Hierarchical Logical Clustering for State-Action Pair Sequences in Markov Decision Process

Shuqi Dai
Dept. of Computer Science
Carnegie Mellon University
shuqi@cs.cmu.edu

Sophie Guo
Dept. of Computer Science
Carnegie Mellon University
yueguo@cs.cmu.edu

Nov 26th, 2019

Abstract

When an expert performs a task that contains multiple sub-goals, how can we infer the underlying logical dependencies, i.e. similarity and potential hierarchy? For example, if the left hand holds a fork and the right hand holds a knife, can we tell the similarity of manipulating tableware? Can we understand the sub-goals performed by left and right hands separately can form a higher goal of eating? We are particularly interested in the hierarchical logical clustering of sub-sequential trajectories in the Markov Decision Process, each composed of state-action pairs in discrete time steps. By partitioning expert trajectories into sub-sequences and clustering recursively, we aim at building a hierarchical logical tree structure that understands the relations among sub-goals from expert demonstrations: are they parallel, sequential, or form a higher goal? We have tested the validity of our proposed system in the experiment on the noisy synthetic data, with a high success rate. The resulting hierarchical clusters could then be used for traditional Inverse Reinforcement Learning methods to compute the reward functions of the corresponding sub-goals, and can also be used in other Markov Decision Process domains.

1 Introduction

The problem we want to solve is the following:

- GIVEN: expert demonstrations set $D = \{\tau_0, ..., \tau_{n-1}\}$, where n is the number of demonstrations and τ_i is the ith trajectory. A trajectory τ_i consists of T_i state-action pairs $\{(s_i^{(0)}, a_i^{(0)}), ..., (s_i^{(T_i-1)}, a_i^{(T_i-1)})\}$. The actual construction of state-action pairs depends on specific scenarios, and please see the sections of Methods and Experiment for example details.

- FIND: tree-structured hierarchy structure for a set of strings $\{l_1, ..., l_R\}$, where each l is the letter label for the cluster of sub-sequences $\{(s^{(t)}, a^{(t)})\}$, and tell relations among all ls (whether sequential or parallel).
- OBJECTIVE: 1) minimize the sum of distance variance within each label; 2) reveal the true logical relations among all the labels as well as clusters in a hierarchy.

Successfully identifying the non-static sub-goals of observed experts and the connections between them can help autonomous agents or human novices learn more transparently, logically, and safely. For example, we expect an experienced taxi driver to have a good plan to plan many profitable trips, each has steps of 1) pick up a location that potentially has a good passenger, 2) judge which passenger to give a ride and stop in front of her or him, 3) drive to the destination. Each step, such as in 3), can be further segmented into smaller tasks of 1) pick a route, 2) start driving, 3) navigate to the freeway... It is clear that each tasks can always be segmented be a lower level, until it reaches the lowest level: break, turn left/right, accelerate etc. Clearly there is a hierarchy of tasks, but typically from taxi data unfortunately we can only see the full records of the car positions, the fuel condition, and mile counts! Can we understand more from the set of full information? e.g., is that possible to know the underlying connections among all the sub-goals?

Clustering the output trajectories of a Markov Decision Process (MDP) with a hierarchical structure has been explored widely in the Reinforcement Learning (RL) community, since the hierarchy potentially contains underlying important information such as sub-goals, options, and styles of agents etc. However, the challenges of manually crafting the reward functions result in the rise of Inverse Reinforcement Learning (IRL), where the agent has no knowledge of the reward of any state, but is instead given a variety of expert demonstration trajectories, clustering typically infers a sub-goal, i.e. locally consistent reward [10]. Recently, some work [16, 10, 14] have been done to recover the multiple sub-goals by clustering the segments of expert trajectories, aiming at learning one reward function for each cluster, representing one single sub-goal.

In the context of IRL, there has been rare work on developing a hierarchical clustering to recover hierarchical sub-goal structures. Most of them only utilize a flat clustering, i.e. two layers of hierarchy, instead of a real hierarchical one, and thus ignore the possible dependencies and structure information between the sub-goals [15]. Besides, some of the latest clustering are simply based on the state information [6], and ignore the information of action sequences. We aim to develop a method that fills this gap. However, we would only focus on the hierarchical clustering portion of this project and continue the IRL part after this class.

The main contributions of this project are:

- Pioneer in proposing hierarchical clustering algorithm that is able to cluster state-action dual sequence instead of single-line time series data (Figure 1).
- Pioneer in proposing hierarchical clustering framework for IRL.
- The proposed hierarchical clustering framework and algorithm can be generalized into applications in other areas such as music hierarchical structure discovery.

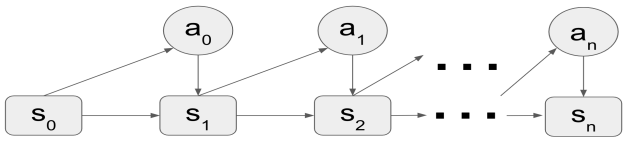

Figure 1: An example of state-action dual sequence in time series.

2 Survey

2.1 Papers read by Shuqi Dai

The first paper was about *G-means* clustering algorithm by Hamerly and Elkan [4].

- *Main idea*: Before this paper, most clustering algorithms required the user to specify the number of clusters k, however, it is often not clear what is the best value for k. This paper introduced an algorithm called *G-means* that can automatically learn k while clustering. The G-means algorithm is based on a statistical test for the hypothesis that a subset of data follows a Gaussian distribution, since the center-based clustering algorithms (in particular k-means and Gaussian expectation-maximization) usually assume that each cluster adheres to an unimodal distribution. The algorithm starts with a small number of k-means centers (e.g. initialize with just $k = 1$), and grows the number of centers. During each iteration, the algorithm runs k-means on the entire dataset and all the centers, if the data currently assigned to a k-means center does not appear to be Gaussian, then splits the k-means center into two centers. G-means repeatedly make split decisions until the statistical test accepts the hypothesis that the data assigned to each k-means center are Gaussian. In all, the only parameter supplied to G-means is the standard significance level of the statistical test α. The experiment showed that G-means works well at finding the correct number of clusters and the locations of genuine cluster centers, and in moderately high dimensions. It takes linear time and space in the number of data points and dimensions, and performs better than the Bayesian Information Criterion method.
- *Use for our project*: G-means can be a possible way for our clustering task since we do not know the exact number of clusters of each level in the hierarchical tree structure. We can run G-means from the bottom level up to the top. For each level, merge the states in a cluster to a new state after G-means clustering. The new state is the parent node of the merged states in that cluster.
- *Shortcomings*: 1) Like k-means, G-means is actually a convex optimization process, and assumes each cluster adheres to a Gaussian distribution; If the data has a non-convex distribution, we need to add a good kernel to solve the problem. 2) How to define and calculate the distance between data samples in our task is very critical to the clustering result. 3) If the hidden classes are highly imbalanced in the data, G-means can perform badly. 4) Might fall into a local minimum instead of a global minimum, use genetic k-means to solve this.

The second paper was on *Dynamic Time Wrapping* by Keogh and Ratanamahatana [5].

- *Main idea*: Dynamic Time Wrapping (DTW) is a much more robust distance measure for time series than the Euclidean distance. It allows an elastic shifting of the time axis, to accommodate sequences that are similar but out of phase. However, DTW's resistance to indexing made it too slow for most applications. This paper introduced a lower bounding measure that tightly approximates the true DTW distance and used it to allow exact indexing. Instead of using the original DTW, the model enforced a temporal constraint δ on the warping window size of DTW, which not only improved its computation efficiency, but also improved its accuracy for measuring time series similarity, as extended warping may introduce pathological matchings between two time series and distort the true similarity. The constraint warping is also utilized for developing the lower-bounding distance. The experiment showed that the amortized cost for computing DTW on large data sets is linear.
- *Use for our project*: The exact indexing of DTW can be a very useful way to calculate the distance in our clustering algorithm efficiently. However, our input data is somehow different from the normal time series data since our data not only contain states transitions but also have action transition information. We need to figure out a way to adapt this DTW algorithm to our problem settings.
- *Shortcomings*: 1) This paper only considered the case where the two sequences are of the same length, but in our task, the states and action sequences might be of any length, we can re-interpolate the query to any desired length by partitioning the sequence. 2) This algorithm can only index sequences if assumed the warping path is constrained, we need to figure out a good constrain value in the task since it highly relates to indexing accuracy and efficiency.

The third, fourth, and fifth papers were a survey of partitional and hierarchical clustering algorithms by Reddy and Vinzamuri [13] and overviews of algorithms for agglomerative hierarchical clustering by Murtagh and Contreras [8] [9].

