Generic Itemset Mining Based on Reinforcement Learning

KAZUMA FUJIOKA[1] **AND KIMIAKI SHIRAHAMA**[2,3]
[1]Graduate School of Science and Engineering, Kindai University, Higashiosaka, Osaka 577-8502, Japan
[2]Department of Informatics, Kindai University, Higashiosaka, Osaka 577-8502, Japan
[3]Cyber Informatics Research Institute, Kindai University, Higashiosaka, Osaka 577-8502, Japan

Corresponding author: Kimiaki Shirahama (shirahama@info.kindai.ac.jp)

ABSTRACT One of the biggest problems in itemset mining is the requirement of developing a data structure or algorithm, every time a user wants to extract a different type of itemsets. To overcome this, we propose a method, called *Generic Itemset Mining based on Reinforcement Learning* (GIM-RL), that offers a unified framework to train an agent for extracting any type of itemsets. In GIM-RL, the environment formulates iterative steps of extracting a target type of itemsets from a dataset. At each step, an agent performs an action to add or remove an item to or from the current itemset, and then obtains from the environment a reward that represents how relevant the itemset resulting from the action is to the target type. Through numerous trial-and-error steps where various rewards are obtained by diverse actions, the agent is trained to maximise cumulative rewards so that it acquires the optimal action policy for forming as many itemsets of the target type as possible. In this framework, an agent for extracting any type of itemsets can be trained as long as a reward suitable for the type can be defined. The extensive experiments on mining high utility itemsets, frequent itemsets and association rules show the general effectiveness and one remarkable potential (agent transfer) of GIM-RL. We hope that GIM-RL opens a new research direction towards learning-based itemset mining.

INDEX TERMS Data mining, itemset mining, knowledge discovery, reinforcement learning.

I. INTRODUCTION

Much research effort has been made on itemset mining that aims to discover interesting relations among items in a large-scale dataset [1]. The dataset consists of a large number of transactions each of which contains different items. From such a dataset, the goal of itemset mining is to extract interesting sets of items (i.e., itemsets) in terms of a user-specified interestingness measure [2]. Depending on the user's needs, different interestingness measures can be chosen to extract, for instance, *Frequent Itemsets* (FIs) consisting of frequently co-occurring items, *Association Rules* (ARs) comprised of correlated items, *High Utility Itemsets* (HUIs) formed by highly profitable items, and so on. From a more general perspective, items in a transaction can be viewed as attributes in one data instance like image, sensor recording and amino acid set. Thus, starting from customer transaction analysis [3],

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

itemset mining is utilised in various applications such as image classification, healthcare, bioinformatics and so on [1].

Itemset mining is difficult because of the huge search space. Assuming that there are M distinct items in a dataset, the search space is defined as the set of all the possible $2^M - 1$ itemsets. The naive approach to examine each of these itemsets is clearly infeasible. Hence, researchers have developed various data structures and algorithms to effectively prune the search space by considering a property specific to a target itemset type. The most popular one is the "downward closure property" for FIs, meaning that any subset of an FI must also be frequent [1], [3]. Apriori algorithm utilises this property to dramatically prune the search space by ignoring itemsets that contain one or more infrequent subsets of items [3]. In addition, the downward closure property is used to construct FP-tree (Frequent Patten tree) that offers efficient, hierarchical organisation of items related only to FIs [4]. Also, considering the lack of the downward closure property for HUIs, upper bound utilities are defined so that

the property is maintained among "potential" itemsets that have possibilities to be HUIs [5], [6]. These upper bound utilities are exploited to construct a tree structure that hierarchically maintains items related only to potential itemsets.

However, the above kind of itemset mining methods are inflexible because a specific data structure or algorithm is needed to extract a different type of itemsets. Especially, one itemset type is defined by an interestingness measure and there exist many such measures as listed in [2]. It is clearly impractical to develop a data structure or algorithm for each of these itemset types. In other words, diverse types of itemsets remain undiscovered because neither data structure nor algorithm is available. Therefore, one crucial issue in itemset mining is the development of a unified framework to extract different types of itemsets.

For this issue, we propose *Generic Itemset Mining based on Reinforcement Learning* (GIM-RL). The main idea comes from how human searches for a target type of itemsets in a dataset. Most probably, human begins with briefly going through the dataset to get rough knowledge about which items seem to be important (or unimportant) for the target type, which items are related to each other and so on. Then, he/she makes a "plausible itemset" consisting of some important items, and examines whether it matches the target type or not. Afterwards, based on knowledge obtained in the past search experiences, human edits the plausible itemset by adding or removing items to create a new plausible itemset. One important point is that human adaptively changes an approach to make plausible itemsets depending on a target type as well as past experiences. Hence, human's itemset search is considered a key for devising a unified itemset mining framework.

We focus on *Reinforcement Learning* (RL) that is rooted in behavioural psychology and provides a framework where an artificial agent interacts with an uncertain environment to adaptively acquire the optimal policy of sequential decision-making [7]–[9]. The agent takes an action at each time step and the environment produces a reward as the response to the action. This trial-and-error step is repeated numerous times to accumulate rewards obtained by various actions at diverse states of the environment. Thereby, RL attempts to find the optimal policy that enables the agent to decide a sequence of actions which maximise cumulative rewards.

RL is utilised in GIM-RL to train an agent for extracting a target type of itemsets from a dataset in the following way: First, the agent performs an action to update an itemset by adding or removing an item. Then, the environment characterised by the dataset generates a reward that expresses how relevant the itemset updated by the agent is to the target type. GIM-RL collects a large number of trial-and-error steps in which the agent sometimes succeeded or sometimes failed in forming itemsets of the target type. By analysing these trial-and-error steps, the agent is trained to have the optimal policy for adding or removing items to form itemsets of the target type. Note that GIM-RL can extract any type of itemsets as long as one can define a reward that appropriately represents the relevance of an itemset to the type. We demonstrate that

GIM-RL can be generally applied to mining of HUIs [5], [6], [10], [11], FIs [1], [3], [4] and ARs [1], [3], [12].

One big by-product of GIM-RL is that it outputs not only extracted itemsets but also a trained agent. On the other hand, most of existing methods leave nothing behind except extracted itemsets. In other words, GIM-RL retains knowledge obtained in the mining process as the trained agent, while it is quite wasteful that most of existing methods throw such knowledge out. With respect to this, there exist many cases where the same mining process is performed on multiple related datasets. For example, one may want to inspect trend changes by extracting HUIs from two datasets collected in different terms. It is reasonable to use knowledge obtained from one of these datasets to speed up the mining process on the other dataset. Moreover, if the latter dataset is too huge to perform itemset mining from scratch, knowledge from the former dataset may be useful for accomplishing adequate mining on the latter one.

Based on the above consideration, we investigate GIM-RL's novel potential called *agent transfer*. This means that an agent is firstly trained on a source dataset, and then it is transferred to another compatible agent for a target dataset. If the source and target datasets are related to each other, they are thought to be characterised by similar relations among items. Thus, the agent for the source dataset is expected to be useful also for the target dataset. That is, the transferred agent only needs fine-tuning and offers more efficient itemset mining on the target dataset, compared to an agent trained from scratch. Unlike itemset mining, this kind of "transfer learning" is popular in different application domains where a model pre-trained on a source dataset is transferred to another model designed for a target dataset [13], [14]. We bring transfer learning into itemset mining as agent transfer, and show its possibility to significantly accelerate itemset mining.

This paper is organised as follows: The next section provides a survey of existing itemset mining methods to clarify the advantages of GIM-RL. Section III presents a methodological explanation of GIM-RL together with our reward designs to extract HUIs, FIs and ARs. The experimental results using GIM-RL based on these rewards are shown in Section IV. Here, our agent transfer approach and the results demonstrating its effectiveness are also described. Section V concludes this paper by discussing several future directions to extend GIM-RL. In addition to the above-mentioned main contents, Appendixes A to E give implementation details of GIM-RL, additional experiments and a small remark of one experimental dataset. Finally, many abbreviations and mathematical symbols are used in this paper. Thus, Appendix F offers a list of abbreviations and the one of symbols in order for readers to follow this paper more easily.

