the normalized cut to partition the local transition graph. In [20], the method proposed in [19] was improved in an offline manner by eliminating some of its requirements in terms of advance knowledge. In [16], the authors used a measure based on the centrality measure, called as *"betweenness"* to partition the graph.

To sum up, these methods are generally based on applying different clustering approaches on the states transitions to partition the state space and extract the bottlenecks.

B. HRL METHODS IN FACTORED MDPS (FMDPS)

Some of the current HRL methods which are based on extracting the task-dependent hierarchy in FMDPs include HEX-Q [21], variable influence structure analysis (VISA) [22], and hierarchy induction via models and trajectories (HI-MAT) [23], [24]. Since there are implicit structure representations of the problems among the state variables in FMDPs, DBNs as a high-level source of pre-knowledge are often used to decompose the tasks in such processes, noting their capability to extract the impact of each action on the state variables. VISA analyzes the impacts of state variables on one another by building a causal graph using DBNs. The state variables that affect others are assigned to deeper levels in the hierarchy. HI-MAT and VISA algorithms rely on the availability of DBNs for each action [22]–[24]. Since VISA considers the impacts of all actions regardless of the domain, it can create unnecessary branches in the extracted hierarchy or unnecessary sub-tasks. Thus, it may result in an ''exponentially sized hierarchy'' that limits its application in some domains [23], [24]. To address this problem, HI-MAT was proposed to remove such unsuccessful and redundant action cycles. This method leverages a single and carefully constructed trajectory to construct a MAXQ hierarchy. It is shown that the constructed hierarchy is compact and comparable to manually engineered ones. However, the main disadvantage of both these methods is that utilizing DBNs require high-level knowledge that should be provided by an expert or needs to be extracted via a large number of computations [25]. Among these HRL methods proposed for factored MDPs, HEX-Q is the only one that does not rely on DBNs. HEX-Q extracts a task hierarchy based on an ordering of the frequency of the state variable changes. The state variables with the highest change frequency are assigned to the lowest level of the hierarchy, and the state variable with the lowest number of changes are considered as the root nodes [21]. However, this method is not capable of identifying the relations among the states variables that can potentially result in divergence of the learning process [23].

C. HRL METHODS IN CONTINUOUS DOMAINS AND DEEP RL

There are several methods based on skill discovery that have been proposed in continuous domain [26]–[31]. The authors in [28] introduced skill chaining, in which each skill leads to a designated target event. They used supervised learning to learn the initiation sets of skills as they are able to reach each other successfully. They attempted to extend their work in [27] by defining a library of abstraction for skill acquisition. In [30], the authors proposed skill learning from demonstration trajectories. In [26], the authors constructed parameterized skills of some experiences by making a generalization estimation topology of them. In [29], a segmentation approach for policies is developed to improve the scalability of the method. Since these methods rely on supervised learning, they work based on what they have been trained for, and they cannot be easily extended to larger states and actions spaces.

Deep RL has been introduced as a bridge between RL and representation learning. The majority of the methods that attempt to use hierarchical structures in deep RL can only work in applications in which the states are images. Some of them can also work in classical RL problems such as [32]–[34], but still similar to [35]–[44], these methods suffer from the inability to extract hierarchical structures in

several layers as the HRL methods that have been proposed for FMDPs [22]–[24]. In [11], the authors used the values of observed states to train function approximations in order to estimate the values of unseen states for different goals. This method can be used for defining options. However, they assumed the goals are given in advance which is not a practical assumption for real tasks in RL. In [12], the authors used some predefined sub-goals in multi-task and transfer learning in deep RL.

It should be noted that some HRL approaches have been applied in deep RL techniques [9]–[12], [35]–[45] which consider some level of in advance knowledge or internal rewards to extract the temporally extended actions, although these methods are considerably different from the ones utilized in FMDPs. In other words, the HRL methods that have been applied in each of these domains are essentially different from one another and there are not any scalable methods that work in all of these domains. A few methods including [32]–[34] can work in MDPs and deep RL groups. They define another learning or approximation parts that possibly conflict with the goal reward of the original problem in RL. These methods cannot scale up in continuous or image state spaces, cannot be applied in FMDPs, and also they do not extract the hierarchical structure of tasks.

In summary, conventional sub-goal extraction methods that can work in MDPs, do not extract a hierarchical task structure. The few existing hierarchical structure extractor methods in RL including HEX-Q, HI-MAT, and VISA only work in FMDPs. More importantly, HI-MAT and VISA rely on DBNs knowledge, which is a high-level supplementary knowledge provided by human experts. We should note that HI-MAT, the most recent HRL approach in the literature, cannot be applied in MPDs, and it can only handle one trajectory while the need to work with several trajectories is required in several RL applications. However, our proposed *SARM-HSTRL* method extracts a hierarchical optimum policy of the task structure for both MDPs and FMDPs, while it does not rely on DBNs as a pre-knowledge structure provided by human experts in FMDPs. *SARM-HSTRL* is also the first method that can extract the hierarchical optimum policy of the task structure with multiple policies.

III. BACKGROUND

In this section, we explain the concepts of MDPs and FMDPs as the two domains that the proposed method is applied to. Then, we provide a brief introduction to association rule mining.

A. MDPS AND FMDPS

RL tasks are typically defined in a MDP framework as a $5-tuple$: $\langle S, A, P, R, \gamma \rangle$. In this paper, we focus on finite MDPs, where $S = \{s_1, \ldots, s_n\}$ is a finite set of states, $A = \{a_1, \ldots, a_m\}$ is a finite set of primitive actions, $P : S \times A \times S \to [0, 1]$ is a one-step probabilistic state transition function, $R : S \times A \to \mathbb{R}$ is a reward function, and $\gamma \in (0, 1]$ denotes the discount rate. The agent's goal is to find a policy (a mapping from the states to the actions), $\Pi : S \times A \to [0, 1]$ that maximizes return as the accumulated discounted reward $R = \sum_{i=0}^{T} \gamma^i r_i$, for each state in S.

FMDPs are known as an extension of MDPs that contain a structured representation of problems, where T and R are represented in a compact way. In factored MDPs, the states are described by a set of state variables. To have a unified definition for both MDPs and FMDPs, each state in an MDP can be described by a random variable X which contains one variable $X_1, X = (X_1)$, that takes different values. In FMDPs, X is a multivariate random variable, $X = (X_1, X_2, \ldots, X_n)$. Each state x is an instantiation of X, and it can be shown as a vector of $(x_1, x_2, \ldots, x_n)$ such that $\forall i\, x_i \in Dom(X_i)$, in which $DOM(X) = \langle D_1, D_2, \ldots, D_n \rangle$ refers to the set of possible values for X as a multivariate variable [46].

The value of a state s based on a policy π is defined as the expected return that is gained by following π of the state s. There is always at least one policy that its expected return is equal or greater than any other policies for all states [47] that such policy or policies are known as optimal policies and denoted π^*. Hence, the corresponding state-value function, V, and the action-value function, Q, are optimal and shown as follows: $V^*(s) = \max_\pi V_\pi(s)$ for all $s \in S$, and $Q^*(s, a) = \max_\pi Q_\pi(s, a)$ for all $s \in S$ and for $a \in A(S)$, respectively.

B. AN OVERVIEW ON ASSOCIATION RULE MINING

Association rule mining (ARM) methods use a combination of two key measures of *support* and *confidence* in a proven effective extraction strategy to obtain and evaluate the most efficient and reliable relationships among the variables in a dataset. ARM has been applied in bioinformatics to discover the patterns in datasets that are statistically important [48], or in retail stores to find the items that are commonly being sold together among millions of transactions [49], [50].

An ARM problem is defined by a pair of $\langle$ *ITEMSET*, *Transaction* $\rangle$, where $ITEMSET = \{i_1, \ldots, i_g\}$ is the set of all items and $Transaction = \{\Omega_1, \ldots, \Omega_N\}$ is the set of all transactions. Each transaction is a subset of items of *ITEMSET*. The relationship among the items in the transaction set can be defined by an *association rule*. An association rule is expressed in the form of $A \to B$, where A and B are disjoint sets of items; $A \cap B = \emptyset$. The frequency of the occurrence of A and B together in a *Transaction* is defined as a *key factor*, also known as *support* of the association rule. The frequency of occurrence of A and B, relative to the frequency of the occurrence of A, is known as *confidence*. The definition of support and confidence are as follows [50]:

$$support(A \to B) = \frac{\sigma(A \cup B)}{N}$$

$$confidence(A \to B) = \frac{\sigma(A \cup B)}{\sigma(A)}$$

where $\sigma(\cdot)$ is the number of observed transactions including the elements inside of the parenthesis, and N is the total number of transactions. The support factor is often used as a measure to disregard the items that do not occur together so

frequently relative to N, and confidence can express the reliability of the extracted rules. The corresponding thresholds for support and confidence, known as *minsup* and *minconf*, respectively, can be used to extract the important rules [50].

ARM algorithms typically consist of two parts: 1) Frequent Itemset Generation: All of the itemsets that satisfy the *minsup* condition are extracted, i.e., frequent item sets. 2) Rule Generation: Building upon the outputs of the Frequent Itemset Generation, this step calculates the confidence of the obtained frequent itemsets and checks their eligibility by comparing their confidences with the minconf threshold. The frequent pattern growth (FP-growth) algorithm has been proposed for Frequent Itemset Generation by constructing a compact data structure, called a FP-tree. The confidence value is calculated for each rule and evaluated based on minconf. The FP-tree is constructed by reading the transactions one by one and mapping them onto a path of the FP-tree in which FP-tree uses the common item sets to provide a compressed data structure. FP-Growth is an efficient approach that uses FP-tree and extracts frequent item sets directly from that [51]. This algorithm outperforms the majority of frequent patterns extraction algorithms in large datasets; for the analysis of time complexity and more details about FP-growth algorithm see [50]–[52].

IV. PROPOSED SARM-HSTRL ALGORITHM

In this section, we describe the proposed algorithm to extract a hierarchical structure of tasks in RL named *SARM-HSTRL* for both MDPs and FMDPs. Despite the existing HRL methods in MDPs, *SARM-HSTRL* extracts a hierarchical abstraction, not a flat abstraction in MDPs. Also, it works in FMDPs without having advance knowledge like DBNs. In both of these domains, it is proven theoretically that the proposed method extracts hierarchical optimal policies. We also show that the extracted hierarchical task structures are consistent with the trajectories and can provide the most efficient, reliable, and compact structure under appropriate assumptions. The experimental results confirm the ability of this method to find the hierarchical optimal policies when dealing with multiple trajectories. Also, the performance of this method is compared to several known methods in the literature via multiple experiments, while the proposed method does not rely on any pre-knowledge input. In the next subsections, we define the terms and concepts used throughout the paper, followed by a detailed description of this method through multiple examples.

A. DEFINITIONS

Definition 1: To assign a unique representation to a set of multiple state variables in FMDPs, here we define a reversible encoder-decoder operation. A map function, MF, is defined as an encoder which maps the state variables in FMDPs to one variable. MF^{-1} denoted the the decoder which performs the reverse process of retrieving the FMDPs' state variables from just that single value, L, as described in Algorithm 1.