- *Main idea*: hierarchical clustering algorithms approach the problem of clustering by developing a binary tree-based data structure called the *dendrogram*. They can be categorized into *agglomerative* and *divisive* clustering methods.
  1) Agglomerative methods start by taking singleton clusters (containing only one data object per cluster) at the bottom level and continue merging two clusters at a time to build a bottom-up hierarchy of the clusters. There are different kinds of agglomerative clustering methods which primarily differ from each other in the similarity measures that they employ, including *single link* (the similarity of two clusters is the similarity between their most similar members, on the other hand, the nearest neighbor), *complete link* (measures the similarity of two clusters as the similarity of their most dissimilar members, which is equivalent to choosing the cluster pair whose merge has the smallest diameter), *group average* (considers the similarity between all pairs of points present in both two clusters), *centroid similarity* (calculates the similarity between two clusters' centroids), and *Ward's criterion* (uses the k-means squared error criterion to determine

the inter-cluster distance, which can be seen as minimum variance). These proximity measures can be expressed in *Lance-Williams Dissimilarity Update Formula*.
2) Divisive methods start with all the data objects in a huge macro-cluster and split it continuously into two groups generating a top-down hierarchy of clusters. It has the advantage of being more efficient compared to agglomerative clustering especially when there is no seed to generate a complete hierarchy all the way down to the individual leaves. The critical factors here including *splitting criterion* (e.g. Ward's k-means square error criterion, Gini index widely used in decision tree construction in classification), *splitting method* (G-means, k-means), *choosing the cluster to split* (e.g. checking the square errors of the clusters and splitting the one with the largest value), and *handling noise* (use a threshold to determine the termination criteria rather splitting the clusters further since noise points might result in aberrant clusters). *Minimum Spanning Tree-Based Clustering* (MSTBC) is an efficient divisive clustering method that removes the largest weighted edge to get two clusterings and subsequently removes the next largest edge to get three clusterings and so on.
3) There are also other methods such as hierarchical self-organizing maps and mixture models, and more recent developments in grid-based or cell-based clustering, focusing on hierarchical density-based approaches. There is also a quadratic computational time hierarchical clustering algorithm that employs the reciprocal nearest neighbor and nearest neighbor chain algorithm, to support building a hierarchical clustering in a more efficient manner compared to the Lance-Williams or basic geometric approaches.

- *Use for our project*: We can try both agglomerative and divisive clustering in our task. Different proximity measures can also be tried or even combined to achieve global similarity and precept the overall structure of the clusters, also to avoid the effects of noise and outliers in the data. For a divisive approach, MSTBC can be a good way to explore the state-action hierarchical clustering with weighted edges.
- *Shortcomings*: Nearly all the hierarchical clustering algorithms use binary tree-based data structure, however in our task, a node in the tree structure might have multiple children and siblings. We have to extend these models to adapt to our data, which will bring a problem of how to keep a good balance between the depth and width of the tree. To solve this, we can design an objective function composed of both two factors.

2.2 Papers read by Sophie Guo

The first paper was the partition-and-group framework for trajectory proposed by Lee [7].

- *Main idea*: The paper proposes an innovative framework that first divides the all the given trajectories into a set of line segments that are separated by some "interesting points", and then group those similar line segments into clusters, as well as calculates a representative trajectory for each of the cluster. The work is based on the intuition that clustering trajectories as a whole, which is the mainstream, might not detect the useful information shown in the similar portions of the input trajectories. The authors perform the trajectory partition task by adopting the widely-used minimum

description length (MDL) principle in information theory to achieve two desirable properties of preciseness and conciseness. The clustering algorithm is built upon the density-based method DBSCAN, which is known for its robustness to noise, and the algorithm can also be extended to support trajectories of various weights. The paper also describes in details how to compute the density of line segments by presenting their distance function, and designs a "trajectory cardinality" to ensure the number of trajectories from which line segments have been extracted from is not too small. Finally, the authors conduct experiments on two real data sets of hurricane track and animal movement and show their results.

- *Use for our project*: Although there is no hierarchy of clustering discussed at all and the trajectories are only for the 2-D physical trajectories in this paper, the algorithms of division and regrouping are very important to our project. It is especially inspiring to check the design of interesting points to segment the whole trajectories. The clustering of segments and calculating the representative of each cluster can form our first layer of grouping the segments, and then we can think of further grouping (maybe recursively) based on it.
- *Shortcomings*: The paper claims the difficulty of measuring the clustering quality varying its parameters, and uses the sum of squared error (SSE) instead. The challenges of selecting parameters also naturally emerges when the algorithm is applied to other tasks. It is also not a hierarchy of clusters but only focus on the segmentation and groupings, however, what if the clusters can be further grouped into hierarchy? i.e. can I compare the clusters of different hurricanes to understand the relations between hurricanes? Moreover, the designed distance is clearly only for the physical trajectories but not for the abstract trajectories (such as in RL) at all, and the performance of them into higher dimension is unknown.

The second paper was the multiple sequence alignment paper by Corpet [3].

- *Main idea*: Corpet presents an algorithm that performs a hierarchical clustering on the sequences that have common segments. Previous to this work, the alignment of two sequences can be performed, but the simultaneous comparison of all the sequences was challenging given lengthy sequences. The author utilizes a matrix of pairwise scores initially, group the sequences that are the closest, and then form a new matrix repeatedly until all the sequences form one cluster. The major advantage is that the sequences are firstly aligned within its family that share common sub-sequences, and the weights of already aligned residues in a family becomes big when families are compared. The best scored pair can thus be given the most significant weight from this algorithm. The experiments is on forming a hierarchical clustering on 39 sequences of cytochrome c, and the result aligns with a previous multiple alignment work.
- *Use for our project*: The well-known algorithm developed from this paper to align sequences in a hierarchical clustering allows us to form a cluster of given trajectories that might contain similar sub goals, as we are especially interested in the similar sub sequences in the trajectories. Although there is a gap between the algorithm and our goal since the paper only builds a hierarchy of all trajectories, we would like to break

the sub trajectories and group a hierarchy of clusters of the sub-goal-directed segments, and thus requires our further thinking if we want to utilize this.

- *Shortcomings*: First, as the paper points out, it seems to be difficult to prove mathematically the convergence of the iterative process though it can always be observed in the treated examples. In addition, the convergence also depends on the distance function, which might vary among different areas, which might lead to unknown difficulties. It is also hard to determine a good criterion for assessing the quality of alignment except for comparing with previous works. However, since it is a classical paper published for years, we might be able to find solutions to cope with these disadvantages.

The third paper was the theoretical paper on discovering objective functions and algorithms by Cohen [2].

- *Main idea*: Compared to the usage in practice, algorithms of hierarchical clustering has received considerably less attentions. The previous work identifies that this lag of theoretical study is partially due to the lack of well-defined objective function, which is the major focus for this paper. The authors first propose the characterization of "good" objective functions, and prove that satisfying the criterion is equivalent to the ground-truth tree in the assumption having strictly optimal cost for an objective function. Referring them to admissible, they prove the classical Dasgupta's objective function is admissible. They then formally proved that it achieves an $O(\sqrt{logn})$ approximation improved from a previous approach as the worst case. The case of dissimilarity-based clustering is also analyzed, and also utilize stochastic models and adversarial models beyond the worst case. They give proofs on algorithms used in practice (the linkage algorithm, the bisection 2-centers, etc.) as well as the algorithm they construct, that they can correctly reconstruct a ground truth tree.
- *Use for our project*: Even though we are not aiming at building a theoretical heavy project but instead more focus on the application/experiment aspect, the proofs and analysis in this paper provide us with algorithms and approaches that have some theoretical support. Using this paper as a start point, we could go further with the algorithms that theoretically guarantee performance, as well as how we can design our objective function.
- *Shortcomings*: Since this is a rather theoretical approach, and as the authors state that they do not consider the case where the noise occurs in the model, but focus more on the formal mathematical algorithms and analysis, the paper provides more authenticity of approaches than utility in the real world. Meanwhile, the paper still presents a variety of related works in applications that we could refer to, and give refined analysis to previous works, too.

The fourth paper was the pedestrian counting paper by Antonini [1].

- *Main idea*: In a video where there are multiple pedestrians whose actions form sequences, it is very possible to overestimate the number of pedestrians in the target detecting process. The authors of the paper try to solve this bias of overestimating

by allowing the redundancy in the detecting step, and reduce the number of pedestrians by clustering similar trajectories. It assumes the same human bodies have similar trajectories and spatial information, and uses this to build a hierarchical clustering. To calculate the distance metric, they use the Hausdorff distance, and to calculate the similarity, they use either the longest common sub-sequence (LCSS) derived from the Levenshtein distance, or the Euclidean distance. They perform two experiments, with the first one comparing the result of a clustering dendrogram with time series and ICA representations.