II. RELATED WORK

Existing itemset mining methods are roughly divided into two categories, *exhaustive* and *non-exhaustive*. The former includes methods that enumerate all the itemsets matching a target type. However, the runtime of exhaustive methods

significantly degrades as a dataset enlarges, especially, the increase in the number of distinct items causes the exponential expansion of the search space. To overcome this, non-exhaustive methods generate an approximate set of itemsets matching the target type. Below we first review several existing methods in the exhaustive and non-exhaustive categories, and then clarify the advantages of GIM-RL compared to those methods.

In general, an exhaustive method uses a data structure or algorithm specialised to a target type. For example, considering the downward closure property for FIs, Apriori algorithm [3] and FP-growth (Frequent Pattern growth) algorithm based on FP-tree [4] are developed for efficient enumeration of FIs. Apart from FIs, a divide-and-conquer approach based on a bitwise vertical representation of a dataset is developed to extract closed FIs, each of which has no superset supported by the same set of transactions [15]. The method in [16] enumerates maximal FIs that are included in no other FI, by devising a depth-first itemset expansion and dataset reductions. For efficient extraction of weighted FIs consisting of items associated with high weights, FP-growth is extended by crafting upper bound weights, three pruning techniques and a parallel mining algorithm [17]. Infrequent weighted itemsets consisting of rare and lowly weighted items are extracted by revising FP-growth with a specialised interestingness measure and a technique of early discarding unpromising items [18] (please see [19] for a survey of existing infrequent itemset mining methods and [20] for an up-to-date survey). An HUI is an extension of a weighted FI in the sense that its utility is computed by considering both the weight and quantity of each item in a transaction. To efficiently extract all the HUIs, researchers have developed a list that facilitates the expansion of an itemset and the corresponding utility calculation [10], and a tree structure based on the downward closure property of upper bound utilities [5], [6]. Finally, to extract FIs satisfying certain constraints like "the median of weights of items in an FI must be larger than a threshold", those constraints are converted so as to exhibit the downward closure property, and incorporated into FP-growth [21].

Exhaustive methods reviewed above are inflexible because a different data structure or algorithm is needed to extract a different type of itemsets. In contrast, GIM-RL can extract diverse types of itemsets only by defining a reward suitable for each type. By referring to the reward definitions for HUIs and FIs in Section III-B, one can easily design rewards for weighted FIs and infrequent weighted itemsets. In addition, it is expected that most of maximal FIs can be extracted using a reward, which is computed by checking the frequency of an itemset and the existence of its supersets in the set of already explored itemsets. This reward leads an agent to form itemsets that not only are frequent but also include more items. Closed FIs are likely to be extracted using a similar reward. Furthermore, many types of itemsets like ARs are divided into the antecedent and consequent, and let us assume the extraction of itemsets characterised by a consequent with one item. Under this setting, by exploiting the approach described in

Section III-B3, GIM-RL is expected to train an agent that can extract itemsets matching each of 38 interestingness measures listed in [2].

One of the most popular non-exhaustive itemset mining approaches is *Evolutionary Computation* (EC) that offers a metaheuristic where nature-inspired operators are iteratively used to update itemsets into better ones in terms of a fitness function [22]. One feature of EC-based methods is their predictable runtimes because itemset extraction is terminated by a specified number of iterations and no complicated process is needed for updating itemsets. One main class of EC-based methods is characterised by *Genetic Algorithm* (GA) that iteratively selects promising itemsets using a fitness function, and exploits them to create new itemsets based on crossover and mutation operators [23], [24]. Another main class is based on *swarm intelligence-based algorithms* that iteratively update itemsets based on operations inspired by the collective behaviours of swarms like ants, bees and bats [11], [22].

Since there is no guarantee that GIM-RL can extract all the itemsets of a target type, it is classified as a non-exhaustive method and has a high similarity to EC-based methods. This is because a reward in GIM-RL corresponds to a fitness function in EC-based methods, and any type of itemsets can be extracted by defining a suitable fitness function for the type. However, the biggest difference is that each EC-based method relies on a heuristically pre-defined strategy (i.e., metaheuristic) to extract itemsets, while GIM-RL analyses a dataset and learns such a strategy as a trained agent. In other words, the former only outputs extracted itemsets, whereas GIM-RL produces those itemsets as well as the trained agent that captures generalised characteristics of items in the dataset and can be transferred to another similar dataset.

Another popular approach to non-exhaustive itemset mining is *pattern sampling* that approximates a probability distribution over the search space by associating each itemset with a probability, which is proportional to its relevance to a target type in terms of an interestingness measure [25], [26]. Thus, itemsets sampled according to this probability distribution are a representative subset of itemsets matching the target type. However, only a limited number of itemset types can be treated by pattern sampling because of their compatibilities with sampling algorithms, for instance, an itemset type needs to be represented by a specified weight function form [25] or by a combination of XOR constraints [26]. GIM-RL can extract a much more variety of itemsets. In addition, no consideration is given on whether a probability distribution approximated for a source dataset can be transferred to the one for a target dataset. For this, GIM-RL offers a very flexible agent transfer in which an agent can be transferred between the source and target datasets even if they are characterised by different sets of distinct items.

Agent transfer is related to incremental itemset mining where a dataset is updated by adding, deleting and modifying transactions [1], [5], [27], and itemset mining in a stream where transactions arrive in rapid succession [1], [28]. While these two tasks treat datasets that change over time, source

and target datasets for agent transfer are fixed before extracting itemsets. As another substantial difference, the above two tasks focus on data structures to efficiently manage or summarise information necessary for extracting itemsets [5], [27], [28]. In agent transfer, such information is held by an agent (neural network) that captures generalised relations among items in the source or target dataset.

Another related approach is multitask itemset mining that performs joint analysis of multiple related datasets to extract "global" itemsets which are eligible on most of those datasets [29]. Compared to this, agent transfer carries out ordered analysis of source and target datasets because an agent is firstly trained on the former and then transferred to the latter. In addition, the method in [29] extracts global itemsets by directly holding and checking "local" itemsets extracted from each of multiple datasets. With respect to this, instead of itemsets extracted from the source dataset, agent transfer holds the trained agent as the abstracted information of those itemsets.

To our best knowledge, GIM-RL is the first method that adopts RL to train a machine learning model (i.e., agent defined by a neural network) for itemset mining. Correspondingly, agent transfer is not explored in any existing work. As discussed in Section IV-C, one problem of GIM-RL is its slow runtime because it needs multiple scans over a dataset to compute rewards and states of an environment. Of course, fast reward/state computation is one important future work. But, we believe that the efficiency of GIM-RL cannot be measured only by its runtime. The reason is that, according to a user-defined reward for a target type, GIM-RL adaptively trains an agent that can extract itemsets of this type, although one or more days may be needed. This seems much more efficient compared to a case where someone spends one or more months to implement a specialised data structure or algorithm for the target type.

III. GIM-RL

In this section, we first describe GIM-RL's general framework that can be commonly used to extract various types of itemsets. Then, we provide specific reward definitions to extract HUIs, FIs and ARs. Finally, the agent training process of GIM-RL is intuitively illustrated using a small example dataset. Also, a list of mathematical symbols used to explain GIM-RL is provided in Appendix F to make this section easier to follow.

A. GENERAL FRAMEWORK

Let $\mathcal{D} = \{T_1, \cdots, T_N\}$ be a dataset containing N transactions, and $\mathcal{I} = \{i_1, \cdots, i_M\}$ be the set of M distinct items each of which is included in at least one transaction in $\mathcal{D}$. Each transaction T_n ($1 \leq n \leq N$) in $\mathcal{D}$ is a subset of $\mathcal{I}$ (i.e., $T_n \subseteq \mathcal{I}$). Letting $|T_n|$ denote the number of items in T_n, we describe $T_n = \{i_{n,1}, \cdots, i_{n,|T_n|}\}$ where $i_{n,l}$ ($1 \leq l \leq |T_n|$) is the lth item in T_n (i.e., $i_{n,l} \in \mathcal{I}$). We denote by X an itemset consisting of items in $\mathcal{I}$ (i.e., $X \subseteq \mathcal{I}$). Let us consider an interestingness measure $\varphi(X)$ that takes X as input, analyses

transactions in $\mathcal{D}$ and outputs a value expressing the relevance of X to a target type. For example, $\varphi(X)$ for FIs returns the support (frequency) of X in $\mathcal{D}$ and $\varphi(X)$ for HUIs outputs X's utility. In addition, $\varphi(X)$ can be flexibly used for an itemset type requiring multiple conditions. For instance, to extract ARs by considering the support and confidence of X, one can design $\varphi(X)$ that outputs a value depending on whether X meets either or both of support and confidence thresholds. Under the setting described above, the goal of itemset mining is to extract from $\mathcal{D}$ every itemset X for which $\varphi(X)$ is larger than a pre-defined threshold ξ (i.e., $\varphi(X) \geq \xi$).