Algorithm 1 $MF^{-1}(L)$

Input: L as a MDPs' state value that is mapped from the FMDPs state value R, n is the number of variables in the FMDPs' state space.
Output: R as the corresponding state value in the FMDPs' state space.
for $i = n : 2$ **do**
  $TEMP = \prod_{j=1}^{i-1} numD_j$;
  $R_{x_i} = L/TEMP$;
  $L = mod(L, TEMP)$;
  if $L == 0$ **then**
    $R_{x_i} = R_{x_i} - 1$;
    $L = TEMP$;
  end if
end for
$R_{x_1} = L$

MF is defined as

$$MF(x_1, x_2, \ldots, x_n) = R_{x_1} + \sum_{i=2}^{n} R_{x_i} \prod_{j=1}^{i-1} numD_j$$

and should be a surjective, injective, and invertible function, where D_i refers to the set of possible values for each state variable, $numD_i$ denotes the number of possible values in D_i, and R_{x_i} shows the index of x_i in $numD_i$. Therefore, $\prod_{i=1}^{n} numD_i$ is the number of total possible values for X.

Definition 2: Transitions are considered *unpredictable* when they lead to entering or leaving the sub-goals. The boundaries among the states' clusters which have unpredictable transitions are considered as *exits* and defined by a state-action pair $G_i = (s^{T_i}, a)$ when taking action a, as a primitive action, from state s^{T_i}, as a sub-goal, leads to the resultant state that is a goal state to complete sub-task T_i [21]. A region is a set of states which are reachable from each other such that any exit states in a region can be reached from other states with probability 1 [21]. This concept has been further explained with an example in Section IV-C— "An Example of SARM-HSTRL".

Definition 3: A task hierarchy H is generally shown as a tree, or a directed acyclic graph, $\langle T, E \rangle$, in which the root as the main task, T_0, is decomposed to other sub-tasks $T_1, \ldots, T_n$ and the edges, E represent the relation among them. A sub-task, T_i, is a semi-MDP (SMDP), shown by $\langle X_i, S_i, G_i, C_i, \rangle$ [24], where X_i is the set of variables that their corresponding values change during performing the sub-task, S_i denotes the set of admissible states of T_i, G_i shows the exits of corresponding sub-tasks as termination conditions of T_i, and C_i is the set of child tasks of T_i. Child tasks, C_i, can be formed based on different HRL frameworks such as MAXQ or option.

In the task hierarchy graph, leaf nodes correspond to sub-tasks that interact with the environment directly by applying primitive actions, A, to states, S_i. Other nodes of the

Algorithm 2 *SARM − HSTRL*

Input: Transaction, *minsup*, *minconf*.
Output: *HST*.
1. SARM: Sequential association rule mining
1.1) Frequent Itemset = FP-growth (Transaction, *minsup*);
1.2) Association Rules = Rule Generation (Frequent Itemset, *minconf*);
2. HST: HST-construction (Association Rules) // See Algorithm 3.

task hierarchy include sub-tasks as abstract states and their corresponding local policies as abstract actions. We should note that the sub-tasks are defined over the extracted regions as the policies that lead to leaving these regions via *exits*. These definitions guarantee that no action can lead to leaving a sub-task except via its *exits*. Each sub-task similar to a region includes a set of states, actions, Markov transitions, and reward functions.

B. ALGORITHM OF SARM-HSTRL

The proposed *SARM-HSTRL* algorithm decomposes the tasks into multiple sub-tasks by extracting the sub-goals as the sub-tasks' termination or *exits*. Sub-goals are the nodes of the task graph and the relations of the tasks, in the form of association rules, are the graph's edges. The state space is partitioned recursively in a top-down manner, and the state abstraction and the corresponding sub-tasks (i.e., temporal abstraction) are formed for these partitions. The state abstraction and the temporal abstraction can limit the policy search space that leads to increasing the speed of learning. In other words, each sub-task, T_i, in which i shows the index of sub-task, is defined based on the sub-goal states. These sub-goals, as the exit states, are defined as the states that are frequently visited in successful trajectories (i.e., the trajectories where the agent reaches a goal state). In other words, the problem of finding the sub-goals and the relations among them can be seen as extracting association rules, $A \rightarrow B$ in which $|A| >= 1$ and $|B| = 1$, such as $\{s_d, \ldots, s_g \rightarrow s_h\}$, where $\{s_d, \ldots, s_g, s_h\}$ are the sub-goal states. We consider $|B| = 1$ in the association rule in the second step of SARM to have similar forms of the extracted association rules to construct the structure in HST. It should be noted that there often exists a set of some key sub-goals that are common among different tasks, and the proposed *SARM-HSTRL* method can extract such key sub-goals by processing a set of trajectories of tasks with random start and goal states.

The proposed *SARM-HSTRL* is composed of two phases (see Algorithm 2). In the first phase, several association rules are extracted using an *SARM* approach following the two steps of i) Frequent Itemsets Generation, and ii) Rule Generation procedure. Then, the proposed *HST-construction* method converts these association rules to a hierarchical structure tree.

Input: *Transaction* alongside with two parameters of *minsup* and *minconf* are the inputs given to *SARM-HSTRL*. Transaction is a set of successful trajectories. A successful trajectory is defined as a trajectory of states that leads to the goal reward [23]. Each trajectory of visited states, $\Omega_k = \{s_1, \ldots, s_h\}$, is considered as a transaction member of the *Transaction* set, in which h shows the number of states in that transaction. In FMDPs, the proposed function *MF* (see Definition 1) is used to map the multivariate state variables to a univariate state variable. All visited states in successful trajectories are stored in the *ITEMSET*. Since we apply SARM on the states, we considered the transactions based on the trajectories of states, while the corresponding actions of these trajectories of states are used in the final step of HST to define the exit pairs and the task hierarchy.

The algorithm also takes two parameters of *minsup* and *minconf* as input. If the *minsup* is set to its maximum possible value (i.e., one), the sub-goals must be visited in each trajectory of each transaction. If we set a very small value to the *minsup*, the performance of FP-growth will be degraded as *SARM-HSTRL* may provide some false-positive itemsets for the evaluation of Rule Generation. Hence, we face a trade-off in selecting reasonable values for *minsup* and *minconf*. On one hand, these values should be small enough to capture different sub-goals and relations in RL domains with multiple types of successful trajectories. On the other hand, if the *minsup* and *minconf* are set to very small values, the extracted hierarchical structure would extract some unnecessary sub-goals and relations. Thus, the proper range of these parameters can be set based on the number of trajectories of encountered tasks.

An interesting fact about this method is that the parameters *minsup* and *minconf* are defined based on the features that make some states sub-goals. There is a significant margin between the values of support and confidence of the regular states and the sub-goal states. As a result, we can perform a simple search starting from the states with larger values of support and confidence to the ones with smaller values to ensure that all the notable states are captured. In fact, high support and confidence of some states relative to other ones make these states as sub-goal states. The range of these values can vary based on the application domains, but there is still a considerable margin between these numbers for a typical state and a sub-goal.

SARM: The input of SARM are *Transactions*, trajectories of states, and the two thresholds of *minsup* and *minconf* which are used in its two steps to extract some states of the *Transactions* as sub-goals in form of association rules. In the first step, the frequent itemset- the biggest groups of states which are visited together in the *Transaction* as their support are bigger than *minsup*— are nominated for further processing to the second step of SARM. In the second step of SARM, the association rules, in form of $A \rightarrow B$ in which $|A| >= 1$ and $|B| = 1$ are extracted as $\{s_d, \ldots, s_g \rightarrow s_h\}$, where $\{s_d, \ldots, s_g, s_h\}$ are the sub-goal states. It should be noted that the sequence of states as the trajectory of states

without the corresponding actions are given as the input to SARM to find the sub-goals. The corresponding actions are used later in the final step of HST to define the exits.

Here, sequential association rule mining is applied in two consecutive steps: 1) Frequent Itemset, and 2) Association Rules.

1.1) Frequent Itemset: First, we use the FP-growth algorithm to perform Frequent Itemset Generation. The transaction set is defined by $Transactions = \{\Omega_1, \ldots, \Omega_N\}$ in which N shows the number of input transactions and a transaction $\Omega_k = \{s_1, \ldots, s_n\}$ is a successful trajectory. Also, *ITEMSET* is equivalent with S, $S = \{s_1, \ldots, s_n\}$ in which n denotes the size of state space. In this step, the *support* factor of different possible sets of states through the *Transaction* set is calculated by FP-growth algorithm. This algorithm can extract different sets of states which have been observed together in members of *Transaction* set bigger than *minsup* in an efficient way. In other words, the Frequent Itemset Generation algorithm searches for the largest sets of states which their occurrence frequency is larger than minsup.

Figure 1 shows the performance of a simple pruning approach in an example. Let us consider four states of s_1, s_2, s_3 and s_4 with the corresponding support values of 0.6, 0.8, 0.5, and 0.9, respectively. In a naive tree search, there are $2^4-1 = 15$ possibilities to be evaluated. When using this pruning method with *minsup* of 0.75, since the two states of s_1 and s_3 are visited in trajectories with a frequency less than *minsup*, all of the further nodes that include these states do not to be considered. As there are only 5 nodes that should be extended, hence this method offers 75% less evaluation without losing optimally. Let us assume that the support value of s_2s_4 is 0.78, then s_2s_4 is recognized as a frequent itemset. In fact, s_2s_4, in form of the association rule, $s_2 \rightarrow s_4$, is the largest set of states that satisfies the *minsup*.

1.2) Association Rules: Next, the Rule Generation procedure is performed on the extracted frequent itemsets as the output of FP-growth algorithm. The objective is to find the states that there is a meaningful temporal relation among them and to provide those states and their relations in the form of rules. In other words, different temporal relations among the qualified states of frequent itemsets' levels are evaluated by comparing the *minconf* as the threshold with the confidence of those frequent itemsets and representing the qualified ones in the form of association rules. Recall that a confidence value is the conditional probability of occurrence of a consequent of a certain rule when its premise has been seen, and is calculated using the *minconf* threshold [1].

For the example described in Figure 1, the permutations of s_2 and s_4 in form of $s_2 \rightarrow s_4$ or $s_4 \rightarrow s_2$ are the only inputs for rule generation. In the rule generation step, the confidence of the association rules $s_2 \rightarrow s_4$ and $s_4 \rightarrow s_2$ are compared with the *minconf*. If their confidence is above the *minconf*, they are considered as the tasks to be constructed from the two sub-goal states of s_2 and s_4.