- *Use for our project*: The major part that we can exploit from this paper is the distance and the similarity matrix. The paper introduces the Hausdorff distance that we might be able to use to compare point sets, while the LCSS can be used to compare the similarity between two strings, which can be utilized by us to compare the trajectories. We might also use similar diagrams of clustering for visualization as well.
- *Shortcomings*: The hierarchical clustering proposed by the authors is for the whole trajectories instead of the segments of the trajectories. This works perfect with this specific task because the pedestrians cannot be separated into two persons. However, many assumptions such as "the length of the trajectories of the same person should be the same"of this approach cannot work if the cluster further aims to group pedestrians of a crowd together, but ideally this clustering intuition should be consistently held throughout the entire process.

The fifth paper was the scalable trajectory prediction framework by Rathore[12].

- *Main idea*: The authors take challenge to build a scalable trajectory prediction framework for large volumes of overlapping trajectories in a dense road network, which cannot be handled by most of the existing trajectory prediction schemes as they could only deal with short-term trajectory prediction. They first cluster the large trajectory data using a modified two-stage clusiVAT (clustering using improved Visual Assessment of Tendency) algorithm that extracts smart sample to serve as a representation of the input cluster structure. Then iVAT(improved visual assessment of cluster tendency) is used to determine the how many clusters there should be, and partition the trajectory samples. Finally all non-sampled trajectories are partitioned into those clusters using the nearest prototype rule (NPR). Experiment is performed on real Singapore taxi data of a month.
- *Use for our project*: The paper handle both the short-term and the long-term trajectory prediction by its major design on first sample trajectories and form clusters, and based on the existing clusters, sample the rest of the trajectories. We can perform the similar procedures to ensure our approach is not limited to the small scale clustering.
- *Shortcomings*: From the experiment, it seems to be applicable for physical trajectory data, but performing on higher dimensional trajectory of data seems to have unknown challenges. In additions, it is only one layer of clustering, which means it never explores the relationship between different clusters.

The sixth paper was the Potter's Wheel paper by Raman[11]. This paper was read specifically for phase 3 because we need its ideas for structure evaluation. We briefly summarize the main ideas of the entire paper, but we focus more on partial contents that are the most related to our project.

- *Main idea*: The paper presents an innovative data cleaning system called Potter's Wheel that ensures the data cleaning process is robust to transformation and discrepancy detection. The structures for data values could be automatically inferred from the user-defined domains, and if there is any constraint violation, it can be detected very easily. Its architecture mainly contains a data source, a transformation engine, an online reorderer, and an automatic discrepancy detector. To extract the structure of a given value during the parsing process from the data source, Potter's Wheel has a method of evaluating how suitable a structure is because it is very likely that multiple structures are all possible. There are three characteristics that the authors want to include: recall, precision, and conciseness. Based on the traditional utilization of Minimum Description Length(MDL) principle, the structure quality metric is defined by a "description length", which encodes a set of values for a given structure. To be more specific, the formula for description length is the sum of the length of theory, and the expected number of bits to encode, which is the weighted sum of space to express a single value. Therefore, the final chosen structure is the one that has the minimum description length, and can handle the balance among maximizing the match rate with the target data (recall), minimizing the match rate with other data (precision), and simplifying the complexity of the structure itself (conciseness).
- *Use for our project*: The paper inspired us to build our own structure evaluation method given the above three important characteristics. Although the paper only specifically deals the structure of regular expression for a plain given string, while we want to handle a more complicated structure measurement, the three criteria are still insightful. We design our own measurement by computing the space of the structure to take, and the expected number of space we expect to store the target values if using this structure. Our methods are explained in more details in the section of Structure Quality Evaluation.
- *Shortcomings*: Again, this paper only presents the evaluation on the structure of plain strings, and thus is not applicable to the more generalized structures, such as the hierarchical ones in our model. It also requires the user-defined domains, leading to a certain amount of knowledge being required by the system in advance. The paper also mentions that it needs to investigate its client interface to see if it is effective. It seems that the system might still make progress on this part as well.

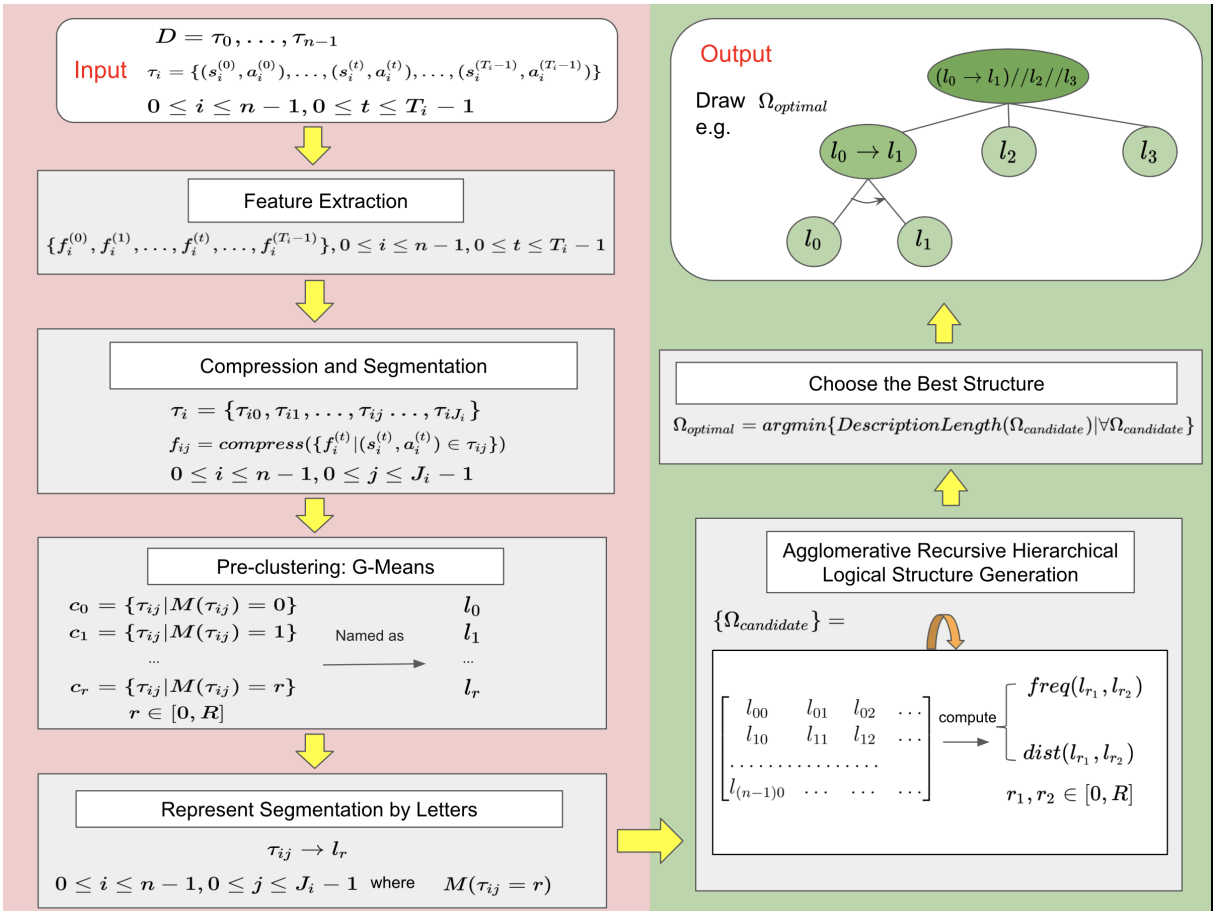

Figure 2: Illustration Diagram for the Model Framework

3 Proposed Method

3.1 Model Framework

Figure 2 briefly shows the entire model framework. The left half deals with the minimum-unit clustering on the given expert trajectories, and then feeds the result of the labeled trajectories, which are simplified to be strings of labels to the right half, to agglomeratively and recursively construct the hierarchical logical structure. We now go through the entire process to illustrate how we generate the hierarchical structure from the beginning to the end. Let's use one scenario to illustrate our Markov Decision Process setting for simplicity.