We formulate itemset mining as an RL problem shown in Fig. 1. It is assumed that the environment signifies one step of the mining process. Specifically, the kth step is performed to modify the itemset defined by a bit-vector $\boldsymbol{b}_k$ into a new one that is likely to match a target type. Formally, $\boldsymbol{b}_k = (b_{k,1}, \cdots, b_{k,M})^T$ is an M-dimensional binary vector where $b_{k,m} \in \{0, 1\}$ ($1 \leq m \leq M$) represents the inclusion of the mth item i_m in $\mathcal{I}$, namely, $b_{k,m} = 1$ indicates i_m is included, otherwise not-included. We express the itemset defined by $\boldsymbol{b}_k$ as $X(\boldsymbol{b}_k)$. The environment produces a state $\boldsymbol{s}_k = (s_{k,1}, \cdots, s_{k,M})^T$ having the same dimensionality to $\boldsymbol{b}_k$. Here, $s_{k,m}$ ($1 \leq m \leq M$) exhibits how useful it is to change $b_{k,m}$ for forming an itemset of the target type. In our implementation, $s_{k,m}$ is computed based on the simulation of the one-step-ahead future, in which $b_{k,m}$ is virtually changed to create the bit-vector $\boldsymbol{b}_k'$ and the corresponding itemset $X(\boldsymbol{b}_k')$. That is, $X(\boldsymbol{b}_k)$ and $X(\boldsymbol{b}_k')$ differ only in the inclusion of i_m. Then, $s_{k,m}$ is calculated as $\varphi(X(\boldsymbol{b}_k'))$ by applying the interestingness measure $\varphi(X)$ to $X(\boldsymbol{b}_k')$. Please see Appendix B for more details to compute $s_{k,m}$. Since the computation of $\boldsymbol{s}_k$ needs to check transactions in $\mathcal{D}$, $\boldsymbol{s}_k$ in Fig. 1 is connected with the non-filled arrow from $\mathcal{D}$.

$\boldsymbol{s}_k$ shows hints about which items are likely to be included or excluded to form an itemset of the target type. Thus, as illustrated by the solid arrow from $\boldsymbol{s}_k$ to the agent in Fig. 1, $\boldsymbol{s}_k$ is fed into the agent to decide an action of changing one value in $\boldsymbol{b}_k$. As a result, $\boldsymbol{b}_k$ is updated into $\boldsymbol{b}_{k+1}$, meaning that the new itemset $X(\boldsymbol{b}_{k+1})$ is created by adding or removing one item to or from $X(\boldsymbol{b}_k)$. The agent's action is designed based on human's itemset search where he/she modifies an itemset by thinking about which item's inclusion or exclusion is crucial for the target type. Then, the environment generates a reward r_k that has a close relation to $\varphi(X(\boldsymbol{b}_{k+1}))$. Like $\boldsymbol{s}_k$, the computation of r_k needs to check occurrences of $X(\boldsymbol{b}_{k+1})$ in $\mathcal{D}$, so a non-filled arrow is placed between the bit-vector (that is now $\boldsymbol{b}_{k+1}$) and $\mathcal{D}$ in Fig. 1. As depicted by the dashed arrow in Fig. 1, the agent receives r_k as an evaluation score for the action taken at the kth step. If $X(\boldsymbol{b}_{k+1})$ matches the target type, the agent obtains high r_k. Afterwards, the environment creates a new state $\boldsymbol{s}_{k+1}$ based on which the agent updates $\boldsymbol{b}_{k+1}$ into $\boldsymbol{b}_{k+2}$ and receives r_{k+1}. This way, the agent iteratively updates the bit-vector to form various itemsets that possibly match the target type. In this framework, our goal is to train the agent that maximises cumulative rewards
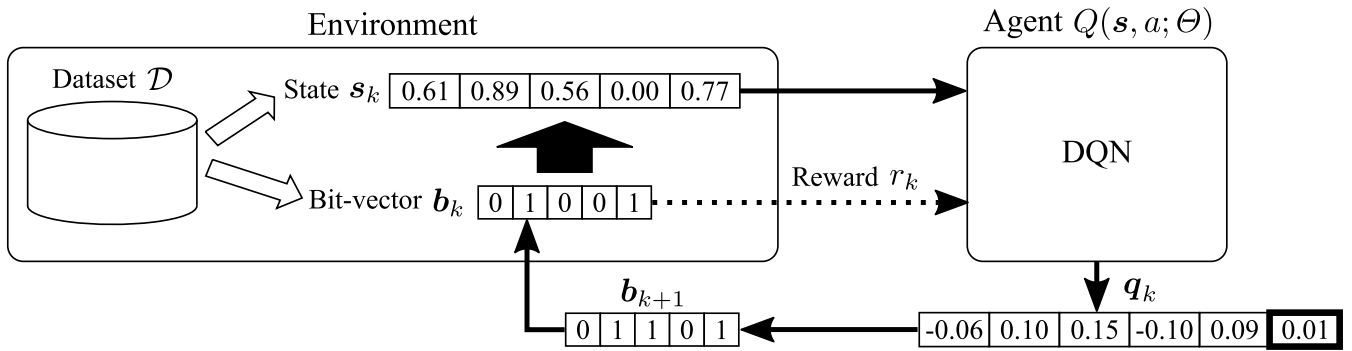
FIGURE 1. An overview of GIM-RL.

over steps. This means that the agent can extract as many itemsets matching the target type as possible.

It should be noted that s_k only provides the information for selecting one action, and does not include information for determining a sequence of multiple actions. In other words, the target type of itemsets are not necessarily extracted by greedily changing b_k's value that corresponds to s_k's highest value. Instead, the agent needs to have an intelligent policy to select an action at the current step by considering what kind of itemsets will be obtained at the future steps. Intuitively, we aim to train an agent that takes actions causing itemsets of non-target types at some consecutive steps, in order to extract many itemsets of the target type after those steps.

To this end, we employ *Q-learning* that trains an agent characterised by a *Q function* which takes as input an action a and a state s of the environment, and outputs a's quality at s [7]–[9]. Ideally, the optimal Q function $Q^*(s, a)$ quantifies a's quality as follows:

$$Q^*(s, a) = \max_{\pi} \mathbb{E}\left[\sum_{k'=k}^{K} \gamma^{k'-k} r_{k'} \,\middle|\, \tilde{s}_k = s, \tilde{a}_k = a, \pi\right], \quad (1)$$

where K is the final step of the mining process, γ is a discount factor by which a reward at a further future step is discounted more strongly, and π represents a policy for action selection. Eq. 1 means that a's quality at s is computed as the maximum expected cumulative reward, which is achievable after seeing s at the kth step and taking a. Here, s and a are assigned to the variables $\tilde{s}_k$ and $\tilde{a}_k$ for representing a state and an action at the kth step, respectively. Simply speaking, in itemset mining, $Q^*(s, a)$ indicates the maximum number of itemsets of the target type after taking a at s. However, it is impractical to directly build $Q^*(s, a)$ because of the huge number of state-action combinations. Thus, considering the recent success of deep Q-learning [7]–[9], we approximate $Q^*(s, a)$ by a neural network, called *Deep Q-Netwrok* (DQN), defined by a set of parameters Θ. Therefore, $Q^*(s, a)$ is parametrised as $Q(s, a; \Theta)$, and our goal is to optimise Θ of the DQN so that actions selected based on $Q(s, a; \Theta)$ form many itemsets of the target type. Note that we interchangeably use the terms

"agent" and "DQN" and the notation "$Q(s, a; \Theta)$" in the following discussions.