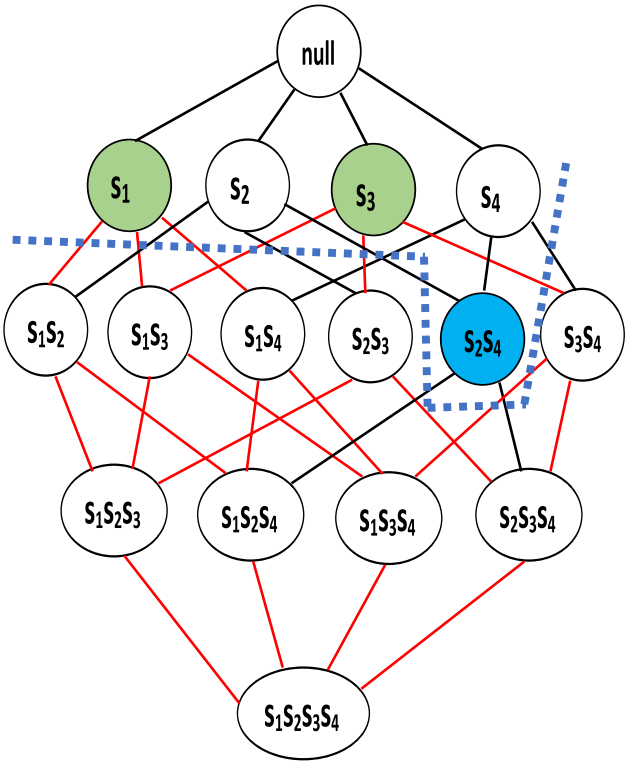


FIGURE 1. An example of a tree search for 4 states. If the support values of s_1 and s_3 are smaller than *minsup*, possible combinations of them are not evaluated– meaning that only the states above the dashed blue line are evaluated. The red lines show the nodes which are not evaluated. s_2s_4 is considered as a frequent itemset. FP-growth algorithm in a practical way based on the FP-tree data structure provides such pruning to perform the evaluation without missing any proper possibilities.

Hierarchical Structure of Tasks (HST): Besides extracting a set of sub-goals as the association rules, *SARM-HSTRL* also extracts different possible sequences of these sub-goals for HST construction in a sequential association rule mining procedure. The value of t, time of each sub-goal in each trajectory, can be compared to create a sequence of observed sub-goals.

Each sequence shows the relationship among the sub-goals in a flat manner of one association rule. Let us consider an example with four sub-goals denoted by a, b, c, and d and two trajectories of $a, b, c \rightarrow d$ and $b, a, c \rightarrow d$, where the t's values of a and b in the trajectories are $\{1, 2\}$ and $\{2, 1\}$, respectively. If the frequencies of those orders were the same, it means that the order of visiting a and b is not important to achieve the consequence sub-goal although each sequence could have a different probability value.

Algorithm 3 describes the HST-construction method which generates the hierarchical structure of tasks. HST helps an agent to choose the correct sub-tasks. Each association rule AR_i can be shown in the form of $AR_i = s_{ti}, \ldots, s_{(t+n)i} \rightarrow s_{(t+n+1)i}$, where $\{s_{ti}, \ldots, s_{(t+n)i}\}$ denotes a sequence of sub-goals of the AR_i. In this algorithm, Len_i denotes the number

[1]The confidence value of each association rule can be used as a priority score to select from the corresponding temporally extended actions of association rules.

Algorithm 3 HST-construction

1: **Input**: *AR-set* is the set of association rules. *AR-set* = $\{AR_1, \ldots, AR_{NumRules}\}$.
2: **Output**: Constructing a tree, T, with one node that is the root node, R.
3: *num* : the number of children of the Parent-Node.
4: PN_t : the t_{th} child of the Parent-Node.
5: **for** $i = 1 : NumRules$ **do**
6: Parent-Node=R;
7: **for** $j = Len_i : 1$ **do**
8: $t = 1$;
9: $FlagM = 0$;
10: **repeat**
11: **if** $AR_{ij} == PN_t$ **then**
12: Parent-Node=PN_t;
13: $FlagM = 1$;
14: **end if**
15: $t = t + 1$;
16: **until** $t <= num$ and $FlagM == 0$
17: **if** $FlagM == 0$ **then**
18: A new child Node in the Parent-Node is created: $PN_{num+1} = AR_{ij}$;
19: Parent-Node = PN_{num+1};
20: **end if**
21: **end for**
22: **end for**

of items in AR_i. The number of elements of the premise of the AR_i is $n+1$, and the number of elements of the consequence of each AR is 1; thus, the Len_i is $n+2$. $AR_{i,j}$ is the jth element from the end of $AR_{i,j}$. For example, $AR_{i,2}$ is s_{t+n} and AR_{i,len_i} is s_t. *NumRules* is the number of association rules. *"Num"* is defined in line 3 as the counter or the number of children of the parent node which changes as the nodes are added to the parent node. The parent node refers to the current node to which the elements are being added.

As mentioned earlier, HST is a tree of sub-goal states which shows the relation among them in a hierarchical manner. Now, the task hierarchy H as a tree of sub-tasks $T_1, \ldots, T_n$ is built based on HST in which each T_i corresponds to one of the sub-goals. Thus, n shows the number of sub-tasks which is the same as the number of sub-goals.

S_i of T_i, the set of admissible states of T_i, is defined as a set of states that the i^{th} sub-goal, s^{T_i}, is reachable from them without passing other sub-goals. The corresponding exit of the sub-task T_i, $G_i = (s^{T_i}, a)$, is defined as pair of a state (a sub-goal) and an action that makes the sub-task completed. In other words, after choosing the action, a, the agent leaves the sub-goal, s^{T_i} and the corresponding state space of the sub-task, S_i of the T_i. After extracting the sub-goals states and their relations, the trajectories are partitioned based on these sub-goals to find the corresponding states that lead to the sub-goals.

C. AN EXAMPLE OF SARM-HSTRL

In this section, we provide a detailed example to describe the proposed *SARM-HSTRL* on a testbed described in Figure 2. To define an association rule, a pair of $\langle$ *ITEMSET*, *Transaction* $\rangle$ needs to be defined. *ITEMSET* is equivalent with S, $S = \{s_1, \ldots, s_n\}$ in which n denotes the size of state space. Therefore, $ITEMSET = \{s_1, \ldots, s_{60}\}$ in the following example. $Transaction = \{\Omega_1, \ldots, \Omega_N\}$ is the set of all transactions. As mentioned earlier, each transaction is defined as a successful trajectory of states from a start state to a goal state. Since the start states and goal states are chosen randomly, the first elements as the start states and the last elements as the goal states of these trajectories can be different.

Let us define an experiment to describe the different steps and terms of the *SARM-HSTRL* in the following state space depicted in Figure 2. In this experiment, there are 3 phases in the system and the agent has five primitive actions: *up, right, down, left, and enter*. The goal states are in the third phase. The agent starts from the first phase and can move in the second phase if the agent enters state s_7 and takes the action *enter*. The third phase activates if the agent is in the second phase and enters s_{34} and executes the action *enter*. There are 60 states, where the first four actions are the movement actions (i.e., *up, right, down, left*) and the action *enter* can take the agent to the next phase. The agent starts from a random place and should pass through states s_{27} and s_{54} to go to the goal states, which are selected randomly, in order to receive the goal reward. There is a positive reward to reach the goal state by passing these phases in the right order and a negative smaller reward for taking each action. If we run the Q-learning method for the agent on this maze for different start and goal states, it learns policies gradually during different episodes. An episode is a trajectory of the sequence of states and actions and it leads to a goal reward if it reaches the goal state that is in the third phase. Clearly, many episodes in the beginning of the run will not lead to the goal reward. But, Q-learning gradually learns the policies to reach the goal state that goes through s_{27} and s_{54}.

1) ITEMSET AND TRANSACTION

We consider three runs, each run corresponds to a different start and goal state, and 200 episodes of learning for each run. Among 200 episodes of each run, we select the two that have the largest accumulated rewards as follows:
The first run: the start state is s_1 and the goal state is s_{57}.

$$\Omega_1 = \{s_1, s_2, s_3, s_7, s_{27}, s_{31}, s_{35}, s_{34}, s_{54}, s_{58}, s_{57}\}.$$
$$\Omega_2 = \{s_1, s_5, s_6, s_7, s_{27}, s_{31}, s_{30}, s_{34}, s_{54}, s_{53}, s_{57}\}.$$

The second run: the start state is s_3 and the goal state is s_{59}.

$$\Omega_3 = \{s_3, s_7, s_{27}, s_{26}, s_{30}, s_{34}, s_{54}, s_{55}, s_{56}, s_{60}, s_{59}\}.$$
$$\Omega_4 = \{s_3, s_7, s_{27}, s_{31}, s_{35}, s_{34}, s_{54}, s_{58}, s_{59}\}.$$

The third run: the start state is s_{12} and the goal state is s_{59}.

$$\Omega_5 = \{s_{12}, s_8, s_7, s_{27}, s_{26}, s_{30}, s_{34}, s_{54}, s_{58}, s_{59}\}.$$

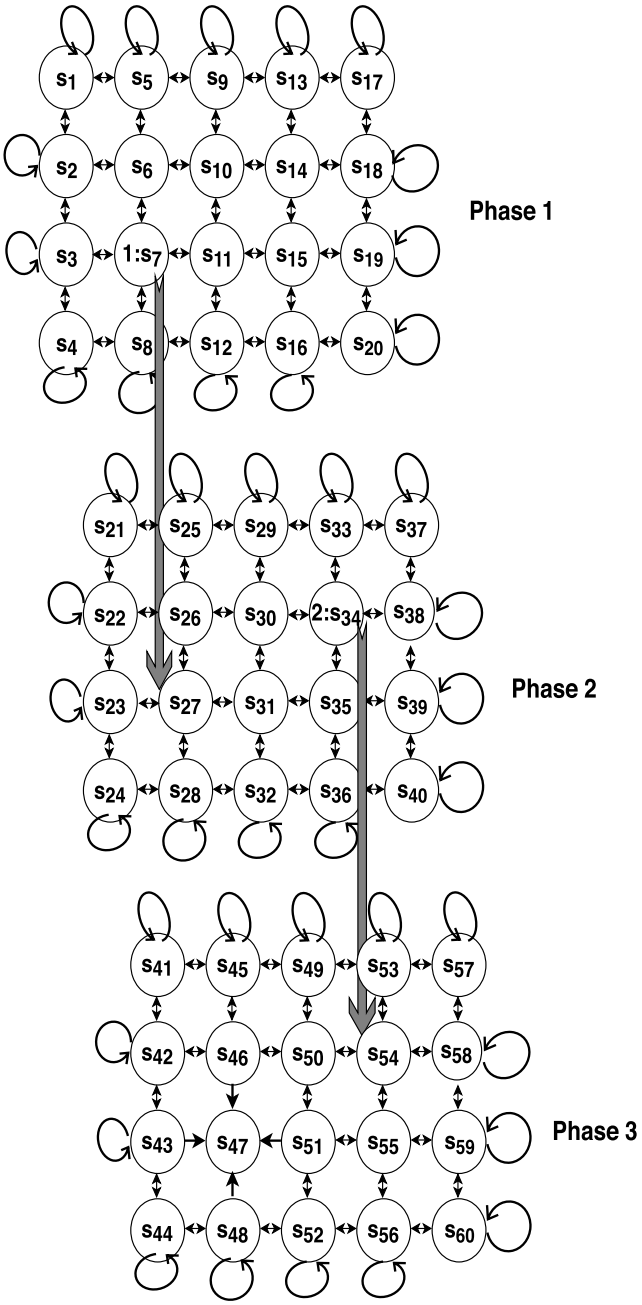


FIGURE 2. The testbed for the example described in Section IV-C. The size of the testbed is 4 × 5 × 3 = 60 states.

$$\Omega_6 = \{s_{12}, s_{11}, s_7, s_{27}, s_{26}, s_{30}, s_{31}, s_{35}, s_{34}, s_{54}, s_{58}, s_{59}\}.$$

In the above example, $ITEMSET = \{s_1, \ldots, s_{60}\}$, $Transaction = \{\Omega_1, \ldots, \Omega_6\}$, and the number of transactions is 6 ($N = 6$). All of these transactions lead to the goal states of their runs, it means they are successful trajectories.