We want the robot to first navigate to pick up a bucket of water located at position $P_W^{(0)}$, and then carry the water to put out the fire located at position $P_F^{(0)}$. When the robot carries the water, the location of the water moves with that of the robot, and both the water and fire locations become -1 when the fire is put out. Assume the size of the grid world, the initial location of the robot $P_R^{(0)}$, as well as $P_W^{(0)}$ and $P_F^{(0)}$ are all randomly selected, and we have observed n expert demonstrations.

Each demonstration τ_i, $0 \leq i \leq n-1$, is composed of a sequence of state-action pairs $(s_i^{(t)}, a_i^{(t)})$, where each $s_i^{(t)}$ is a tuple of object locations and object conditions at time t, i.e. $s^{(t)} = [P_R^{(t)}, P_{O_1}^{(t)}, P_{O_2}^{(t)}, \ldots, P_{O_w}^{(t)}, M_{O_1}^{(t)}, M_{O_2}^{(t)}, \ldots, M_{O_w}^{(t)}]$ if there are w objects, and we omit i for simplicity. In this water-fire scenario, let $O_1 = W, O_2 = F$, then we have $s^{(t)} = [P_R^{(t)}, P_W^{(t)}, P_F^{(t)}, M_W^{(t)}, M_F^{(t)}]$. We have 4 actions for the robot: move forward, move backward, pick up the water bucket, and pour the water bucket, and has no action when the fire is put out, and we denote $a^{(t)} \in [0, 3]$ accordingly.

In order to understand what is significant, and ignore the extraneous information, we want to extract features for each state-action pair. We denote the features of $(s_i^{(t)}, a_i^{(t)})$ in the ith trajectory as $f_i^{(t)}$, and this feature vector may or may not contain the information of adjacent state-action pairs or that of the whole trajectory according to different scenarios.

After we have extracted the feature vector $f_i^{(t)}$ for the state-action at time t in trajectory i for all t and i, we use them as inputs to compress and segment each expert trajectory into sub-trajectories. Denote the number of segmented sub-trajectories of each trajectory τ_i as J_i, and each segmented sub-trajectory as τ_{ij}. Then it is clear that τ_{ij} is composed of a sequence of state-action pairs, and all τ_{ij} with $0 \leq j \leq J_i - 1$ connected end to end could recover τ_i. For each sub-trajectory τ_{ij}, We further compress the features of all the state-action pairs into f_{ij}, which is the feature vector of τ_{ij}, so that we do not need to keep track of all the features $f_i^{(t)}$ for each single time step.

We then utilize G-Means to cluster the sub-trajectories obtained in the previous step based on the similarity of their compressed features. This section finds out the number of clusters R as well as a mapping function M that maps each sub-trajectory to a cluster c_r, with $0 \leq r \leq R-1$. In the water-fire scenario, we have two clusters: one for the robot to move to pick up a bucket of water, which we can denote by label "W", and the other is for the robot to move to put out the fire, labeled as "F". If we look back to our expert trajectories, it is easy to see they should all have two minimum segments, with the first one belong to the cluster labeled by "W", and the second one belong to the cluster labeled by "F". We thus create the string "WF" for each of the expert trajectories accordingly, and the labels "W" and "F" become leaves of the whole hierarchical structure. Generally, with the strings of labels l_r that represent the expert trajectory segments τ_{ij}, where τ_{ij} is mapped to the cluster c_r, and a set of those possible labels $\{l_r | r \in [0, R]\}$, now we are ready to consider building the true logical hierarchical clustering that is represented on the right half of the framework.

Let's make it clear: up till now we have a number of sequences from the left part, and each sequence is composed of R different clusters. We want to output a hierarchical logical tree structure that best describes the logical relations between clusters in the input sequences. Here we propose an agglomerative recursive hierarchical logical structure generation model that computes two heuristics *position distance* ($dist$) and *order probability* ($freq$) between every two clusters to construct a number of structure candidates $\Omega_{candidate}$. Then we define a metric called *description length* to evaluate the suitability for each structure candidate, and finally output the best structure $\Omega_{optimal}$ that has the minimum description length. The output tree structure and its corresponding string representation will be introduced in detail

in section 3.3.1.

Next, we address each section in the model framework in detail.

3.2 Minimum Unit Clustering

This part corresponds to the left half of the entire framework. As mentioned in the overall framework, it takes a set of expert Markov decision sequence trajectories as inputs, compresses as well as segments them, makes clusters for those minimum-unit segments using G-means, and finally represents those input expert trajectories by a set of strings, with each letter as the name of the cluster that each small segment belongs to respectively. The reason we want to perform the minimum unit clustering is that we want to gather similar subsequences together, instead of dealing with low-level state-action sub-sequences, because we are interested in recovering the general structure of the expert trajectories. It is non-trivial to examine exactly which features are the most important ones, or even find out how to compute the features from raw data in real data, therefore, even though this work might not seem to be significant on synthetic data, we believe this part is non-trivial and makes our whole work holistic.

3.2.1 Feature Extraction

How to recognize and extract good features from the input trajectories is critical to the performance of the clustering model. For now, we design and extract the following features for each state-action pair in the trajectories:

- $start\_state = s^{(t)}$, is the state before taking action $a^{(t)}$.
- $end\_state = s^{(t+1)}$, is the state after taking action $a^{(t)}$.
- $action\_type = a^{(t)}$, represents the type of action $a^{(t)}$.
- $\Delta s^{(t)} = [M_{O_1}^{(t+1)} - M_{O_1}^{(t)}, M_{O_2}^{(t+1)} - M_{O_2}^{(t)}, \ldots, M_{O_w}^{(t+1)} - M_{O_w}^{(t)}]$, is the difference between object conditions in consecutive states.
- $\Delta d^{(t)} = [|P_{O_1}^{(t+1)} - P_R^{(t+1)}| - |P_{O_1}^{(t)} - P_R^{(t)}|, \ldots, |P_{O_w}^{(t+1)} - P_R^{(t+1)}| - |P_{O_w}^{(t)} - P_R^{(t)}|]$, is the difference between Robot-Object distances in consecutive states.

We compute the features for the t-th state-action pair of the i-th trajectory into a feature vector $f_i^{(t)}$. For the current scenario, the above features are sufficient enough for the clustering model (proved in the experiment). To deal with more complicated tasks and scenarios, we can add other features accordingly. Moreover, these features can not only represent a single state-action pair, but also present a sub-trajectory that contains a sequence of consecutive state-action pairs which is used in the following sections.

3.2.2 Compression and Segmentation

Compression From a logical behavior point of view, there are some redundant state-action pairs in the original input trajectories. For example, in the water-fire scenario, in order to get the water, the robot might continue walking east for many steps. These consecutive actions of 'walking east' share exactly the same action and objects' conditions in their feature vectors, and they can be compressed together as one logical unit representing 'walk east'.

For the current scenario, The compression algorithm is to compress consecutive state-action pairs that have 1) no changes in object conditions and 2) exactly the same action, into one sub-trajectory. In the previous section we compute feature vectors for each state-action pair, so here we directly use the feature vectors as input, then the problem is equivalent to compressing redundant consecutive feature vectors into a new feature vector, which is described as:

$$
\begin{aligned}
Input: \quad & Features(\tau_i) = \{f_i^{(0)}, f_i^{(1)}, \ldots, f_i^{(T_i-1)}\}, \quad 0 \le i \le n-1. \\
Output: \quad & \tau_i = \{\tau'_{i1}, \ldots, \tau'_{ij}, \ldots, \tau'_{iJ'_i}\}, \quad 0 \le i \le n-1, \\
& \tau'_{ij} = \{(s_i^{(t_j)}, a_i^{(t_j)}), (s_i^{(t_j+1)}, a_i^{(t_j+1)}), \ldots, (s_i^{(t_{j+1}-1)}, a_i^{(t_{j+1}-1)})\}, \text{ where } 0 \le j \le J'_i \\
& \text{and } a_i^{(t_j)} = a_i^{(t_j+1)} = \ldots = a_i^{(t_{j+1}-1)} \text{ and } \Delta s^{(t_j)} = \Delta s^{(t_j+1)} = \ldots = \Delta s^{(t_{j+1}-2)} = \vec{0},
\end{aligned}
$$

$$
\begin{aligned}
Features(\tau'_{ij}) &= compress(\{f_i^{(t_j)}, f_i^{(t_j+1)}, \ldots, f_i^{(t_{j+1}-1)}\}) \\
&= \{start\_state = s^{(t_j)}, end\_state = s^{(t_{j+1})}, action\_type = a^{(t_j)}, \\
&\quad \Delta s = \Delta s^{(t_{j+1}-1)} - \Delta s^{(t_j)}, \Delta d = \Delta d^{(t_{j+1}-1)} - \Delta d^{(t_j)}\}.
\end{aligned}
$$

Notice there are J'_i sub-trajectories in τ_i after compression, each sub-trajectory starts at time stamp t_j and ends before the next sub-trajectory starts at t_{j+1}. The compression step is efficient for removing redundant data. In more complicated scenarios, we can further modify the compression algorithm to adapt to the task settings accordingly and even reduce the noises in the original input data when the expert data is not perfect.