Let $\mathcal{A}$ be a set of possible actions. In Fig. 1, $Q(s, a; \Theta)$ is used to compute $Q(s_k, a; \Theta)$ that approximates the quality of each action $a \in \mathcal{A}$ at the specific state s_k of the kth step. Then, the action a_k is chosen as the one corresponding to the highest value of $Q(s_k, a; \Theta)$. With no deep elaboration, one may define $\mathcal{A}$ to include M actions each of which changes $b_{k,m}$ in b_k (i.e., $|\mathcal{A}| = M$). But, as depicted by the bold-lined rectangle at the bottom-right of Fig. 1, in order to help the agent explore diverse itemsets, we add to $\mathcal{A}$ one more action that randomly initialises the bit-vector. This means that the agent is given an opportunity to stop updating b_k and re-start itemset mining from the initialised bit-vector. As an implementation detail, the random bit-vector initialisation is performed based on the probability distribution where the probability of $b_{k,m} = 1$ is proportional to the frequency of the mth item i_m in $\mathcal{D}$, and is repeated until the bit-vector defines an itemset that exists in $\mathcal{D}$. In other words, it is meaningless to examine itemsets not-existing in $\mathcal{D}$.

Also, an $(M+1)$-dimensional vector $q_k = (q_{k,1}, \cdots, q_{k,M}, q_{k,M+1})^T$ is defined to represent the collection of $Q(s_k, a; \Theta)$s for all the $M + 1$ actions at the kth step, as depicted at the bottom-right of Fig. 1. That is, $q_{k,m}$ ($1 \leq m \leq M$) indicates an approximate quality of the action to change the inclusion of the mth item i_m in $X(b_k)$, and $q_{k,M+1}$ expresses an approximate quality of the random bit-vector initialisation. q_k will be used to simplify the descriptions in Section IV-A and Appendix B.

According to Bellman equation, the optimisation of Θ is done by making $Q(s_k, a_k; \Theta)$ and $r_k + \gamma \max_{a' \in \mathcal{A}} Q(s_{k+1}, a'; \Theta)$ as close as possible [7]–[9]. An intuitive interpretation is that if $Q(s, a; \Theta)$ is a good approximation of $Q^*(s, a)$, the maximum expected cumulative reward estimated for taking a_k at s_k should be equal or very similar to the sum of r_k obtained by taking a_k at s_k with the maximum expected cumulative reward estimated for the action at the next state s_{k+1}. However, the following two issues make it unstable to optimise Θ by directly using $Q(s_k, a_k; \Theta)$ and $r_k + \gamma \max_{a' \in \mathcal{A}} Q(s_{k+1}, a'; \Theta)$. First, the agent's experiences (s_k, a_k, r_k, s_{k+1})s obtained at consecutive steps are very

similar because only one value in $\boldsymbol{b}_k$ is changed at each step except its random initialisation. As a result, $Q(\boldsymbol{s}, a; \Theta)$ trained on those experiences is very biased. To overcome this, we use a *replay memory* $\mathcal{P}$ that is a queue to store the recent $|\mathcal{P}|$ experiences, and optimise Θ on experiences randomly sampled from $\mathcal{P}$ [7], [8]. Second, Θ is included in the ''target value $r_k + \gamma \max_{a' \in \mathcal{A}} Q(\boldsymbol{s}_{k+1}, a'; \Theta)$'' that $Q(\boldsymbol{s}_k, a_k; \Theta)$ targets to approximate. That is, the target value changes every time Θ is updated, which makes the optimisation of Θ unstable. To address this, separate DQNs are used for $Q(\boldsymbol{s}_k, a_k; \Theta)$ and the target value [7], [8]. The parameters Θ of the DQN $Q(\boldsymbol{s}, a; \Theta)$ are updated every step. On the other hand, the DQN $Q(\boldsymbol{s}, a; \Theta^-)$ used for the target value is called *target network* and characterised by the parameters Θ^- that are periodically updated every a certain number of steps. To sum up, as shown in the following equation, $Q(\boldsymbol{s}, a; \Theta)$ is trained by minimising the squared difference between $Q(\boldsymbol{s}_k, a_k; \Theta)$ and the target value $r_k + \gamma \max_{a' \in \mathcal{A}} Q(\boldsymbol{s}_{k+1}, a'; \Theta^-)$ on experiences randomly sampled from $\mathcal{P}$:

$$\mathbb{E}_{(\boldsymbol{s}_k, a_k, r_k, \boldsymbol{s}_{k+1}) \sim \mathcal{P}} \left[\left(r_k + \gamma \max_{a' \in \mathcal{A}} Q(\boldsymbol{s}_{k+1}, a'; \Theta^-) \right. \right.$$
$$\left. \left. - Q(\boldsymbol{s}_k, a_k; \Theta) \right)^2 \right] \quad (2)$$

Finally, GIM-RL to train an agent $Q(\boldsymbol{s}, a; \Theta)$ is summarised in Algorithm 1 of Appendix A. Overall, GIM-RL executes E episodes consisting of K steps where the agent updates a bit-vector to form different itemsets. Here, each episode is used as a unit of itemset mining in the sense that a new mining process is initiated by a randomly initialised bit-vector, as shown at line 7 in Algorithm 1 of Appendix A. Although itemset mining is not episodic, we believe that using episodes is useful for the agent to reinitialise the itemset search and explore diverse itemsets.

B. REWARD DEFINITION

We explain our definitions of rewards used to extract HUIs, FIs and ARs. Of course, it is possible to define better rewards than ours below. All the rewards in this paper are based on the common scheme dealing with the following four cases:

Case 1: The itemset $X(\boldsymbol{b}_{k+1})$ resulting from the action a_k at the state $\boldsymbol{s}_k$ does not exist in $\mathcal{D}$. The agent receives a reward of -1, so that it is guided to not perform a_k at $\boldsymbol{s}_k$ as well as at a similar state.

Case 2: The interestingness measure value $\varphi(X(\boldsymbol{b}_{k+1}))$ is less than the quarter of a pre-specified threshold ξ (i.e., $\varphi(X(\boldsymbol{b}_{k+1})) < \xi/4$) and the agent receives a reward of 0. This means that taking a_k at $\boldsymbol{s}_k$ or a similar state has no important impact on forming the target type of itemsets.

Case 3: $\varphi(X(\boldsymbol{b}_{k+1})) \geq \xi/4$ is the condition of this case, which is further divided into four sub-cases, $\xi/4 \leq \varphi(X(\boldsymbol{b}_{k+1})) < \xi/2$, $\xi/2 \leq \varphi(X(\boldsymbol{b}_{k+1})) < 3\xi/4$, $3\xi/4 \leq \varphi(X(\boldsymbol{b}_{k+1})) < \xi$ and $\xi \leq \varphi(X(\boldsymbol{b}_{k+1}))$, in which the agent receives rewards of 1, 2, 3 and 4, respectively. These rewards are defined according to how close $X(\boldsymbol{b}_{k+1})$ is to the target type. Thereby, the agent is informed of how important it is to take a_k at $\boldsymbol{s}_k$ or a similar

state for forming itemsets of the target type. The reason why the last sub-case $\xi \leq \varphi(X(\boldsymbol{b}_{k+1}))$ is included is explained below.

Case 4: This case is defined by two conditions. The first examines whether $X(\boldsymbol{b}_{k+1})$ matches the target type (i.e., $\xi \leq \varphi(X(\boldsymbol{b}_{k+1}))$), and the second checks whether $X(\boldsymbol{b}_{k+1})$ has not yet been extracted in the current episode. $X(\boldsymbol{b}_{k+1})$ satisfying these two conditions is a newly extracted itemset of the target type, and we aim to extract such itemsets. Thus, by providing a very high reward of 100, the agent is led to extract many unique itemsets of the target type. Also, if $X(\boldsymbol{b}_{k+1})$ is an already extracted itemset, it seems unreasonable to regard a_k as meaningless because $X(\boldsymbol{b}_{k+1})$ anyway matches the target type. This situation is captured by the last sub-case in Case 3, and the agent gets a reward (4) that is much smaller than the one (100) in this case.