2) FREQUENT ITEMSET

To calculate the association rules in the *ITEMSET*, we need to calculate the support for each possibility of extracted association rules in the form of $A \rightarrow B$ by performed by using FP-growth algorithm, in which $|A| >= 1$ and $|B| = 1$ like $\{s_d, \ldots, s_g \rightarrow s_h\}$ and $\{s_d, \ldots, s_g, s_h\}$ are the sub-goal states. As defined in Section III-B, the support factor is calculated as follows:

$$support(A \rightarrow B) = \frac{\sigma(A \cup B)}{N}$$

The FP-growth algorithm checks all the possibilities of the state sequences without missing the most important ones by pruning the combination of elements that their support value or the support value of their children in the search-graph is smaller than *minsup* based on the FP-tree. Hence, by using a pruning approach based on the support values, we did not proceed all the combinations of nodes that are composed of s_1 or s_3 in the Figure 1 since the support of s_1 and s_3 were smaller than *minsup*.

Now, we describe the process of calculating the support measure in several examples. For instance, let us consider $A = s_1$ and $B = s_{58}$. Since these two states (i.e., s_1 and s_{58}) are only simultaneously observed in Ω_1, then $\sigma\ (s_1 \cup s_{58}) = 1$. Thus, $support\ (s_1 \rightarrow s_{58}) = \frac{1}{6}$. As another example, if we consider $A = s_7$ and $B = s_{34}$, since s_7 and s_{34} are visited in all $\{\Omega_1, \ldots, \Omega_6\}$; thus, $\sigma\ (s_7 \cup s_{34}) = 6$ and $support\ (s_7 \rightarrow s_{34}) = \frac{6}{6}$. The support values of all the combinations should be calculated while the further appearance of those combinations is not proceeded if their support factor is less than *minsup*. If the *minsup* value is set to 0.9, by a simple pruning approach and calculating the support of the units with one element, single states, there are just 4 states, (i.e., s_7, s_{27}, s_{34}, s_{54}) for which their support values (i.e., 1) is larger than 0.9. Thus, a simple pruning approach results in not proceeding the other nodes. Now, the support value of dual combinations of these states are checked (i.e., s_7s_{27} or s_7s_{54} or $s_{54}s_{34}$). If their support values are calculated, it can be seen that the support of all of these dual combinations are 1 (larger than 0.9). Next, we calculate the support values of three-state combinations, it can be seen all of trinary combinations satisfy the *minsup*. Finally, the quadric combinations, which are based on permutations of $s_7s_{27}s_{34}s_{53}$, are checked. These support values are 1, and therefore they are nominated as the frequent Itemsets for the second level of SARM.

Association Rules: As defined in Section III-B, the confidence value of an association rule is calculated as follows,

$$confidence(A \rightarrow B) = \frac{\sigma(A \cup B)}{\sigma(A)}$$

In this paper, we use the sequential ARM (*SARM*) technique instead of a conventional ARM technique due to its capability to consider the order of items in addition to their occurrence frequency. If we consider $minconf = 0.9$ then $(s_7, s_{27}, s_{34} \rightarrow s_{54})$ satisfies the constraint since the confidence of $(s_7, s_{27}, s_{34} \rightarrow s_{54})$ is 1. In this case, the other orders have the confidence of 0 (e.g., $confidence\ (s_{54}, s_7, s_{34} \rightarrow s_{27}) = 0$.)

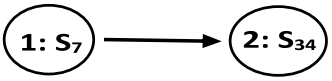


FIGURE 3. The extracted task hierarchy for Figure 2. An example of a sequential association rule mining and HST-construction of one rule.

3) HST

The states of s_7, s_{27}, s_{34} and s_{54} are given to HST as its input. Since there are not any states between s_7 and s_{27}, we just consider the first element, s_7, as a sub-goal for HST and disregard s_{27}. Similarly, since there are not any states between s_{34} and s_{54}, we just consider s_{34} which is the first element among them for the *HST*. Therefore, we just have the association rule $s_7 \rightarrow s_{34}$ which is constituted of two sub-goals of s_7 and s_{34}, and the HST constructed based on them in order as shown in Figure 3. In this example, there exists only one rule ($i = 1$) in which the length of the rule is $2 - Len_1 == 2$. The Parent-Node refers to the root with an empty child, since the length of the rule is 2 ($Len_1 == 2$), thus, j is initialized to 2, and AR_{12} is s_{34} which is compared in line 11 of Algorithm 3 if the Parent-Node has at least a child. Since the Parent-Node is an empty child, hence, a node, $2 : s_{34}$, is created for that in line 18 of Algorithm 3 and the Parent-Node refers to its child that is empty. Now, AR_{11} is s_7 that is added in form of $1 : s_7$ as the child of $2 : s_{34}$.

Now, we define the sub-tasks T_0 and T_1 for s_7 and s_{34} correspondingly. The state space is partitioned based on s_7 and s_{34} as the states which have been observed before each of them without passing other sub-goals. The state space are recognized as the $S_1 = \langle s_1, \ldots, s_{20} \rangle$ and $S_2 = \langle s_{21}, \ldots, s_{40} \rangle$. Since s_7 and s_{27} are consecutive by one action, *enter*, the exit is defined as (s_7, *enter*) in which s_{27} does not belong to S_1. In the same way for s_{34} and s_{54}, (s_{34}, *enter*) is considered as the exit.

The exits $G_0 = (s_7, enter)$ and $G_1 = (s_{34}, enter)$ are formed for sub-tasks T_0 and T_1, respectively. The edge between sub-tasks T_0 and T_1, denoted by E, in the extracted graph shows the relation between these sub-tasks. T_0 as the sub-task is formed of $S_0 = \{s_1, \ldots, s_{20}\}$ and $G_0 = (s_7, enter)$ and it does not have any childs. T_1 as the sub-task is formed of $S_1 = \{s_{21}, \ldots, s_{40}\}$, $G_1 = (s_{34}, enter)$, and its child is $C_1 = T_0$. T_0 and T_1 nodes can be considered as the *abstract states* and their corresponding policies as the *abstract actions*[2]. The policy of sub-task T_0 is a local policy for S_0 that leads reaching to s_{27} by G_0. The policies of sub-tasks can be obtained of the episodes. The sequences of states of episodes are used as the transactions to extract the exits, and the corresponding actions which lead to the exits are used to extract the policies.

D. AN EXAMPLE OF HST CONSTRUCTION OF SEVERAL RULES

Let us consider an example with several rules, $AR_1 = bcde$, corresponded to $b, c, d \rightarrow e$, $AR_2 = dbce$, corresponded to $d, b, c \rightarrow e$, $AR_3 = acde$, corresponded to $a, c, d \rightarrow e$ (see Figure 4). In the first step, the proposed algorithm constructs a tree with the reverse of AR_1, creating one branch with values *edcb*. Then, the reverse of AR_2 is added to the tree, making a new branch from c since $AR_{2,2} = c$ cannot be matched in the tree from that point in line 18 of Algorithm 3. Thus, a new branch from e is created to contain the remaining values of AR_2. Finally, the reverse of AR_3 is added to the tree, where a mismatch happens in $AR_{1,4}$ that results as a new branch created at node c.

[2] In the task hierarchy, the sub-tasks are considered as the abstract states and their corresponding local policies as the abstract actions [21].

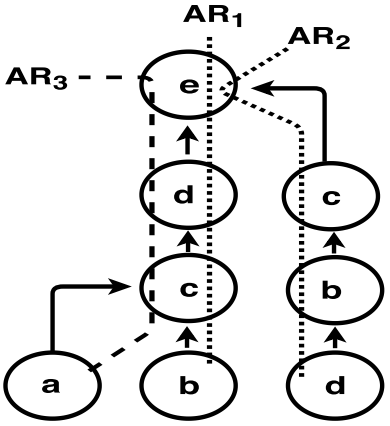


FIGURE 4. An example of a HST-construction of several rules.

V. THEORETICAL ANALYSIS

In this section, we provide a theoretical analysis to study the properties of the extracted hierarchical structure of the tasks using the proposed *SARM-HSTRL* method. The problem of extracting a hierarchical structure in RL can be considered as a hierarchical credit assignment problem of MAXQ, and the proposed *SARM-HSTRL* provides a solution to automatically perform such extraction in MDPs and FMDPs. Since the required convergence conditions of *SARM-HSTRL* are different in MDPs and FMDPs, its theoretical analysis has been evaluated in each domain separately.

Theorem 1: The proposed *SARM-HSTRL* converges to a hierarchical optimum policy in MDPs. A hierarchical policy is defined as an assignment of a local policy to each sub-task. A hierarchical optimum policy is a hierarchical policy that makes the best accumulated reward [24].

Proof: Using Definition 2 in [21], the proof to show the extracted HST leads to a hierarchical optimal policy is straightforward if we show that only one variable of X changes as a result of each action; then the Q function of such a hierarchy can be recursively expanded and mapped to a Q function of a flat MDP. In our proposed method in MDPs, each node of HST corresponds to a sub-MDP and based on Definition 1, there exists only one variable in our state variable, X. Therefore, the proposed *SARM-HSTRL* converges to a hierarchical optimum policy as proved for HEX-Q algorithm [21]. Hierarchical execution can be applied by using a decomposed value function since the proposed

method similar to MAXQ breaks down the MDP to interlinked sub-MDPs directly. Q function for each node, exit, as a sub-MDP of the tree is defined recursively as follows:

$$Q^*_{T_i}(s^{T_i}, a) = \sum_{s'} P^a_{s^{T_i}s'}[R^a_{s^{T_i}s'} + V^*_{T_i}(s')]$$

where s' is the hierarchical next state. $Q^*_{T_i}(s^{T_i}, a)$ shows the expected value of node T_i after performing (abstract) action a in (abstract) state s^{T_i} and in the continue pursuing the optimal hierarchical policy. $V^*_{T_i}(s)$ is the decomposition of optimal hierarchical value function that is calculated recursively as follows:

$$V^*_{T_i}(s) = \max_a [V^*_{C_i(a)}(s) + Q^*_{T_i}(s^{T_i}, a)]$$

where $V^*_{C_i(a)}$ shows the child of T_i implementing action a. □

Next, we study the convergence of the proposed method in FMDPs. For FMDPs, if the state abstraction and temporally extended actions are constructed based on one state variable in each layer, then the proof of theorem 1 is still valid (as shown in HEX-Q [21]). However, the assumed condition in HEX-Q of only having changes in one state variable for FMDPs is not a practical assumption; therefore here we evaluate the optimality of the *SARM-HSTRL*'s solution for a general case. Basically, there is not a straightforward proof for convergence of methods which extract the hierarchical structure of tasks in FMDPs [22], [24]. It is proven in [53] that having the stochastic substitution and reward respecting characteristics preserves the optimality for reduced MDPs such as FMDPs. Thus, stochastic substitution and reward substitution can be used to prove the optimality by showing that each reduced MDP has the mentioned characteristics. Next, we review such characteristics of the proposed method.