Segmentation After the compression, there can still be too many sub-trajectories for clustering. Also, considering the goal of a hierarchical logical structure, these compressed sub-trajectories might be too small to represent the minimum unit of a logical and semantic function. Thus, we design a segmentation (partition) algorithm to segment each trajectory based on the previous compressed results. For now, the segmentation algorithm is based on Δs in the feature vector, which means that the object conditions do not change within each segment. For example, in the water-fire scenario, after compression, there might be two consecutive sub-trajectories whose semantic meanings are 'walking east' and 'grabbing the water'. However, 'walking east' has no change of object conditions, so it will be combined with its consecutive sub-trajectory 'grabbing the water', and together form a segment 'getting the water' with condition changed in object 'Water'. We can upgrade and adapt the current segmentation algorithm to other tasks and scenarios in the future. The output of the segmentation stage is shown in Figure 2.

3.2.3 Pre-Clustering: G-Means

We first cluster the minimum unit of sub-trajectories obtained in the section of Compression and Segmentation as leaves of the tree-like hierarchical structure. This can be considered as the lowest level clustering of sub-trajectories. This part is modified from the traditional G-means clustering method, but with each feature as a vector, not a single number.

Let the segmented sub-trajectory τ_{ij}, i.e. the jth segment of the ith trajectory, have m features, namely $f_{ij1}, f_{ij2}...f_{ijm}$, and each feature f_{ijk}, where $1 \leq k \leq m$, is a vector. We want to find an assignment that maps each τ_{ij} to a cluster c_r, where $1 \leq r \leq R$ and R is the number of clusters. Similar to K-Means but does not have a known number of clusters R, G-Means determines R by initially assuming a single cluster, and then making split testings step by step. The algorithm is shown in Algorithm 1.

input : m features f_{ijk}, where $1 \leq k \leq m$, for all the sub-trajectories τ_{ij}.
output: 1) R, the number of clusters, and 2) a mapping M from τ_{ij} to r, meaning τ_{ij} belongs to cluster c_r, $1 \leq r \leq R$.

Initialize: $R = 1$, *and* $M(\tau_{ij}) = 1$ *for all* i, j.
while *can_split*(c_r) == *True for any* r **do**
  for *each splittable cluster* c_r **do** `// K-Means with` $k = 2$
    pick 2 random τ_{ij} in c_r, and let features f_{ij} as new centroids cen_{r1} and cen_{r2}
    R ++
    while Δ*splitting result* > *threshold* **do**
      for *each* τ_{ij} *assigned to* c_r **do** `// re-assign all` τ_{ij} `in` c_r
        $d_1 = \text{dist}(cen_{r1}, f_{ij})$
        $d_2 = \text{dist}(cen_{r2}, f_{ij})$
        $M(\tau_{ij}) = argmin(d_1, d_2)$
      end
      for cen_{r1} *and* cen_{r2} **do** `// re-calculate the two centorids`
        for *each feature* k **do**
          $cen_{r1k} = \text{mean}(\{f_{ijk} | M(\tau_{ij}) = r_1\})$
          $cen_{r2k} = \text{mean}(\{f_{ijk} | M(\tau_{ij}) = r_2\})$
        end
      end
    end
    Update $\{c_r | 1 \leq r \leq R\}$
  end
end

Algorithm 1: G-Means for Sub-trajectories

In this algorithm, there are two parts that especially require attention (and possibly modifications in phase 3: the distance function and the splitting criteria.

First is the distance function that calculates the distance between the sub-trajectory and the centroids. Recall that each feature is a vector, and we need a function that calculates the

single distance for each feature, so that finally we can sum up all the distances of different features as the entire distance between a single sub-trajectory and a centroid. The function naively calculates the Euclidean distance between two input vectors, and is normalized by the standard deviation of all elements in the cluster for each feature. If the two vectors have different lengths, we align them by the ends first, with a slight penalty added to the distance, and then calculate the distance accordingly.

The second is the splitting criteria that determines if a single cluster should be further split into more clusters. Traditionally G-means splits a single cluster when it considers the cluster non-Gaussian. It has a "significance parameter" for how strict the Anderson-Darling test is in judging whether clusters are Gaussian or not. Besides, there is a parameter requiring a minimum of elements in each cluster, to ensure the clusters we generate capture the major difference in data.

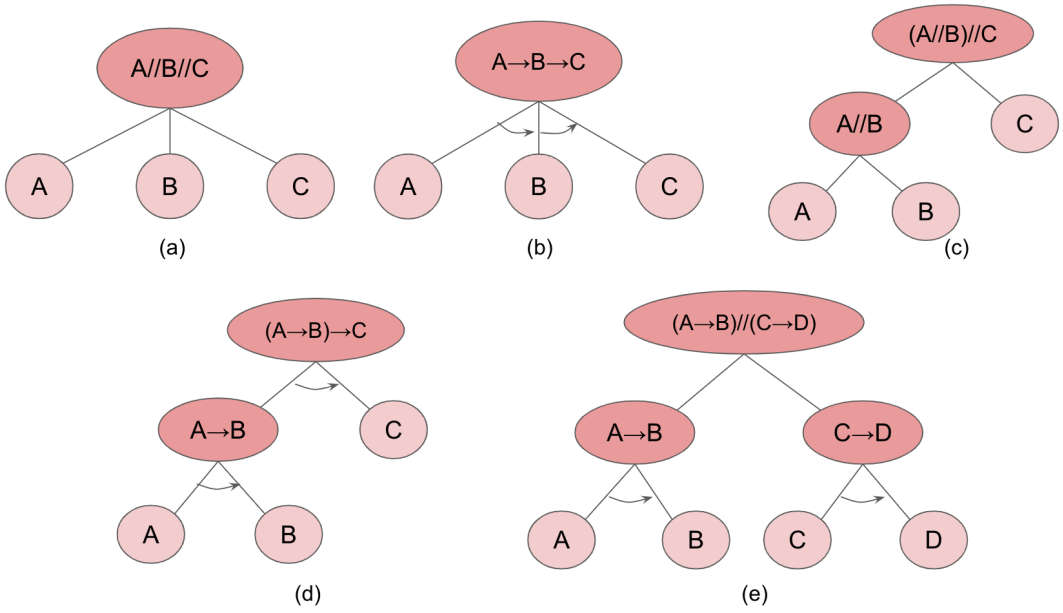

Figure 3: Hierarchical logical structure examples, each node is written with the string representation of its sub-tree structure. **(a)** Basic relation *parallel*; here A, B and C are parallel. **(b)** Basic relation *sequential*; ABC are in sequential order. **(c)** A and B are parallel and they always appear together, the result of A and B is parallel with C; this is different from structure (a). **(d)** Emphasizing that A and B always appear together is not necessary for sequential relations; it is exactly the same structure as (b). **(e)** A to B, and C to D are both in sequential order, while their sequential results are parallel.

3.3 Hierarchical Logical Structure

Given a number of input sequences, each sequence is composed of the same number of different clusters, e.g. {ABCD, CDAB}; we output a hierarchical logical structure that best

describes the given data, e.g. Figure 3(e) for input data {ABCD, CDAB}. Meanwhile, we also developed algorithms that can generate data according to a given structure. In this section, we will discuss the definition and representation of this proposed hierarchical logical structure, the structure quality evaluation method, and the structure generation model in detail.

3.3.1 Definition and Representation

We define a logical tree structure to represent the hierarchical logical relations among the given input data. As shown in Figure 3, there are two types of fundamental logical relations among siblings: *parallel* (//) and *sequential* ($\rightarrow$). Parallel means the sibling clusters are independent of each other, and they might happen in any order. For example, imagine humans putting on shoes, the actions of "get the left shoe" (represented as "L") and those of "get the right shoe" (represented as "R") are independent. You can get the left shoe first and then get the right shoe, or in the reverse order. Thus, L and R are parallel, and the input data will contain both "LR" and "RL" sub-sequences. On the other hand, sequential means the sibling clusters can only happen in sequence. For example, in the water-fire scenario, we can only put out the fire ("F") as long as we already get the water ("W"). "W" always happens before "F" in the data, so they are in a sequential relation.