To complete the above-mentioned scheme, we describe the computation of $\varphi(X(\boldsymbol{b}_{k+1}))$ for each of HUI, FI and AR.

1) $\varphi(X(\boldsymbol{b}_{k+1}))$ FOR HUI EXTRACTION

For the nth transaction $T_n = \{i_{n,1}, \cdots, i_{n,|T_n|}\}$ in $\mathcal{D}$, the lth item $i_{n,l}$ ($1 \leq l \leq |T_n|$) is associated with the item-specific utility $p(i_{n,l})$ and the quantity $q(i_{n,l})$ in T_n. Under this setting, $\varphi(X(\boldsymbol{b}_{k+1}))$ representing the utility of $X(\boldsymbol{b}_{k+1})$ is computed as follows [5], [6], [10], [11]:

$$\varphi(X(\boldsymbol{b}_{k+1}))$$
$$= \sum_{X(\boldsymbol{b}_{k+1}) \subseteq T_n \wedge T_n \in \mathcal{D}} \sum_{i_{n,l} \in X(\boldsymbol{b}_{k+1}) \wedge i_{n,l} \in T_n} p(i_{n,l}) \, q(i_{n,l}) \quad (3)$$

The inner summation refers to each transaction containing $X(\boldsymbol{b}_{k+1})$ as T_n, and computes the sum of item-specific utilities weighted by their corresponding quantities for all items in $X(\boldsymbol{b}_{k+1})$. As signified by the outer summation, $\varphi(X(\boldsymbol{b}_{k+1}))$ is calculated as the total of such weighted sums for all transactions containing $X(\boldsymbol{b}_{k+1})$.

2) $\varphi(X(\boldsymbol{b}_{k+1}))$ FOR FI EXTRACTION

$\varphi(X(\boldsymbol{b}_{k+1}))$ is defined as $sup(X(\boldsymbol{b}_{k+1}))$ representing the support (frequency) of $X(\boldsymbol{b}_{k+1})$ in $\mathcal{D}$, that is, the number of transactions containing $X(\boldsymbol{b}_{k+1})$.

3) $\varphi(X(\boldsymbol{b}_{k+1}))$ FOR AR EXTRACTION

To define an AR, an itemset X is firstly divided into two subsets X_a and X_c (i.e., $X = X_a \cup X_c$), and a rule $X_a \rightarrow X_c$ is created by viewing X_a and X_c as the antecedent and consequent, respectively. $X_a \rightarrow X_c$ is regarded as an AR if $sup(X)$ is larger than the minimum support threshold *min_sup* and the confidence, which is the conditional probability of X_c given X_a (i.e., $sup(X)/sup(X_a)$), is larger than the minimum confidence threshold *min_conf*. GIM-RL extracts ARs using $\varphi(X(\boldsymbol{b}_{k+1}))$ that jointly considers the support and confidence conditions.

As one remark, the current GIM-RL is limited to extracting ARs each of which is characterised by X_c with one item. But, focusing only on such ARs is considered reasonable because
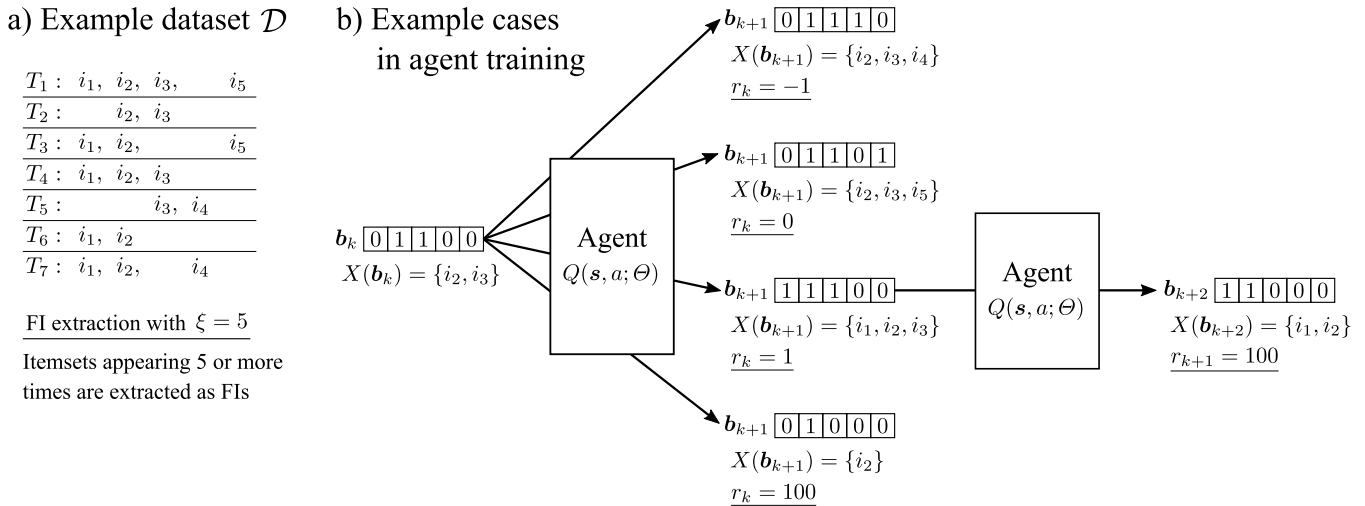
a) Example dataset $\mathcal{D}$ b) Example cases in agent training

$T_1:$	$i_1,$	$i_2,$	$i_3,$		i_5
$T_2:$		$i_2,$	i_3		
$T_3:$	$i_1,$	$i_2,$			i_5
$T_4:$	$i_1,$	$i_2,$	i_3		
$T_5:$			$i_3,$	i_4	
$T_6:$	$i_1,$	i_2			
$T_7:$	$i_1,$	$i_2,$		i_4	

FI extraction with $\xi = 5$

Itemsets appearing 5 or more times are extracted as FIs

FIGURE 2. An illustration of how an agent $Q(s, a; \Theta)$ for FI extraction is trained in GIM-RL.

allowing X_c to include multiple items often produces an extremely large number of ARs. In addition, several existing methods address the extraction of ARs with X_cs characterised by single items [30], [31]. Moreover, assuming that human is paying attention to one itemset, he/she is likely to think whether a rule is created by regarding the itemset as X_a and a new item as X_c, or by seeing an item in the itemset as X_c and the other items as X_a. GIM-RL implements this human's search based on the difference between $X(\boldsymbol{b}_k)$ and $X(\boldsymbol{b}_{k+1})$. If $X(\boldsymbol{b}_{k+1})$ is the itemset created by adding one item to $X(\boldsymbol{b}_k)$, we consider that the agent takes the action to use $X(\boldsymbol{b}_k)$ and the added item as X_a and X_c, respectively. For an inverse case where $X(\boldsymbol{b}_{k+1})$ is the itemset created by removing one item from $X(\boldsymbol{b}_k)$, the agent's action is to use $X(\boldsymbol{b}_{k+1})$ and the removed item as X_a and X_c, respectively.

Such an agent's action to decide X_a and X_c that may constitute an AR, is evaluated by a reward based on two interesting measures $\varphi_1(X(\boldsymbol{b}_{k+1}))$ and $\varphi_2(X(\boldsymbol{b}_{k+1}))$. The former is used in Cases 2 and 3 to check the support of $X_a \cup X_c$ which is the itemset including more items between $X(\boldsymbol{b}_k)$ and $X(\boldsymbol{b}_{k+1})$. For Case 3 where $\varphi_1(X(\boldsymbol{b}_{k+1})) = sup(X_a \cup X_c)$ is equal to or larger than $\xi_1/4 = min\_sup/4$, the agent gets a reward of 1, 2, 3 or 4. This is because these X_a and X_c are close to constituting an AR in terms of the support condition. $\varphi_2(X(\boldsymbol{b}_{k+1}))$ is used in Case 4 and checks the confidence of $X_a \rightarrow X_c$, that is, $\varphi_2(X(\boldsymbol{b}_{k+1})) = sup(X_a \cup X_c)/sup(X_a)$. Note that Case 4 is triggered only when Case 3 is passed by $min\_sup \leq \varphi_1(X(\boldsymbol{b}_{k+1}))$. Thus, if $\xi_2 = min\_conf \leq \varphi_2(X(\boldsymbol{b}_{k+1}))$, X_a and X_c obtained by the agent action are verified as the AR $X_a \rightarrow X_c$ and a reward of 100 is given. Otherwise, the agent gets a reward of 4 resulting from Case 3.