Definition 4: Transaction, Ω, a set of extracted trajectories, is called *representative* if Ω includes all possible state action pairs that lead to the ultimate goals.

In *SARM-HSTRL*, the trajectories are used instead of high-level sources of knowledge (e.g, DBNs). DBNs show the casual relations among the state variables for each action; the HRL models based on DBNs can present irrelevant states variables in the state abstraction. More importantly, as we mentioned earlier, the assumption of having DBNs in advance is not practical in autonomous settings. Our proposed method solves this problem by extracting the relations among the states and the state abstraction in an autonomous manner, where the trajectories are the only source of knowledge to show the effects of actions on state variables.

Definition 5: A non-redundant trajectory is defined as a trajectory which is not possible to remove one or more of its states and action pairs such that the remaining sequence still leads to the goal states [24].

In [24], a trajectory-task pair, $\langle \Omega_k, T_i \rangle$, where $\Omega_k = \langle s_0, a_0, \ldots, s_{n-1}, a_{n-1}, s_n \rangle \subset \Omega$, is called *consistent* with H if the following two conditions are held: i) sub-task T_i, as an SMDP, corresponds to a node in H; ii) if the observed states except the last two ones in Ω_k are a subset of S_i; in other words, $\{s_0, \ldots, s_{n-2}\} \subseteq S_i$ and $\{s_0, \ldots, s_{n-2}\} \cap G_i = \emptyset$. Also, (s_{n-1}, a_{n-1}) is an exit of G_i of the sub-task T_i. Clearly, a trajectory Ω_k is consistent with the extracted HST, H, if $\langle \Omega_k, T_0 \rangle$ is consistent with H.

Theorem 2: If each member of the set of trajectories, *transactions* $\Omega = \langle \Omega_1, \ldots, \Omega_m \rangle$, is non-redundant, *SARM-HSTRL* builds a hierarchy H, as every trajectory of the set is consistent with H.

Proof: Our proposed method first generates a hierarchy H based on the extracted association rules of the representative and non-redundant trajectories. Since a sequential ARM is used, it selects a sequence of states of trajectories that preserves the appeared order of them as the association rules. These association rules as a whole are added to the HST tree one by one and if two nodes cannot be matched, a branch of the parent node of H is created (i.e., *lines 17-20* of Algorithm 3). If a trajectory Ω_k is denoted by $\Omega_k = \langle s_0, a_0, \ldots, s_{n-1}, a_{n-1}, s_n \rangle$, the proposed method finds the conjunction of values of X that are true in s_{n-1} and not before that and assign them to the goal G_i [24]. If there are not such values of X, some suffix of the sequence can be disregarded without any impacts to achieve the goal, it is a contradiction with the property of non-redundancy. As a consequence, S_i will be the set of all states which do not satisfy G_i; thus, $\{s_0, \ldots, s_{n-2}\}$ will satisfy the required condition to be in S_i.

It is noted that the trajectories can be considered as a sequence of sub-trajectories, where each of these partitioned sub-trajectories is a conjunction of values of X as termination conditions of that sub-trajectory. With this, the above argument can be applied to each sub-trajectory recursively [23]. □

Definition 6: A hierarchy is called *safe* if it guarantees that "the state variables in each task are sufficient to capture the values of any trajectories consistent with the sub-hierarchy rooted at that task node [24]". In fact, the concept of *safe* refers to the stochastic substitution and reward respecting for sub-tasks as mentioned in [53].

Theorem 3: The hierarchical structure of tasks being extracted by *SARM-HSTRL*, H, of a representative Ω guarantees that "the total expected reward during each trajectory of Ω is only a function of the values of $x \in X_i$ in the starting state of Ω [24]" for any trajectory-task $\langle \Omega_j, T_i \rangle$ that is consistent with H. Also, there is just one hierarchical structure of tasks that can be extracted based on the extracted exit states, sub-goals, which are safe with respect to Ω.

Proof: *SARM-HSTRL* constructs $T_i = \langle X_i, S_i, C_i, G_i \rangle$ directly based on the sub-goals, exit states, of several trajectories. The sub-goals are used to partition the sequence of states of trajectories, Ω. The actions in any sequence of state-action pairs of each trajectory are primitive and change the values of X_i as their resultant states. Their resultant states are in the same partition, except the exit states as termination conditions, G_i. If it changes the variables outside of the current sub-task, T_i, that variable, X_k, should appear in the sequence of state-actions pairs before exit states' variables of T_i. Thus, it will be placed inside of sub-task T_i which is a contradiction

with the assumption that it can have effects on variables more than X_i which are outside of current sub-task. In the same way, we can say that all immediate rewards in the trajectory are functions of the variables in X_i. Therefore, the summation of discounted rewards and the probability of transition in each trajectory are just related to X_i; therefore, the extracted hierarchical structure of tasks is safe with respect to Ω. The proposed method forms the sub-tasks of sub-goals all in once; thus, if there is another hierarchy, H', as it is consistent with Ω, it will violate the *safe* characteristic with respect to Ω. This completes the prove that the extracted hierarchy by *SARM-HSTRL* leads to the hierarchical optimal policy. □

Theorem 4: The extracted hierarchical structure of tasks using *SARM-HSTRL* provides the most efficient, reliable, and compact hierarchical structure considering the representative and non-redundant set of trajectories, Ω, when the problem is sparse in both of MDPs and FMDPs. Efficiency is measured by the probability of usage and the resultant performance. Reliability is a function of the accuracy for the certainty of occurrence of next sub-tasks depending on which sub-tasks have been done so far. Resultant performance captures the compactness concept and is defined as how much the extracted structure abstract the action space. Thus, efficient sub-tasks are considered as the sub-tasks which summarize the longest frequent sequence of actions in temporally extended actions.

Proof: The sub-tasks are extracted based on the support and confidence measures in the form of association rules as the most efficient and reliable sequence of sub-tasks, exit states, among the representative and non-redundant trajectories, Ω. The *support* measure checks the ratio of witnessing all possible sub-tasks to all observations in Ω. Thus, it finds the sub-tasks that happen with the highest probability related to other ones. In other word, these sub-tasks are the best summarization, the longest and the most frequent of what happened in the past. Reliability implies providing the highest accuracy of predicting the next sub-tasks based on summarization of several trajectories and what have been done so far. The *confidence* measure evaluates every possible sequence by constructing a tree considering all possible eligible sequences among several trajectories. It preserves their sequences and compacts the extracted sub-tasks in the form of sequential association rules by matching and mapping them from the last task to the first ones. Also, it can be said the required size for value function table is a function of the depth, l, and branch, d, of the tree. The branch of the tree is the number of sequences of sub-tasks as they cannot be matched to the current nodes of the tree. The depth of the tree is the number of sub-tasks that in the worst case equals to the length of trajectory when they are not sub-parts of each other. Thus, the space complexity of value function tables of the hierarchy is $O(ld)$. □

A. RELATIVE ADVANTAGES OF SARM-HSTRL

In this section, a summary of key advantages of the proposed *SARM-HSTRL* related to other methods is provided. One key contribution of this method is that despite other methods in the literature that are restricted to only MDPs or FMDPs, *SARM-HSTRL* can be applied in both MDPs and FMDPs since it does not need in advance knowledge such as the state transitions, or some knowledge or constraints about the size of abstraction or reversible state transitions. Our proposed method works from scratch based on trajectories and without the need for the state transition graph or DBNs, and considers both topological and value intrinsic relationships and structures in trajectories to extract the hierarchical structure of tasks.

The proposed *SARM-HSTRL* can also outperform the HI-MAT algorithm in the sense that HI-MAT only works on a single successful trajectory, while in many RL settings, there are several optimal or near-optimal trajectories that cannot be represented in HI-MAT, unless it is generalized by using another function (i.e., *action generalization*). However, our proposed method does not require a single, carefully formed, trajectory, and it can efficiently handle the funnel property of sub-tasks, while HI-MAT cannot be generalized from many different starting places in a few terminal states (i.e., it does not have the *funnel* property [23]).

SARM-HSTRL searches for the sub-goals, and the number of sub-goals in an RL task is much less than the size of state space. Thus, using the FP-growth algorithm is efficient and practical in *SARM-HSTRL* when the state space is large, and the number of sub-goals is relatively low. Such sparsity is a very common assumption in HRL methods [22], [24]. In scenarios where the state space is small, or there is a considerable level of similarities among the successful trajectories, then many states will be visited frequently, and hence detected by the association rule mining algorithm as the sub-goals. This may, in turn, degrade the performance of the proposed method. In the domains that there is a considerable dynamism or they have small state spaces, probably other techniques based on the flat representation of sub-goals work faster and are more practical.

Finally, the proposed *SARM-HSTRL* method can be easily scaled up to high dimensional discrete action space and even continuous action space as it considers all paths together at once. The complexity of *SARM-HSTRL* is a function of complexity of FP-growth algorithm as its main component to extract the association rules, which is proven to be very practical in terms of time complexity for real usages [50], [52].

B. TIME COMPLEXITY

In this section, we discuss the time complexity of the proposed *SARM-HSTRL*. The association rule mining has considerably better performance compared to conventional correlation extraction methods such as mutual information or statistical hypothesis testing, since these methods are often not able to precisely extract the intrinsic correlations among these variables [50]. However, ARM improves upon such simple methods by using a combination of two key measures of *support* and *confidence* in a proven scalable extraction strategy to obtain and evaluate the most efficient

and reliable relationships among the variables in the datasets. The number of rules can be calculated as $R = 3^d - 2^{d+1} + 1$, in which d shows the number of items as shown by [50]. Therefore, the exponential growth makes the evaluation impractical to enumerate all possible rules in large datasets in a naive manner. Also, there is an exponential growth in Rule Generation, each frequent k-itemset has $2^k - 2$ rules, where k is the number of items of the corresponding itemset [50]. Thus, ARM is a necessity to handle such large search space for practical situations without compromising the optimality.