We define some basic rules about this logical tree structure:

1) For each tree structure, there is a one-to-one corresponding string representation (shown in the node text of Figure 3). We use "$\rightarrow$" to represent sequential relation and "//" for parallel relation. Parentheses represent subtree structure, which means nodes within corresponding parentheses are in the same subtree. For example in Figure 3(e), $(A \rightarrow B)$ means A and B are sequential and they are in one subtree, and the whole tree can be represented using string "$(A \rightarrow B)//(C \rightarrow D)$".

2) Siblings always appear together in the data. For example, in Figure 3(c), A and B are siblings in the subtree "A//B", and data "ACB" is not a valid matching for this structure since A and B should appear consecutively, while "BAC" is valid for this case. Notice that Figure 3(a) is a valid structure for "ACB" since A, B and C are all siblings. However, in sequential relations, for example Figure 3(d), emphasizing that A and B should appear together is meaningless, because it is representing exactly the same data with Figure 3(b). Figure 3(d) contains redundant information and takes more bits to encode. In the following sections, we will show how to use Description Length and heuristics to evaluate the suitability of different structures.

3) Siblings can only have one type of relationship with each other. For example, structure "A//B$\rightarrow$C" might refer to different structures "(A//B) $\rightarrow$C" and "A//(B$\rightarrow$C)". Because different types of relations between siblings always cause ambiguity in structure interpretation, we consider them invalid structures.

4) A logical tree structure has a finite number of matching sequences. For example, in Figure 3(e), only "ABCD" and "CDAB" are valid matches to the structure.

Given a logical tree structure, we can generate all its matching sequences. On the other hand, given a number of input sequences, there might not exist a tree structure in our

definition that perfectly matches all the sequences. Thus, we need to define metrics to measure the suitability of a structure given input sequences, and use it to pick up the best structure. This intuition leads to the definition of Description Length in the next section.

3.3.2 Structure Quality Evaluation

As in the survey section, we review the usage of description length in [11] that evaluates the suitability of a structure given a single plain string, here we present our design on evaluating the hierarchical structure by calculating our modified version of description length. It captures the three characteristics brought up by the paper: recall, precision, and conciseness.

Similar to the description length in [11], for any structure Ω, the description length of it is defined as $DL(\Omega) = w_{space} * p * log(m) + (1-f) * (log|W|) + f * (log|M|)$, where w_{space} is a weighted constant, p is the length of the structure expression, m is the size of the alphabet for expressing the structure, f is the matching rate for this structure with the target data, $|W|$ is the size of the possible values in the subset data set that does not match with the structure (so we have to encode them directly), and $|M|$ is the size of the possible values in the data set that matches with the structure.

The three terms in the formula represent the penalties for conciseness, recall, and precision respectively that are similar to [11], but differ by giving weights to the hierarchical and relation representation as well. Specifically, the first term measures how much space to store the structure, the second measures how much space to store the data that does not match with the structure, and the third measures how much space to store the matched data.

We want to explain this formula with a toy example: assume we have one candidate structure Ω_1 ="(A//B)→C" that we want to measure its goodness, and the target data we want to match with is given by "BCA, BAC, ABC, BAC, BAC, CAB, BAC, ABC, BAC, BAC". Intuitively, we want to know how accurate this candidate structure is when it is matched with our given data (recall), how they work with other values (precision), and how much space these structures take to store (conciseness).

Notice that in order to calculate the space to store this structure, we have to first find out the alphabet we use to build any of the possible structures. There are three letters needed in this example, two possible relations, and any pairs of parenthesis can represent a sub-tree structure without ambiguity, so our alphabet size is $3+2+2 = 7$ for any candidate structure. The length of the structure "(A//B)→C" contains exactly one character of each type, so its length is 7, and to encode it takes $7log(7) = 19.6$ bits.

For the structure Ω_1 ="(A//B)→C", 8 out of the 10 given data points match with it, so the match rate is 0.8. If we further set the w_{space} to be 0.1, $DL(\Omega_1) = 1.96 + 0.2 * (log(2)) + 0.8 * (log(2)) = 2.96$. It is non-trivial to adjust the weight parameters, and for the experiment's convenience, we have not explored various parameters yet in this course project, but it should give us interesting results.

3.3.3 Agglomerative Hierarchical Logical Structure Generation with Heuristics and Description Length

As shown in Figure 2, given input sequences, we first agglomeratively construct a number of hierarchical logical structure candidates using two heuristics (*position distance* and *order probability*) in a recursive manner, then use description length to choose the best structure from all the generated candidates.

Position Distance is the average distance between two clusters' positions in the given sequences. For example, given data "ABCD" and "CDAB", the position index of A in "ABCD" is 1 and the position index of D in "ABCD" is 4, so the position distance between A and D in "ABCD" is $|4-1|=3$, similarly in "CDAB" is $|2-3|=1$, so the overall position distance for A and D is $(1+3)/2=2$. Given n input sequences indexed from 0 to $n-1$, and R clusters indexed from 0 to $R-1$, let l_i and l_j be two different clusters, $position_k(l_i)$ represents the position index of cluster l_i in the k-th sequence, we compute the position distance between l_i and l_j as:

$$dist(l_i, l_j) = \frac{\sum_{k=0}^{n-1} |position_k(l_i) - position_k(l_j)|}{n}, 0 \le i < j < R-1.$$

Position distance is used to illustrate how close the two clusters are in the given sequences. In most of the cases, the closer they are, the shorter their distance in the tree structure.

Order Probability is the probability that one cluster appears before another cluster in the given sequences. For example, given data "ABCD" and "CDAB", A appears before D in "ABCD" and A appears after D in "CDAB", so the order probability of clusters A and D is 0.5. Given n input sequences indexed from 0 to $n-1$, and R clusters indexed from 0 to $R-1$, let l_i and l_j be two different clusters, $position_k(l_i)$ represents the position index of cluster l_i in the k-th sequence, we compute the order probability between l_i and l_j as:

$$\begin{aligned} freq(l_i, l_j) &= \frac{\sum_{k=0}^{n-1} \mathbb{1}[position_k(l_i) < position_k(l_j)]}{n}, \\ freq(l_j, l_i) &= 1 - freq(l_i, l_j), \quad 0 \le i < j < R-1. \end{aligned}$$

Order probability is used to determine the type of relations between two clusters. For example, in sequential relations, the order probability is always 0 or 1; when there are noises in the input sequences, it will be in a small range around 0 or 1. Here we set a $relation\_threshold$ to recognize relation types. If $freq(l_i, l_j) < relation\_threshold$ or $freq(l_i, l_j) > 1 - relation\_threshold$, then the relation is $l_i \to l_j$ or $l_j \to l_i$, otherwise $l_i // l_j$.

Algorithm Description Given input of n sequences composed of R clusters, we output the best hierarchical structure that describes the input data. The algorithm follows:

- Step 0. Initialization. Treat each cluster in the given sequences as a leaf node in the tree structure.

- Step 1. If every given sequence only has one same cluster, then it means we finished constructing one structure candidate and its root node is the only cluster left in the sequence. Add this newly constructed structure to the candidates set Ω and stop this branch.

- Step 2. For every two cluster pairs, compute their heuristics in the sequences: position distance ($dist$) and order probability ($freq$). Determine their relation types according to their order probability.

- Step 3. Select the maximum number of clusters that all 1) have the minimum position distance among all cluster pairs; 2) share the same type of relation between each other. Merge the selected clusters to a new cluster node in the tree according to their relation type, where the originally selected cluster nodes are children of the new node. Notice that we might have multiple merge choices at this step: different groups of clusters all have minimum position distance, but we only merge one of them at a time. This ambiguity might lead to different recursive branches and construct different structure candidates.

- Step 4. According to the merge results in Step 3, modify the input sequences: replace the original cluster sub-sequences with the newly merged cluster representation. Then go to Step 1.

- Step 5. After finishing all the recursive branches, we calculate the description length for all the structure candidates in Ω, and choose the one with the minimum description length as the best structure to output.

Table 1 shows an example to illustrate the above algorithm. Given input {ABCD, CDAB}, we compute heuristics for each pair of clusters, and find that there are two groups of clusters {A, B} and {C, D} that both have the minimum position distance 1.0 and share the same relation type within each group. If we choose {A, B} to merge at this step, the new cluster node will be A→B, and the input sequences will change to {(A→B)CD, CD(A→B)}. After several steps of merging, we will have a number of structure candidates, and the resulting structure with the minimum description length is shown in Figure 3(e).