Finally, it is possible to extract ARs that are individually characterised by X_c with multiple items, by allowing an agent to take actions for changing multiple values in the bit-vector. However, this causes an exponential increase of the agent's

action space, so a smart approach is needed. This issue is left as our future work.

C. ILLUSTRATIVE EXAMPLE

For clear understanding of GIM-RL, we explain an intuitive example of how an agent $Q(s, a; \Theta)$ is trained by referring to Fig. 2. First of all, let us assume that an agent for FI extraction is trained using the dataset $\mathcal{D}$ shown in Fig. 2 (a). Here, $\mathcal{D}$ contains $N = 7$ transactions $T_1, \cdots, T_7$ defined on $M = 5$ distinct items $i_1, \cdots, i_5$. The minimum support threshold $\xi = min\_sup$ is set to five. Hence, the agent is trained by optimising its parameters Θ to extract itemsets appearing five or more times in $\mathcal{D}$ as FIs.

Fig. 2 (b) depicts some example cases that can possibly occur at the kth, $(k + 1)$th and $(k + 2)$th steps to train an agent on $\mathcal{D}$. The centre of Fig. 2 (b) shows four cases that originate from the bit-vector at the kth step $\boldsymbol{b}_k = (0, 1, 1, 0, 0)^T$ defining the itemset $X(\boldsymbol{b}_k) = \{i_2, i_3\}$. The case at the top is characterised by $\boldsymbol{b}_{k+1} = (0, 1, 1, 1, 0)^T$ resulting from the action to change i_4's inclusion in $X(\boldsymbol{b}_k)$. According to our reward definition, the reward $r_k = -1$ is given to the agent because $X(\boldsymbol{b}_{k+1}) = \{i_2, i_3, i_4\}$ defined by $\boldsymbol{b}_{k+1}$ does not exist in $\mathcal{D}$. As a result, Θ is updated so that $Q(s, a; \Theta)$ outputs a very small value for the action to change i_4's inclusion at the state s_k computed from $\boldsymbol{b}_k$. To put it more simply, the agent becomes to not select this action for $\boldsymbol{b}_k$.

It should be noted that the above-mentioned update of Θ affects action selection also for a bit-vector which is "statistically similar" to $\boldsymbol{b}_k$, although illustrating this in Fig. 2 is difficult. The key is that $Q(s, a; \Theta)$ is defined on a state s represented by a continuous vector. Regarding this, two bit-vectors are considered statistically similar if items contained in the itemsets defined by them have similar statistical characteristics. Accordingly, the states computed from these bit-vectors are characterised by similar continuous vectors,

for which $Q(s, a; \Theta)$ outputs similar values. Thus, when Θ is updated to select (or not select) a certain action at a particular state, this action selection is propagated to states computed from statistically similar bit-vectors. In what follows, such bit-vectors are not described for the sake of brevity. But, please note that Θ's updates described below influence on action selection for statistically similar bit-vectors.

The cases shown at the second-top, third-top and bottom of the centre of Fig. 2 (b) can be interpreted in a similar way to the case at the top. The case at the second-top is based on $\boldsymbol{b}_{k+1} = (0, 1, 1, 0, 1)^T$ obtained by the action to change i_5's inclusion in $X(\boldsymbol{b}_k)$. Since $X(\boldsymbol{b}_{k+1}) = \{i_2, i_3, i_5\}$ appears once in $\mathcal{D}$, the agent receives the reward $r_k = 0$. This means to postpone the evaluation of the action to change i_5's inclusion at the state $\boldsymbol{s}_k$ computed from $\boldsymbol{b}_k$. Because of $r_k = 0$, Eq. 2 requires $Q(\boldsymbol{s}_k, a_k; \Theta)$ to be close to $\gamma \max_{a' \in \mathcal{A}} Q(\boldsymbol{s}_{k+1}, a'; \Theta^-)$. That is, $Q(s, a; \Theta)$ for the action to change i_5's inclusion in $X(\boldsymbol{b}_k)$ depends on what rewards will be obtained at the $(k + 1)$th and later steps. For the case at the third-top, changing i_1's inclusion in $X(\boldsymbol{b}_k)$ produces $X(\boldsymbol{b}_{k+1}) = \{i_1, i_2, i_3\}$ appearing twice in $\mathcal{D}$. Since this case is categorised as the first sub-case of Case 3 in our reward definition (i.e., $\xi/4 < sup(X(\boldsymbol{b}_{k+1})) = 2 < \xi/2$), the agent receives the reward $r_k = 1$. As a result, $Q(s, a; \Theta)$ is refined to output a relatively high value for the action to change i_1's inclusion at $\boldsymbol{s}_k$, making the agent possibly select this action. The case at the bottom is triggered by the action to change i_3's inclusion in $X(\boldsymbol{b}_k)$ and the resulting $X(\boldsymbol{b}_{k+1}) = \{i_2\}$ appears six times in $\mathcal{D}$ and is extracted as an FI. The reward $r_k = 100$ guides $Q(s, a; \Theta)$ to output a high value for the action to change i_3's inclusion at $\boldsymbol{s}_k$, making the agent likely to select this action.

One important remark is that $Q(s, a; \Theta)$ approximates the maximum expected cumulative reward defined in Eq. 1. Thus, training of $Q(s, a; \Theta)$ considers not only the reward r_k at the kth step, but also rewards at future steps. In fact, as seen from Eq. 2, $Q(s, a; \Theta)$ is not trained to directly approximate r_k, but to achieve a situation where the maximum expected cumulative reward approximated at the kth step is close to the sum of the actual reward r_k at the kth step and the maximum expected cumulative reward approximated at the $(k + 1)$th step.[1] Thereby, $Q(s, a; \Theta)$ reflects the estimation of what rewards will be acquired at future steps after taking a at s. This is illustrated in the case shown at the right of Fig. 2 (b). This case is provoked by the action to change i_3's inclusion in $X(\boldsymbol{b}_{k+1})$ and the resulting $X(\boldsymbol{b}_{k+2}) = \{i_1, i_2\}$ appearing five times in $\mathcal{D}$ is extracted as an FI. The reward $r_{k+1} = 100$ obviously supervises $Q(s, a; \Theta)$ to output a high value for the action to change i_3's inclusion at $\boldsymbol{s}_{k+1}$ computed from $\boldsymbol{b}_{k+1}$. In addition to this, $Q(s, a; \Theta)$ for the action to

change i_1's inclusion at $\boldsymbol{s}_k$ is also increased, because this action is now linked to the FI extracted at the $(k + 2)$th step. This way, in GIM-RL, an agent is trained to select an action that leads to not only the immediate extraction of an FI at the current step but also the extraction of many FIs at future steps.

IV. EXPERIMENTAL RESULTS

We test GIM-RL using the five datasets shown in Table 1. These datasets are chosen because they have different characteristics in terms of numbers of transactions, numbers of distinct items and average numbers of items in one transaction, as exhibited in rows N, M and "Avg. $|T_n|$" in Table 1, respectively. In addition, the datasets in Table 1 are popularly used in many existing works [11], [12], [24]–[26]. Through the extractions of HUIs, FIs and ARs from these datasets, we aim to demonstrate the generality of GIM-RL.

TABLE 1. Statistics about the experimental datasets.