As mentioned in [50], "the size of a FP-tree typically is smaller than the size of the uncompressed data," and in the worst-case scenario, the size of a FP-tree is effectively equal to the size of the data. The performance of the FP-growth algorithm is related to the compact factor of the trajectories and the value of *minsup*. In the worst-case scenario, the support values of all combination of items are larger than *minsup*, and 2^{d+1} itemsets will be generated, where d is the number of items.

As mentioned above, *SARM-HSTRL* by using FP-growth algorithm method provides a promising solution in practical applications where the state space is large and sparse. If the state space is small, or the successful trajectories have many similarities to each other, many states will be visited frequently, and hence detected by ARM as the sub-goals. Clearly, the concept of sub-goals becomes meaningless in such conditions. Another possible scenario to consider is when the adjacent states around the sub-goals are visited frequently. For both these conditions, one efficient solution is to cluster the adjacent sub-goals as one entity and create one corresponding temporally extended action for that entity. t, order of occurrence, for each state in each trajectory is already stored by *SARM-HSTRL* as they are used in HST for possible orderings of the sub-goals. They can be also used to find the close sub-goals for clustering purposes.

VI. EXPERIMENTAL RESULTS

In this section, several experimental results are presented to evaluate the performance of *SARM-HSTRL* on four different testbeds. In the first two experiments, the agent has 5 actions, *press-key* and 4 movement primitive actions. The *press-key* does not change the place of the agent. The agent can move with its primitive actions in four directions: *up, right, down, left*. If there is a wall in the way, the agent stays in its current state. In all of the experiments, if the agent does the *press-key* action, it will receive a reward of 0 in the sub-goals and a reward of -10 in other states. The reward of other actions is -1. The agent movement with probability 0.8 is according to an intended action and is randomly in one of the directions with probability 0.2. The discount factor is set to $\gamma = 0.9$.

In constructing the HST, 10 start and goal states are chosen randomly. A goal state is defined as an important, task-specific state that ends an episode once visited. A start state, s_0, is a state from which an agent begins an episode. For each of them, the agent starts the learning using a common learning mechanism such as Q-learning; the learning is finished after 5000 episodes. They are ordered based on the accumulated reward, and the best five ones are selected. They are given to the *SARM-HSTRL* and the HST produces a hierarchical structure of tasks based on the whole length of transactions. Now, the sub-tasks are formed for the agent and the HST helps the agent to choose their phase of learning. If they are expanded as primitive actions, the number of steps to reach a goal is equal to the number of action selection calls.

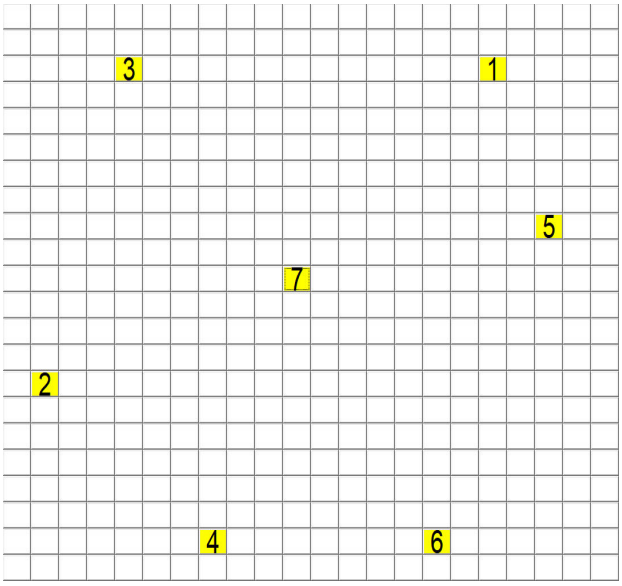
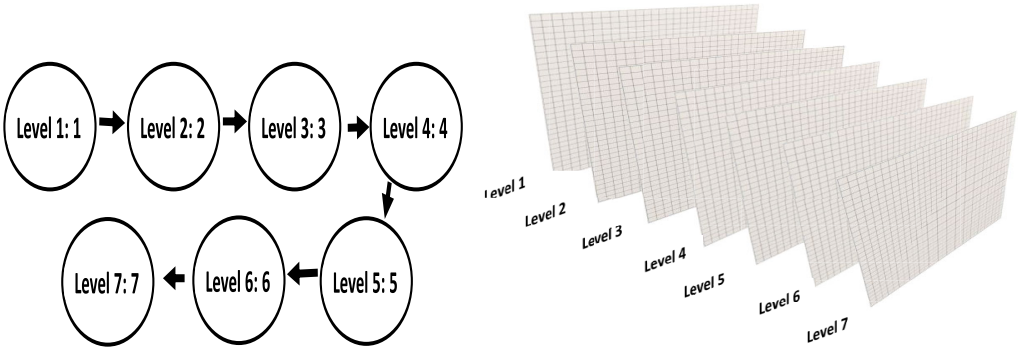


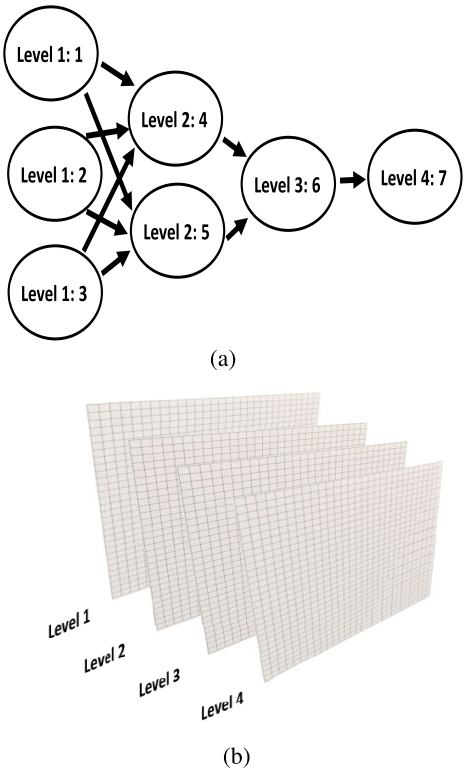
FIGURE 5. The structure of the first testbed: the size of the maze is 22 × 22 and it has 7 sub-goals. The sub-goals are colored with yellow.

The performance of *SARM-HSTRL* in HRL is evaluated for two different hierarchical structures of tasks, experiment 1 as shown in Figure 6 and experiment 2 as shown in Figure 7. In these figures, for the sake of comparison between Q-learning, Cascading Decomposition [20], HI-MAT [23] and *SARM-HSTRL*, 10 runs are considered where in each of them, a start state and a goal state are chosen randomly. The maximum number of actions for each episode is 4000, and the total number of episodes is 8000. *SARM-HSTRL* is compared to Cascading Decomposition as a representative approach in MDPs, which is discussed in [8], [20], is the latest and considerable improvement for methods proposed in [3], [14], [16]. As seen in Figures 8.(a) and 8.(b), the proposed *SARM-HSTRL* method results in a hierarchical optimum policy task structure, as does HI-MAT, while our method does not rely on any prior knowledge (i.e., DBNs). It has been proven in [23] that HI-MAT leads to better results compared to VISA, and this implies that *SARM-HSTRL* outperforms the VISA method too. It is worth mentioning that HI-MAT cannot be implemented in experiment 2 including multiple successful trajectories, as it can only work with one successful trajectory that interprets the tasks. Therefore, HI-MAT does not appear in Figures 8.(c) and 8.(d).



(a) (b)

FIGURE 6. (a) The first task hierarchy that constituted of 8 layers (7 levels +1) of Figure 5 for experiment 1. The number after ":" shows the corresponding sub-goal in Figure 5 that enables moving from the current level to the next one. (b) In other words, each level is Figure 5 in which the corresponding sub-goals of each level that provide moving to the next level are shown in part (a).



(a) (b)

FIGURE 7. (a) The first task hierarchy that constituted of 5 layers (4 levels +1) of Figure 5 for experiment 2. The number after ":" shows the corresponding sub-goal in Figure 5 that provide moving from the current level to the next one. (b) In other words, each level is Figure 5 in which the corresponding sub-goals of each level that provide moving to the next level are shown in part (a).

The proposed method is implemented in MATLAB and the size of the of the first testbed is 22×22 and it has 7 sub-goals. The sub-goals are colored with yellow, Figure 5. The size of the second testbed is 30×16, Figure 9a. In experiment 1, Figure 6, the task hierarchy has 7 levels — it has $(484 \times (7+1)) = 3872$ states. If the agent enters in sub-goals states in the following order 1, 2, 3, 4, 5, 6 and 7 and does the *press-key* action in each of them, and then enters in the goal state of the run and performs the *press-key* action again, the agent receives a reward of +10, and the episode will be finished. The value of *minsup* is 0.9 and the value of *minconf* is 0.9.

In experiment 2, Figure 7, the task hierarchy has 4 levels, but with a more complicated structure- it has $(484 \times (4+1)) = 2420$ states. If the agent enters in one of the sub-goal states from the leaves of tree 1 or 2 or 3 first, then as the second level enters in one of their parent 4 or 5, then as the third and fourth level in 6 and 7 in order and does the *press-key* action in each of them, and finally enters in the goal state of the run and performs the *press-key* action, the agent receives a reward of +10 and the episode will be finished. The value of *minsup* is 0.3 and the value of *minconf* is 0.9.

There is a significant difference in speed of learning between the proposed method with Cascading Decomposition and Q-learning as shown in Figures 8.(b) and 8.(d). The most important attribute of the SMDP framework is using temporally extended actions to decrease the number of steps. As it is shown in HRL in Figures 8.(a) and 8.(c), the temporally extended actions considerably decrease the number of steps. p-values have been calculated between the proposed method with Q-learning and Cascading Decomposition in each diagram by using the t-test for $\alpha = 0.01$; the significant change is validated — p-values are much smaller than 1×10^{-5}.