Cluster	Cluster	Position Distance	Order Probability	Relation Type
A	B	1.0	1.0	sequential
A	C	2.0	0.5	parallel
B	C	2.0	0.5	sequential
B	D	2.0	0.5	parallel
C	D	1.0	1.0	sequential

Table 1: Example of Position Distance and Order Probability from input {ABCD, CDAB}.

Since there might be noises in the input sequences, we define a *distance_threshold* to select the merging clusters within a small range around the minimum position distance in Step 3. Actually, we could use description length to select the merging clusters instead of setting thresholds, or even solve this problem by enumerating all the possible structures using description length without considering any heuristics; however, this will bring a huge increase in both time and space complexity especially with long sequences.

4 Experiments

4.1 Part I: Test on Single Relation

We first generate synthetic data on two very simple 1-D grid world examples to test our clustering methods on single relations of parallel or sequential.

1) Sequential: we first test on the water-fire scenario as we mentioned earlier for illustration in the section of Model Framework. Recall that we have the expert robot to first navigate to get a bucket of water at position $P_w^{(0)}$, and then navigate to the fire at position $P_F^{(0)}$ to put it out. The expert is originally at position $P_R^{(0)}$, and after it picks the water when it reaches $P_w^{(0)}$, the water position moves with the expert, i.e. $P_W^{(t)} = P_R^{(t)}$ for every time step t afterwards. Finally, after both the expert and the water navigate to $P_F^{(0)}$, the expert puts out the fire, which makes both the water and fire locations to be -1 to denote "non-existence". The number of grids in this grid world is randomly determined between 5 and 10, and the initial locations of water, fire, and expert are selected uniformly random.

The state is a tuple of (robot position, water position, fire position, does the robot have water now, has the fire been put out yet), i.e. $s^{(t)} = [P_R^{(t)}, P_W^{(t)}, P_F^{(t)}, M_W^{(t)}, M_F^{(t)}]$, at time step t. Note that the last two elements in each state are binary numbers that we can make from observation. The action $a^{(t)}$ is an integer, representing moving forward, moving backward, picking up a bucket of water, and dropping water to put out the fire, at time step t. We also have a 20% probability for the expert to fail to execute the correct navigation action. For each test, we generate 20 trajectories, with each trajectory τ_i as a sequence of $(s_i^{(t)}, a_i^{(t)})$, varied in length T_i, as it takes different time to complete each different task randomly generated. These 20 sequences of state-action pairs become our input in one single test, and we are able to successfully output the clusters of moving to get water and moving to put out the fire, as well as tell the relation to be sequential as the result.

2) Parallel: we then test on another left-right-shoe scenario where we want the expert robot to wear its left and right shoes on. This differs from the water-fire scenario in that the robot can either first navigate to the location of the left shoe to wear the left shoe on, or do that first for the right shoe instead, whereas, in the water-fire scenario, the robot has to first pick up a bucket of water before it navigates to the fire location. By introducing these two different scenarios, we want to explore the easy tasks that contain either a pure sequential relation, or a parallel relation. Building upon that, we can have more complicated scenarios in Part III.

The data generated for this left-right-shoe scenario is similar to the water-fire scenario.

The size of the gird world is randomly determined between 5 and 10, and the initial locations of the robot $P_R^{(0)}$, the left shoe $P_{left}^{(0)}$, the right shoe $P_{right}^{(0)}$, are all selected uniformly random. The robot is free to first navigate to $P_{left}^{(0)}$, wear the left shoe on, and then navigate to $P_{right}^{(0)}$ to wear the right shoe on, or do the opposite. Whenever the robot wears a shoe, the location of the shoe becomes the same as that of the robot for all the time steps afterward, and the task is finished when the robot successfully has both shoes on. The robot cannot wear two shoes at the same time, so if the two shoes happen to be located at the same position initially, the robot navigates to this location, randomly wearing one first, and then wearing the other.

The state is a tuple of (robot position, left shoe position, right shoe position, does robot have left shoe on, does robot have right shoe on), i.e. $s^{(t)} = [P_R^{(t)}, P_{left}^{(t)}, P_{right}^{(t)}, M_{left}^{(t)}, M_{right}^{(t)}]$, at time step t. The action $a^{(t)}$ is an integer, representing moving forward, moving backward, wearing the left shoe, and wearing the right shoe, at time step t. The probability for the expert to fail to execute the correct navigation action is 20%. For each test, we generate 20 trajectories, as same as the water-fire scenario.

Similar to the result of the experiment on the sequential relation, we are able to successfully output the clusters of moving to pick up the left shoe and moving to pick up the right shoe, and at the same time tell the relation to be parallel.

4.2 Part II: Test on Hierarchical Logical Structure Generation

We first wrote a program to generate all possible matching sequences from a given structure in string representation, and use this program to generate the testing data. To test the performance of our hierarchical logical structure generation model, we simply compare if the model output structures in string representation are the same as the given structures for the data generation program.

We generated three categories of testing data in the experiment: 1) Clean data, uniformly contains all the matching sequences of the desired structure. 2) Non-uniform data. All the sequences in the data matched the desired structure, but in a non-uniform distribution. 3) Non-uniform data with error noise. Besides the non-uniform distribution of matching sequences, there are also sequences that is invalid for the desired structure; for example, in a desired structure "A→B", the noise data will contain a small proportion of sequence BA.

Each of the three categories has five possible lengths of the input sequence (number of clusters for each sequence): 2, 3, 4, 5, 6. For each testing case, the number of input sequences ranges from 1 to 6000. It took about one second to run the test on a MacBook Pro laptop with 2.6 GHz Intel Core i5 and 8 GB 1600 MHz DDR3. Table 2 shows the results of the experiment with $relation_threshold = 0.2, distance_threshold = 0.2$. From the results, we find that our model can perfectly deal with the clean and non-uniform data, and it only had one wrong case in the non-uniform with error noise data. We looked at the wrong case: the standard output is "(A → (B // (C → D)) → E) // F" with DL = 58.41, and our model output is "((A → C → D) → B → E) // F" with DL = 51.02. It is very interesting that our model output has a better DL than the standard one, because the standard structure is

more complicated even though it matches more input sequences. Thus, there is a tradeoff here between choosing structures that match most of the sequences in the data and choosing structures that do not overfit the data. We can explore more into this issue of description length in future work.

Dataset	#Correct Cases	#Wrong Cases	Accuracy
Clean	73	0	100.00%
Non-uniform	73	0	100.00%
Non-uniform with error noise	72	1	98.63%

Table 2: Results on Hierarchical Logical Structure Experiment.

4.3 Part III: Integrated Test

With the success of testing on single relation and hierarchical logical structure generation, we are ready to perform an integrated test for the whole algorithm with the following scenario: similar to the water-fire scenario, the robot needs to first get a bucket of water, and then carry it to the fire location to put out the fire, but this time the fire is in a locked room that the robot originally does not have the key to open with. Besides the requirement of acquiring the key before opening the door, the water location might be in the same room where the robot starts, or it might be in the locked room, which leads to different behaviors in different settings.

1) K $\rightarrow$ D $\rightarrow$ W $\rightarrow$ F: we first test the scenario that the water is in the locked room along with the fire. We have the expert robot to first navigate to get the key at position $P_K^{(0)}$, then navigate to open the door located at position $P_D^{(0)}$ with its acquired key. After the door is open, the robot is allowed to get in the unlocked room to navigate to a bucket of water at position $P_W^{(0)}$, and then navigate to the fire at position $P_F^{(0)}$ to put it out. The expert is originally at position $P_R^{(0)}$, and after it picks the key or the water when it reaches $P_K^{(0)}$ or $P_W^{(0)}$, the key or water position moves with the expert, i.e. $P_K^{(t)} = P_R^{(t)}$ or $P_W^{(t)} = P_R^{(t)}$ for every time step t afterward. When the robot opens the door, the key is eternally left on the door, and does not move with the robot anymore, i.e. $P_K^{(t)} = P_D^{(t)}$. Finally after both the expert and the water navigate to $P_F^{(0)}$, the expert puts out the fire, which makes both the water and fire locations to be -1 to denote "non-existence". The number of grids in this grid world is randomly determined between 10 and 15, and the initial locations of key, door, water, fire, and expert are selected randomly. Indeed, we first select the door position randomly, and then based on it randomly select the location of the other objects to make sure the key and robot are in one room, and the water and fire are in the other.