	Chess	Mushroom	Accidents_10%	Connect	Pumsb
N	3196	8416	34018	67557	49046
M	75	119	468	129	2113
Avg. $\|T_n\|$	37	23	34	43	74

A. RESULTS FOR ITEMSET MINING

For each of the HUI, FI and AR extractions, the following two evaluations are performed: First, we use a baseline method that is developed to exhaustively enumerate all the itemsets of a target type. The number of itemsets extracted by the baseline method is maximum, so we examine how close the number of itemsets extracted by GIM-RL is to this maximum. Second, the effectiveness of agents trained by GIM-RL is examined by comparing the five agents below. Note that these agents commonly take $\boldsymbol{s}_k$ as input and output a_k to change the value of one dimension in $\boldsymbol{b}_k$ in Fig. 1, but they are different in how to decide a_k based on $\boldsymbol{s}_k$.

Ramdom: This agent just changes the value of a randomly selected dimension in $\boldsymbol{b}_k$. With *Random*, we aim to show the difficulty of each itemset extraction task where randomly changing values in $\boldsymbol{b}_k$ yields few itemsets of the target type.

State-ϵ: As described in Section III-A, each dimension's value in $\boldsymbol{s}_k$ represents the usefulness of changing the value of the corresponding dimension in $\boldsymbol{b}_k$ based on the simulation of the one-step-ahead future. One may think that $\boldsymbol{s}_k$ already contains enough information about which value in $\boldsymbol{b}_k$ should be changed. To answer this, we test *State-ϵ* that changes the value of $\boldsymbol{b}_k$'s dimension corresponding to the highest value in $\boldsymbol{s}_k$ with the probability $1 - \epsilon$, while changing the value of a randomly selected dimension with the probability ϵ to preserve the variety of itemsets to be explored.

State-prob: The motivation for testing this agent is the same to the one for *State-ϵ*. But, different from *State-ϵ*, *State-prob* determines a_k by random sampling according to the probability distribution, where the probability of changing the

---
[1]This objective progresses training of $Q(s, a; \Theta)$. The reason is that thanks to the consideration of the actual reward r_k, the target value $r_k + \gamma \max_{a' \in \mathcal{A}} Q(\boldsymbol{s}_{k+1}, a'; \Theta^-)$ is usually a more accurate approximation of the maximum expected cumulative reward than $Q(\boldsymbol{s}_k, a_k; \Theta)$. Hence, the approximation by $Q(\boldsymbol{s}_k, a_k; \Theta)$ can be more accurate by reducing the difference between it and the target value.

value of one dimension in b_k is proportional to the value of the corresponding dimension in s_k.

GIM-RL-Basic: This is the basic version of GIM-RL. A DQN that accepts s_k and outputs q_k is trained in the framework of Fig. 1. Each dimension in q_k indicates an approximate quality of taking one of $M + 1$ actions including the random bit-vector initialisation, as described in Section III-A. DQNs with the same architecture are trained for extracting HUIs and ARs, while simpler DQNs are used for FIs. Please see Appendix B (especially Fig. 6) for details on the architectures of these DQNs.

GIM-RL-Fusion: This is an extended version of *GIM-RL-Basic* by defining the output as the sum of q_k and s_k. To be precise, s_k misses the dimension for the random bit-vector initialisation, so s'_k is created by appending to s_k one dimension with the very small value "$0.02 \times$ (average of s_k)" for the initialisation. Then, the output of *GIM-RL-Fusion* is computed as $q'_k = \lambda s'_k + (1 - \lambda)q_k$. Of course, the additional use of s'_k is expected to make q'_k attain better action selection than q_k. But, q'_k plays a more important role to advance training of an agent. In principle, the agent can be trained when positive rewards are obtained. In other words, training of the agent does not proceed as long as rewards are zero or negative. Regarding this, at the beginning of training, it is statistically difficult for the agent to find "positive itemsets" that offer positive rewards using the imperfect $Q(s, a; \Theta)$. For this, s'_k significantly increases the probability of finding positive itemsets because it represents rough estimation of which items are likely to constitute positive itemsets. After finding some positive itemsets, the agent can be trained to some extend. This boosts the probability that the agent can find positive itemsets by its own exploitation of the trained $Q(s, a; \Theta)$, that is, the agent can do further training by itself. Considering this, λ in q'_k is gradually reduced to weaken the effect of s'_k as training proceeds. Please see Appendix C for the specific setting of λ.

1) RESULTS FOR HUI MINING

Table 2 summarises the results for HUI mining. First, HUI-Miner [10] implemented in SPMF library [32] is used as a baseline to extract all the HUIs from each dataset. The thresholds in the second row of Table 2 are the same to the ones used in [11], [24].[2] As shown in Table 2, 100% or nearly 100% of HUIs are extracted from each of the four datasets using *GIM-RL-Fusion*. This verifies the effectiveness of GIM-RL to train agents for extracting HUIs. Also, the fact that no HUI is extracted by *Random* implies the difficulty of HUI extraction. In addition, the poor performances of *State-ε* and *State-prob* indicate the ineffectiveness of directly using s_k. In what follows, *Random*, *State-ε* and *State-prob* are sometimes called "non-training agents" because their action selection policies are fixed in advance and are not optimised. Finally, the main reason why no HUI is extracted

TABLE 2. An overview of the HUI mining results.

	Chess	Mushroom	Accidents _10%	Connect
ξ (%)	29.0%	14.5%	13.0%	32.0%
Baseline	176	199	127	171
Random	0	0	0	0
State-ε	56	82	51	67
State-prob	0	0	0	0
GIM-RL-Basic	176	199	0	169
GIM-RL-Fusion	176 (100%)	197 (98.9%)	127 (100%)	169 (98.8%)

by *GIM-RL-Basic* for Accidents_10% is that it fails to find first some positive itemsets, so its training does not proceed. In contrast, *GIM-RL-Fusion* aided by s'_k stably produces very good performances on all the datasets.

2) RESULTS FOR FI MINING

Table 3 shows the results for FI mining. FP-growth [4] implemented in SPMF library [32] is selected as a baseline to extract all the FIs. By referring to [12], the thresholds in the second row are determined so that moderate numbers of FIs (1000-10000 FIs) are extracted. Note that FIs are extracted by the non-training agents, although numbers of extracted FIs significantly vary depending on datasets. For this, we point out the following two reasons attributed to the random bit-vector initialisation. First, a bit-vector is initialised according to the probability distribution based on the frequency of each item, so the itemset defined by this initialised bit-vector has a relatively high probability to be an FI. In addition, updating this bit-vector leads to finding other FIs with not-low probabilities. Second, the initialisation is repeated until the bit-vector defines an itemset that exists in $\mathcal{D}$. Focusing only on such itemsets dramatically reduces the search space, especially for small datasets like Chess. Because of the above two reasons, even *Random* can extract FIs. But, despite the fact that the non-training agents occasionally extract many FIs, *GIM-RL-Fusion* or *GIM-RL-Basic* extracts the highest numbers of FIs for all the datasets. This validates the effectiveness of GIM-RL also for extracting FIs.[3]

TABLE 3. An overview of the FI mining results.

	Chess	Mushroom	Pumsb	Connect
ξ (%)	80%	35%	90%	95%
Baseline	8227	1121	2607	2201
Random	2258	63	304	1541
State-ε	587	440	427	394
State-prob	4926	502	955	2144
GIM-RL-Basic	6154	1101	2544	2072
GIM-RL-Fusion	6843 (83.2%)	969 (86.4%)	2578 (98.8%)	2199 (99.9%)

---

[2]Also for the other threshold settings used in [11], [24], we have obtained results to validate the effectiveness of GIM-RL.

[3]For Chess, 7901 FIs (96.0% of FIs) are extracted by *GIM-RL-Fusion* using a DQN with the architecture for the HUI and AR extractions.