In experiment 3, the accuracy of *SARM-HSTRL* is evaluated thorough all possible sub-goals, where 10 random states for the start and goal states are selected. The minsup and minconf are set to 0.6 and 0.9, respectively for Figure 9.(a). The agent has four actions *up, right, down, and left*. Both the stochastic rate and learning rates are 0.1, and the discount factor and the $\epsilon - greedy$ are the same as the previous experiments. The agent receives a reward of zero for each action, unless it enters to the goal state, where it receives 10. The number of trials is 500 for each pair of start and goal

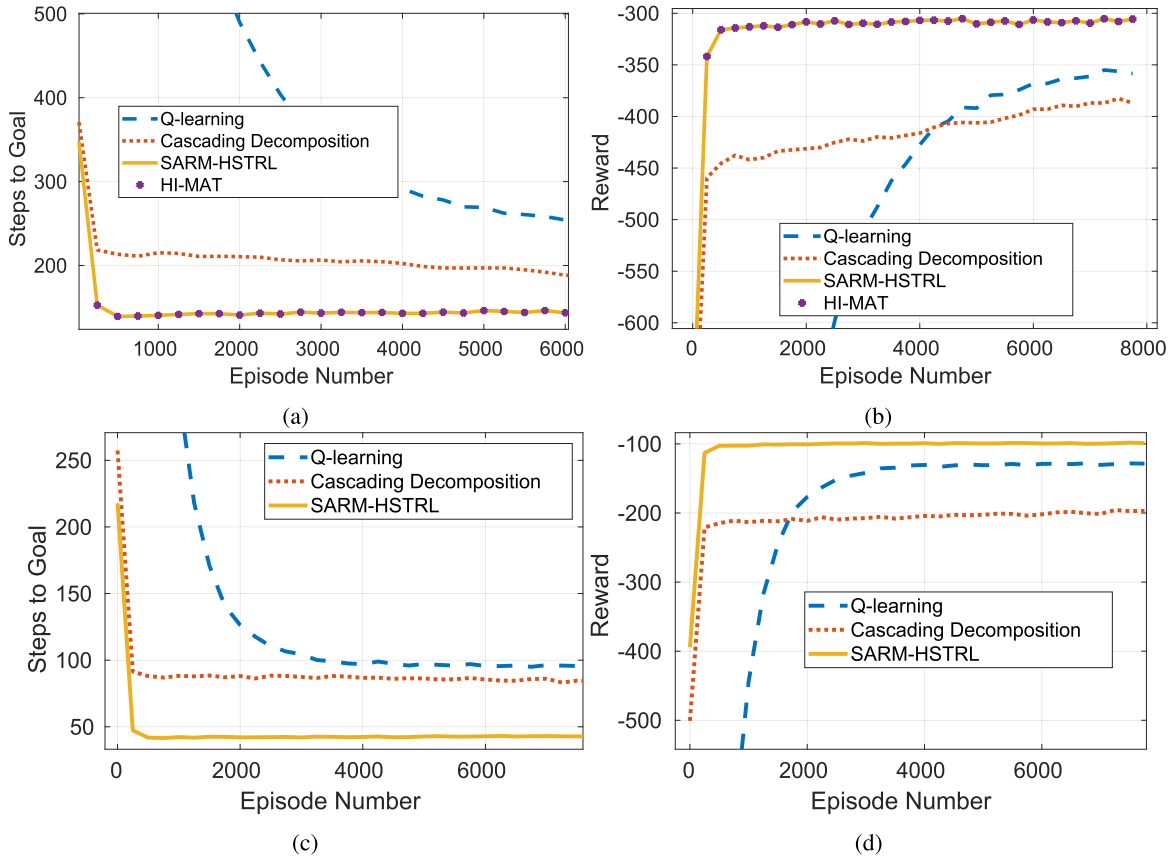


FIGURE 8. The diagrams of comparing the performance of SARM-HSTRL with Q-learning, HI-MAT, and Cascading Decomposition. Performance comparison of *SARM-HSTRL* with Q-learning, HI-MAT, and Cascading Decomposition in experiment 1 (Fig. 6) and experiment 2 (Fig. 7). Since experiment 2 including multiple successful trajectories, HI-MAT cannot be implemented. HI-MAT can only work with one successful trajectory that interprets the tasks. (a) Represents the number of steps along episodes in experiment 1. (b) Comparison of receiving rewards along episodes in experiment 1. (c) Represents the number of steps along episodes in experiment 2. (d) Comparison of received rewards along episodes in experiment 2.

states that 5 of the best trajectories are used. As it can be seen in Figure 9.(b), *SARM-HSTRL* detects the sub-goals properly. *SARM-HSTRL* with the given threshold did not consider all the possible sub-goals in the right side of Figure 9.(b) since the middle ones are placed in better policies, they can reach the possible goals with more probability and fewer actions.

In experiment 4, we aim to show the accuracy of *SARM-HSTRL* in FMDPs, using Taxi driver problem as a known testbed (Figure 9.(c)). We scale up both place dimensions of Taxi driver problem for 4 times to reach 20×20. The taxi domain is composed of a 5×5 grid world, a taxi, and a passenger, where the taxi starts from a random place and *pick-up* the passenger from one of those places (*B, G, R, and Y*) and *put-down* the passenger in one of these places. The place of *pick-up* and *put-down* are chosen randomly. The taxi has six primitive actions, *north, south, east, west, pick-up, and put-down*. The agent receives a reward of -1 for movement actions, a reward of -10 for wrongly doing the action *pick-up or put-down*, and a reward of $+20$ for successfully completing the mission. Each action succeeds in its job with the probability of 0.8 in each state and it has a random effect in that state with the probability of 0.2. The number of trials is 2000, and 16 random start and goal states to capture all possible combinations of *pick-up* and *put-down*. The maximum number of actions is 1000 in each trial. *minsup* is set to 0.0625 and *minconf* is set to 0.7. The *minsup* value is selected as 0.0625, noting that there are 16 combinations for *pick-up*, and *put-down*- $1/16 = 0.0625$. Discount factor, $\epsilon - greedy$, and learning rate have been initialized similar to experiment 1. As is shown in Figure 9.(d), the number of observing detected sub-goals for *pick-up* is correct (the brightest ones). Also, some states in the paths to sub-goals are visited more frequently, and therefore they are being detected as the sub-goals. For example, when these states are in the optimal paths of several sub-goals, or they are adjacent to the wall states, they will be visited more because of the stochastic rate. They can be easily pruned by considering the sequence and their adjacency to states with the largest support. There is another way in such condition, where the adjacent extracted sub-goal states can be considered as a cluster to define just one temporally extended actions for them.

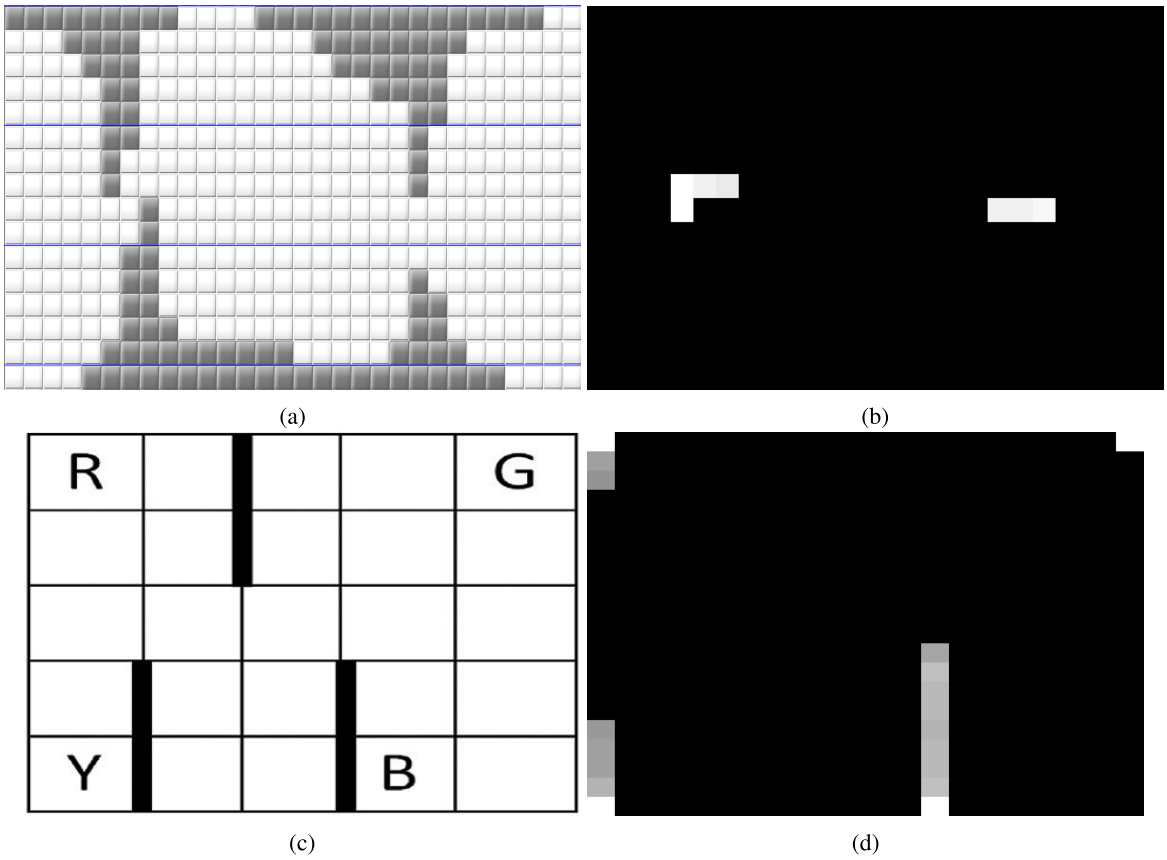


FIGURE 9. The testbeds and the results that show the accuracy of detection of *SARM-HSTRL* on two testbeds. (a) A maze world. (b) The frequency of visiting detected sub-goals by *SARM-HSTRL* in transitions. (c) Taxi driver problem as an example in FMDPs. (d) The frequency of visiting the detected sub-goals by *SARM-HSTRL* in transitions in a 4 times scale in places' dimensions of Taxi driver problem, 16 times larger state space. The states near to wall states of three places are more probable to visit because of they experience less influence of the stochastic rate and the place of pick up places. Also, the four places as the *SARM-HSTRL* are detected correctly that have the most observing, the brightest ones.

VII. CONCLUSION

An HRL method called *SARM-HSTRL* is proposed to autonomously extract a task hierarchy for RL by using a sequential association rule mining approach, where multiple sub-goals are extracted as frequently visited states from the successful trajectories in the form of association rules. These sub-goals are used to define exits as the termination conditions to form temporal and state abstractions. The current methods in MDPs can only extract a flat hierarchy, (i.e., one level) which means that these methods only find the sub-goals or the bottlenecks, rather than a hierarchical structure of those. Despite the majority of the previously proposed HRL methods in FMDPs (e.g., HI-MAT and VISA) that rely on DBNs model to use prior knowledge about the effects of actions on state variables, our proposed method independently extracts the relations among states and state abstraction. Moreover, since DBNs show the causal relations among the state variables for each action, it can determine irrelevant states variables for state abstraction. However, the proposed method only extracts the relevant correlations. The convergence of the proposed method to a hierarchical optimal solution is proven for both MDPs and FMDPs. We also proved that the extracted structure is the most efficient, reliable, and the compact hierarchical structure for discrete MDPs and FMDPs.

The experimental results show a considerable improvement in the speed and quality of the learning process for the analyzed experiments. It is expected that the extracted hierarchical structure in the form of sub-tasks provides a supplementary, more robust, and higher level of knowledge to be transfered among the sub-tasks rather than sharing value functions, which are highly sensitive to the type and the amount of similarity between the source and target domains. Therefore, the decomposed structure of tasks based on *SARM-HSTRL* provides an abstraction that an agent can reuse, generalize, and transfer to new domains. This work is the first one to use the idea of sequential association rule mining in HRL and even RL. Therefore, we used the SARM method in its original form to prove its superior performance and capabilities from both theoretical and experimental aspects. There are more developed versions of SARM that will be studied in our future works. Also, applying the proposed SARM-HSTRL in deep RL algorithms can be considered as future works. There are several types of deep

RL techniques that can potentially benefit from using our proposed HRL technique.