The state is a tuple of (robot position, key position, door position, water position, fire position, does the robot have the key now, is the door open now, does the robot have water now, has the fire been put out yet), i.e. $s^{(t)} = [P_R^{(t)}, P_K^{(t)}, P_D^{(t)}, P_W^{(t)}, P_F^{(t)}, M_K^{(t)}, M_D^{(t)}, M_W^{(t)}, M_F^{(t)}]$, at time step t. Note that the last four elements in each state are binary numbers that we can

make from observation. The action $a^{(t)}$ is an integer, representing moving forward, moving backward, picking up a key or a bucket of water, dropping water to put out the fire, and unlocking the door at time step t. For each test, we generate 20 trajectories, with each trajectory τ_i as a sequence of $(s_i^{(t)}, a_i^{(t)})$, varied in length T_i, as it takes different time to complete each different task randomly generated. These 20 sequences of state-action pairs become our input in one single test, and we are able to successfully output the clusters of navigating to get the key, navigating to unlock the door, navigating to get the water, and navigating to put out the fire, as well as successfully discover the relation of K $\rightarrow$ D $\rightarrow$ W $\rightarrow$ F as the result.

2) (W // K) $\rightarrow$ D $\rightarrow$ F: we then test the scenario that the water is in the room that the robot starts with. Therefore, we have the expert robot either first navigate to get the key at position $P_K^{(0)}$, or navigate to a bucket of water at position $P_W^{(0)}$. When it has both the water and key in hand, it then navigates to open the door located at position $P_D^{(0)}$ with its key. After the door is open, the robot then navigates to the fire at position $P_F^{(0)}$ to put it out. The size of the grid world, the dynamics, and the generated locations on key and door, water, and fire are the same as in the first scenario.

It is clear that navigating to acquire the water and key forms a parallel relation, and together they form a group to have the sequential relations with navigating to open the door and put out the fire. The states, actions, and number of generated trajectories are all the same as the first scenario, and we can successfully detect the relation of this two-layered (W // K) $\rightarrow$ D $\rightarrow$ F as the result.

5 Conclusions

We have developed a holistic system to perform hierarchical logical clustering for state-action sequences in the Markov Decision Process. In the first part, we focus on making minimum-unit sub-trajectory clusters for expert demonstration trajectories. Through the process of feature extraction, compressing, segments into minimum units, and clustering by G-means, we are able to gather similar minimum-unit sub-trajectories together, and simplify the expert demonstrations to be a set of strings of label letters. In the second part, by utilizing the heuristics on ordering probability and position distance, we construct the hierarchical logical structure generation model recursively with the simplified expert labels. We further use the description length to select the optimal structure from candidates and finally output the hierarchical structure.

We have performed three tests: one on the simple relation, one on the large amount of structure generation, and one on the holistic system. The success of our test justifies our algorithms, and we can claim that we are able to better understand the logic among sub-tasks given the expert performance, and our work leads to a better learning foundation in the future.

Advantages (list of innovations)

- it provides a holistic system for hierarchical logical clustering for Markov decision process expert data
- it reveals a clear organizational structure of sub-goals within a task
- it proposes a feasible hierarchical logical structure representation as well as its evaluation method for each structure via description length
- it provides a method to generate hierarchical data and recover the hierarchical structure (robust to noise and error)
- its accuracy is very good for synthetic data
- it can be generalized to any area that is applicable in the Markov Decision Process
- it helps Inverse Reinforcement Learning have a clearer objective when learning composite behaviors from experts.

Future work and discussion

- We can further extend our system, especially the definition and representation of the logical structure in order to include other types of logical relations, such as *optional* (means only one of the sibling clusters will appear in the data sequence), *repeating* (means the same cluster might appear multiple times in one input sequence), *topological* (the hierarchical cluster does not have to form a tree structure).
- more complex integrated data, real data, that includes more complicated tasks, logical structures, and noise or errors.
- allows each expert trajectory to contain sub-trajectories that belong to the same cluster. In that case, we release the assumption that the expert always finishes the current sub-goal before start executing a new one.

References

[1] Gianluca Antonini and Jean-Philippe Thiran. Counting pedestrians in video sequences using trajectory clustering. *IEEE Transactions on Circuits and Systems for Video Technology*, 16(8):1008–1020, 2006.

[2] Vincent Cohen-Addad, Varun Kanade, Frederik Mallmann-Trenn, and Claire Mathieu. Hierarchical clustering: Objective functions and algorithms. *Journal of the ACM (JACM)*, 66(4):26, 2019.

[3] Florence Corpet. Multiple sequence alignment with hierarchical clustering. *Nucleic acids research*, 16(22):10881–10890, 1988.

[4] Greg Hamerly and Charles Elkan. Learning the k in k-means. In *Advances in neural information processing systems*, pages 281–288, 2004.

[5] Eamonn Keogh and Chotirat Ann Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and information systems*, 7(3):358–386, 2005.

[6] Sanjay Krishnan, Animesh Garg, Richard Liaw, Brijen Thananjeyan, Lauren Miller, Florian T Pokorny, and Ken Goldberg. Swirl: A sequential windowed inverse reinforcement learning algorithm for robot tasks with delayed rewards. *The International Journal of Robotics Research*, 38(2-3):126–145, 2019.

[7] Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. Trajectory clustering: a partition-and-group framework. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 593–604. ACM, 2007.

[8] Fionn Murtagh and Pedro Contreras. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):86–97, 2012.

[9] Fionn Murtagh and Pedro Contreras. Algorithms for hierarchical clustering: an overview, ii. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 7(6):e1219, 2017.

[10] Quoc Phong Nguyen, Bryan Kian Hsiang Low, and Patrick Jaillet. Inverse reinforcement learning with locally consistent reward functions. In *Advances in neural information processing systems*, pages 1747–1755, 2015.

[11] Vijayshankar Raman and Joseph M Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, volume 1, pages 381–390, 2001.

[12] Punit Rathore, Dheeraj Kumar, Sutharshan Rajasegarar, Marimuthu Palaniswami, and James C Bezdek. A scalable framework for trajectory prediction. *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[13] Chandan K Reddy and Bhanukiran Vinzamuri. A survey of partitional and hierarchical clustering algorithms. In *Data Clustering*, pages 87–110. Chapman and Hall/CRC, 2018.

[14] Adrian Šošić, Abdelhak M Zoubir, and Heinz Koeppl. Inverse reinforcement learning via nonparametric subgoal modeling. In *2018 AAAI Spring Symposium Series*, 2018.

[15] Liting Sun, Wei Zhan, and Masayoshi Tomizuka. Probabilistic prediction of interactive driving behavior via hierarchical inverse reinforcement learning. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2111–2117. IEEE, 2018.

[16] Monica C Vroman. *Maximum likelihood inverse reinforcement learning*. PhD thesis, Rutgers University-Graduate School-New Brunswick, 2014.

A Appendix

A.1 Labor Division

The team performed the following tasks

- Write the paper survey and discuss on model design [Sophie, Shuqi] - finished
- Build state-action scenario environments [Sophie] - finished
- Generate state-action synthetic data [Sophie] - finished
- Extract features of state-action pairs [Shuqi] - finished
- Make compression and segmentation algorithm for state-action pairs [Shuqi] - finished
- Implement adaptive G-means clustering [Sophie] - finished
- Wrote algorithm to distinguish simple relations in Part I [Shuqi] - finished
- Test simple relations (Part I) with noise [Sophie] - finished
- Agglomerative Hierarchical Logical Structure Generation Model and representations [Shuqi] - finished
- Design and implement the structure evaluation via description length [Sophie] - finished
- Generate all possible matching sequences given input tree structure [Shuqi] - finished
- Generate three types of hierarchical structure synthetic data (Part II) [Shuqi] - finished
- Test and Evaluate the hierarchical structure generation model (Part II) [Shuqi] - finished
- Combine codes and make an integrated test (Part III)[Sophie] - finished
- Combine codes, clean up, and make the distributed version ready [Sophie and Shuqi] - finished
- Paper writing [Sophie, Shuqi] - finished
- Prepare for the poster and presentation [Sophie, Shuqi] - 1 week

A.2 Full disclosure wrt dissertations/projects

Sophie: Her thesis focuses on the intersection of social norms and inverse reinforcement learning (IRL). Although related to the project, she has never learned anything about hierarchical clustering at all, while this project purely focuses on the hierarchical clustering aspect and is not related to IRL, so she studied to implement it in this project. She also considered finishing up the project by adding IRL algorithms to see if the trajectories in clusters can recover sub-goals in the reward functions, and this might lead to a publication in her area.

Shuqi: Her research lies in the area of computer music, especially automatic music generation. She is very interested in music pattern discovery and wants to find a good way in data mining that can help recognize hierarchical music structure information, which is supercritical and remains unsolved in music generation. The model in this paper can be further generalized to help solve this music problem.

Contents