ACKNOWLEDGMENT

The authors would like to appreciate the reviewers' comments and suggestions which helped to improve the research and this article.

REFERENCES

[1] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dyn. Syst.*, vol. 13, no. 4, pp. 341–379, 2003.
[2] A. McGovern and A. G. Barto, "Autonomous discovery of temporal abstractions from interaction with an environment," Ph.D. dissertation, Univ. Massachusetts, Boston, MA, USA, 2002.
[3] M. Stolle, "Automated discovery of options in reinforcement learning," Ph.D. dissertation, McGill Univ., Montreal, QC, Canada, 2004.
[4] T. G. Dietterich, "Hierarchical reinforcement learning with the MAXQ value function decomposition," *J. Artif. Intell. Res.*, vol. 13, pp. 227–303, Jul. 2018.
[5] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *J. Mach. Learn. Res.*, vol. 10, pp. 1633–1685, Jul. 2009.
[6] B. L. Digney, "Learning hierarchical control structures for multiple tasks and changing environments," in *Proc. 5th Int. Conf. Simul. Adapt. Behav. Animals Animats*, vol. 5, 1998, pp. 321–330.
[7] C.-C. Chiu and V.-W. Soo, "Subgoal identifications in reinforcement learning: A survey," in *Advances in Reinforcement Learning*. Rijeka, Croatia: InTech, 2011.
[8] B. Ghazanfari and N. Mozayani, "Extracting bottlenecks for reinforcement learning agent by holonic concept clustering and attentional functions," *Expert Syst. Appl.*, vol. 54, pp. 61–77, Jul. 2016.
[9] X. B. Peng, G. Berseth, and M. Van De Panne, "Terrain-adaptive locomotion skills using deep reinforcement learning," *ACM Trans. Graph.*, vol. 35, no. 4, pp. 1–12, Jul. 2016.
[10] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne, "DeepLoco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Trans. Graph.*, vol. 36, no. 4, pp. 1–13, Jul. 2017.
[11] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1312–1320.
[12] C. Tessler, S. Givony, T. Zahavy, D. J. Mankowitz, and S. Mannor, "A deep hierarchical approach to lifelong learning in minecraft," in *Proc. AAAI*, vol. 3, 2017, p. 6.
[13] A. Vezhnevets, V. Mnih, S. Osindero, A. Graves, O. Vinyals, and J. Agapiou, "Strategic attentive writer for learning macro-actions," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 3486–3494.
[14] S. Mannor, I. Menache, A. Hoze, and U. Klein, "Dynamic abstraction in reinforcement learning via clustering," in *Proc. 21st Int. Conf. Mach. Learn. (ICML)*, 2004, p. 71.
[15] Ö. Şimşek and A. G. Barto, "Using relative novelty to identify useful temporal abstractions in reinforcement learning," in *Proc. 21st Int. Conf. Mach. Learn. (ICML)*, 2004, p. 95.
[16] Ö. Şimşek and A. G. Barto, "Skill characterization based on betweenness," in *Proc. Adv. Neural Inf. Process. Syst.*, 2009, pp. 1497–1504.
[17] C. Drummond, "Accelerating reinforcement learning by composing solutions of automatically identified subtasks," *J. Artif. Intell. Res.*, vol. 16, pp. 59–104, Jul. 2018.
[18] I. Menache, S. Mannor, and N. Shimkin, "Q-cut-dynamic discovery of sub-goals in reinforcement learning," in *Proc. Eur. Conf. Mach. Learn.* Springer, 2002, pp. 295–306.
[19] Ö. Şimşek, A. P. Wolfe, and A. G. Barto, "Identifying useful subgoals in reinforcement learning by local graph partitioning," in *Proc. 22nd Int. Conf. Mach. Learn. (ICML)*, 2005, pp. 816–823.
[20] C.-C. Chiu and V.-W. Soo, "Automatic complexity reduction in reinforcement learning," *Comput. Intell.*, vol. 26, no. 1, pp. 1–25, Feb. 2010.
[21] B. Hengst, "Discovering hierarchy in reinforcement learning with HEXQ," in *Proc. ICML*, vol. 19, 2002, pp. 243–250.
[22] A. Jonsson and A. Barto, "Causal graph based decomposition of factored MDPs," *J. Mach. Learn. Res.*, vol. 7, pp. 2259–2301, Dec. 2006.
[23] N. Mehta, S. Ray, P. Tadepalli, and T. Dietterich, "Automatic discovery and transfer of MAXQ hierarchies," in *Proc. 25th Int. Conf. Mach. Learn. (ICML)*, 2008, pp. 648–655.
[24] N. Mehta, S. Ray, P. Tadepalli, and T. Dietterich, "Automatic discovery and transfer of task hierarchies in reinforcement learning," *AI Mag*, vol. 32, no. 1, p. 35, Jul. 2017.
[25] M. Wynkoop and T. Dietterich, "Learning MDP action models via discrete mixture trees," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*. Springer, 2008, pp. 597–612.
[26] B. Da Silva, G. Konidaris, and A. Barto, "Learning parameterized skills," 2012, *arXiv:1206.6398*. [Online]. Available: https://arxiv.org/abs/1206.6398
[27] G. Konidaris and A. G. Barto, "Efficient skill learning using abstraction selection," in *Proc. IJCAI*, vol. 9, 2009, pp. 1107–1112.
[28] G. Konidaris and A. G. Barto, "Skill discovery in continuous reinforcement learning domains using skill chaining," in *Proc. Adv. Neural Inf. Process. Syst.*, 2009, pp. 1015–1023.
[29] G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto, "Robot learning from demonstration by constructing skill trees," *Int. J. Robot. Res.*, vol. 31, no. 3, pp. 360–375, Mar. 2012.
[30] G. Konidaris, S. Kuindersma, R. Grupen, and A. G. Barto, "Constructing skill trees for reinforcement learning agents from demonstration trajectories," in *Proc. Adv. Neural Inf. Process. Syst.*, 2010, pp. 1162–1170.
[31] G. Konidaris, S. Kuindersma, R. A. Grupen, and A. G. Barto, "Autonomous skill acquisition on a mobile manipulator," in *Proc. AAAI*, 2011.
[32] K. Gregor, D. J. Rezende, and D. Wierstra, "Variational intrinsic control," 2016, *arXiv:1611.07507*. [Online]. Available: https://arxiv.org/abs/1611.07507
[33] A. S. Lakshminarayanan, R. Krishnamurthy, P. Kumar, and B. Ravindran, "Option discovery in hierarchical reinforcement learning using spatio-temporal clustering," 2016, *arXiv:1605.05359*. [Online]. Available: https://arxiv.org/abs/1605.05359
[34] M. C. Machado, M. G. Bellemare, and M. Bowling, "A laplacian framework for option discovery in reinforcement learning," 2017, *arXiv:1703.00956*. [Online]. Available: https://arxiv.org/abs/1703.00956
[35] J. Achiam and S. Sastry, "Surprise-based intrinsic motivation for deep reinforcement learning," 2017, *arXiv:1703.01732*. [Online]. Available: https://arxiv.org/abs/1703.01732
[36] P.-L. Bacon, J. Harb, and D. Precup, "The option-critic architecture," in *Proc. AAAI*, 2017, pp. 1726–1734.
[37] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 1471–1479.
[38] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine, "Diversity is all you need: Learning skills without a reward function," 2018, *arXiv:1802.06070*. [Online]. Available: https://arxiv.org/abs/1802.06070
[39] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel, "Vime: Variational information maximizing exploration," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 1109–1117.
[40] A. Klyubin, D. Polani, and C. Nehaniv, "Empowerment: A universal agent-centric measure of control," in *Proc. IEEE Congr. Evol. Comput.*, vol. 1, Dec. 2005, pp. 128–135.
[41] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 3675–3683.
[42] S. Mohamed and D. J. Rezende, "Variational information maximisation for intrinsically motivated reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 2125–2133.
[43] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jul. 2017.
[44] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, "Feudal networks for hierarchical reinforcement learning," 2017, *arXiv:1703.01161*. [Online]. Available: https://arxiv.org/abs/1703.01161
[45] C. Florensa, Y. Duan, and P. Abbeel, "Stochastic neural networks for hierarchical reinforcement learning," 2017, *arXiv:1704.03012*. [Online]. Available: https://arxiv.org/abs/1704.03012
[46] O. Sigaud and O. Buffet, *Markov Decision Processes in Artificial Intelligence*. Hoboken, NJ, USA: Wiley, 2013.
[47] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, vol. 1, no. 1. Cambridge, MA, USA: MIT Press, 1998.

[48] G. Bebek and J. Yang, "Pathfinder: Mining signal transduction pathway segments from protein-protein interaction networks," *BMC Bioinf.*, vol. 8, no. 1, p. 335, 2007.
[49] W. Lin, S. A. Alvarez, and C. Ruiz, "Efficient adaptive-support association rule mining for recommender systems," *Data Mining Knowl. Discovery*, vol. 6, no. 1, pp. 83–105, Jan. 2002.
[50] P. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining* (Always Learning). Reading, MA, USA: Addison-Wesley, 2006. [Online]. Available: https://books.google.com/books?id=KZQ0jgEACAAJ
[51] P.-N. Tan, M. Steinbach, A. Karpatne, and V. Kumar, *Introduction to Data Mining*, 2nd ed. London, U.K.: Pearson, 2018.
[52] W. A. Kosters, W. Pijls, and V. Popova, "Complexity analysis of depth first and fp-growth implementations of apriori," in *Proc. Int. Workshop Mach. Learn. Data Mining Pattern Recognit.* Springer, 2003, pp. 284–292.
[53] T. Dean and R. Givan, "Model minimization in Markov decision processes," in *Proc. IAAI*, 1997, pp. 106–111.

BEHZAD GHAZANFARI is currently pursuing the Ph.D. degree with Northern Arizona University. He is working on reinforcement learning, multiobjective reinforcement learning, deep learning, multiagent systems, and bio-medical signal processing.

FATEMEH AFGHAH is currently the Director of the Wireless Networking and Information Processing (WiNIP) Laboratory. Before joining NAU, she was an Assistant Professor with the Electrical and Computer Engineering Department, North Carolina A&T State University, from 2013 to 2015.

She is a Representative of the IEEE regions R1-6, on the membership board standing committee for the IEEE Signal Processing Society. Her research areas include wireless communications, game theoretical optimization, and biomedical signal processing. Her current research focuses on developing predictive modeling techniques using game theory and graph theory to optimize the performance of current medical diagnosis methods. She also works on optimizing the performance of autonomous multiagent systems and wireless communications networks.

MATTHEW E. TAYLOR received the Ph.D. degree from the Department of Computer Sciences, UT-Austin, in 2008. After a postdoctoral position at the University of Southern California, he was a Professor with the Lafayette College and with Washington State University. He is currently a Principal Researcher with Borealis AI, a Canadian Institute funded by the Royal Bank of Canada, where he helps to lead a research team in Edmonton focused on reinforcement learning.