3) RESULTS FOR AR MINING

In Table 4, the results for AR mining are presented. Our baseline method first uses FPGrowth_association_rules implemented in SPMF library [32] to extract all the ARs, and then retains ARs each of which is characterised by a consequent with one item. The thresholds $\xi_1 = min\_sup$ and $\xi_2 = min\_conf$ are set based on [12], so that a reasonable number of ARs are extracted from each dataset. The trend of the results in Table 4 is similar to the one in Table 3. Because of the random bit-vector initialisation, the non-training agents can extract ARs from each dataset, but their performances significantly degrade on large datasets like Pumsb and Connect. In contrast, *GIM-RL-Fusion* can stably extract almost all ARs from each of the datasets.
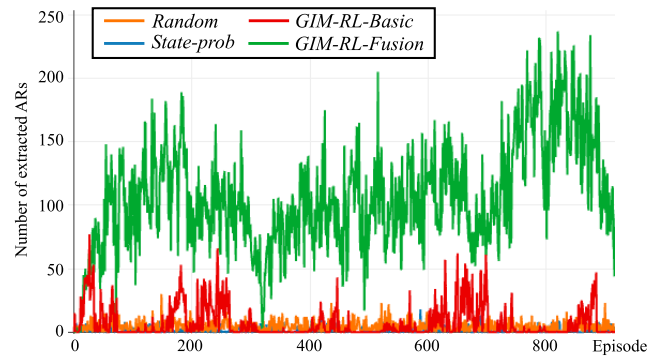

FIGURE 3. The transition of numbers of ARs extracted by each of *Random, State-prob, GIM-RL-Basic* and *GIM-RL-Fusion* over the passage of episodes (the dataset is connect).

TABLE 4. An overview of the AR mining results.

	Chess	Mushroom	Pumsb	Connect
ξ_1 (%)	90%	50%	90%	95%
ξ_2 (%)	80%	80%	80%	80%
Baseline	2351	331	11366	10106
Random	2214	330	120	1287
State-ϵ	639	115	640	615
State-prob	2350	331	481	3545
GIM-RL-Basic	2202	331	4832	4026
GIM-RL-Fusion	2340 (99.5%)	330 (99.6%)	10111 (88.9%)	9225 (91.2%)

We describe a deeper insight in the AR extraction on Connect. Fig. 3 shows how many ARs are extracted in each episode by *Random, State-prob, GIM-RL-Basic* and *GIM-RL-Fusion*. *State-ϵ* is omitted because of the very low number of extracted ARs. In Fig. 3, for each agent, the number of extracted ARs in one episode is counted without considering whether each AR is already extracted or not. Thus, the sum of numbers of ARs extracted by the agent over all the episodes is the total cumulative number of ARs. As shown in Fig. 3, the numbers of ARs extracted by the non-training agents are constantly low over episodes, and they are nearly invisible. On the other hand, *GIM-RL-Basic* is trained to extract ARs during about the first 50 episodes, and afterwards keeps extracting ARs although the extraction is unstable as illustrated by the significantly varied numbers of ARs over episodes. Compared to this, *GIM-RL-Fusion* aided by s'_k is trained to extract ARs in the first 200 episodes, and continues to stably extract the large numbers of ARs.

B. RESULTS FOR AGENT TRANSFER

We investigate agent transfer where an agent trained on a source dataset is transferred into another agent for a target dataset which is related to the source one. For this purpose, each of the datasets in Table 1 is split into two parts, especially, the first 60% of transactions and the remaining 40% constitute the source and target partitions, respectively. As shown in Table 5, one characteristic is that the number of distinct items in the source partition (M_{src}) is different from the one in the target partition (M_{tgt}). That is, some items are

included only in the source or target partition. We perform and test agent transfer between these source and target partitions.

TABLE 5. The difference between the number of distinct items in the source partition (M_{src}) and the one in the target partition (M_{tgt}) for each of the experimental datasets.

	Chess	Mushroom	Accidents_10	Connect	Pumsb
M_{src}	72	78	325	127	1981
M_{tgt}	75	106	308	129	1951

Let DQN_{src} and DQN_{tgt} be agents that are defined for the source and target partitions of a dataset, respectively. Based on the experimental results in the previous section, both of DQN_{src} and DQN_{tgt} are based on *GIM-RL-Fusion* that stably yields high performances. In addition, as with the previous section, DQNs with the same structure are used for extracting HUIs and ARs, and simpler DQNs are for FIs. Our idea of agent transfer is very simple. All the parameters of DQN_{src} trained on the source partition are transferred to DQN_{tgt}, except the first and output layers that need to treat the difference of distinct items between the source and target partitions. That is, DQN_{src} and DQN_{tgt} have the same structure with the same parameters except their first and output layers. Each unit in DQN_{src}'s first layer has a weight for each item. More precisely, it is used to weight s_k's value, which represents the usefulness of changing the item's inclusion in $X(\boldsymbol{b}_k)$ (the itemset defined by $\boldsymbol{b}_k$). Thus, weights of DQN_{src}'s first layer for items that are included in both the source and target partitions are replicated on DQN_{tgt}'s first layer. On the other hand, weights of DQN_{src}'s first layer for items included only in the source partition, are discarded. For items included only in the target partition, new randomly initialised weights are added to DQN_{tgt}'s first layer. The same replication, discarding and initialisation of weights are carried out for DQN_{src}'s and DQN_{tgt}'s output layers, which individually output $\boldsymbol{q}_k$ to select an action for changing the inclusion of an item in $X(\boldsymbol{b}_k)$. After the above-mentioned agent transfer, DQN_{tgt} is retrained on the target partition. Assuming that the source and target partitions have similar relations among items and parameters transferred from DQN_{src} to DQN_{tgt} are useful for
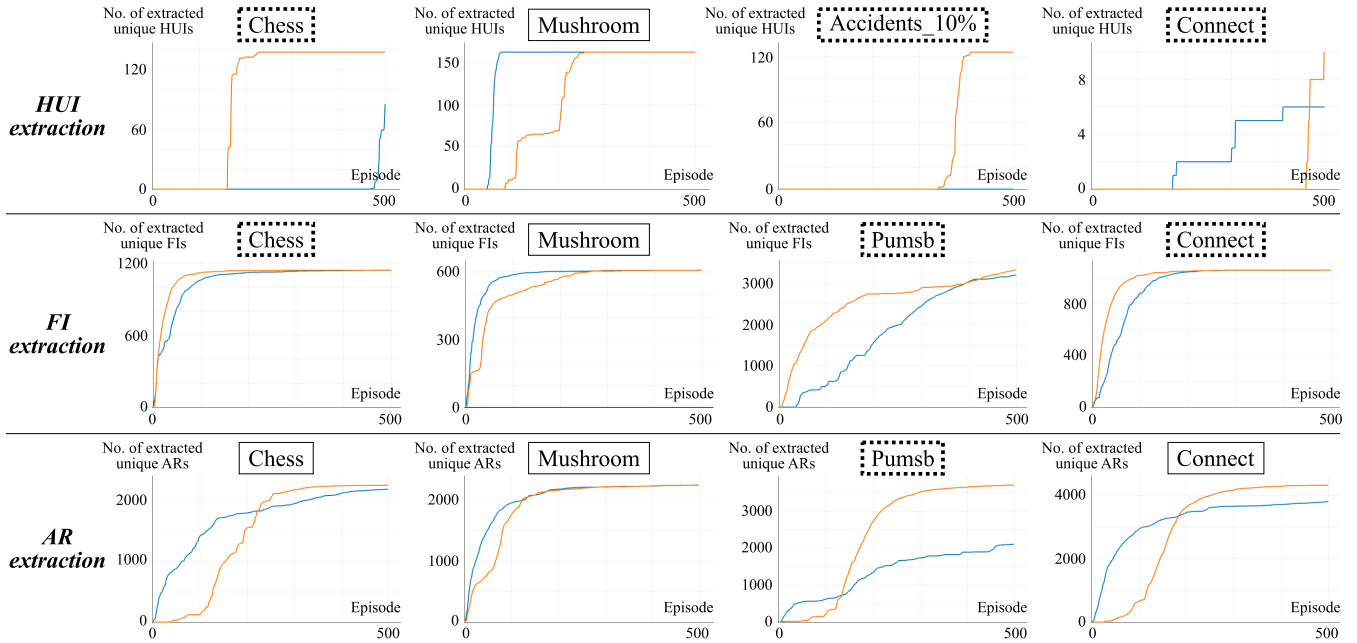
FIGURE 4. A comparison between the increase of the number of unique itemsets extracted by a transferred agent DQN_{tgt} (orange) and the one by an ordinarily trained agent $DQN_{scratch}$ (blue) over the passage of 500 episodes for each dataset and each itemset extraction task.

capturing those relations, retraining DQN_{tgt} is expected to be much faster than training a DQN from scratch on the target partition. For simplicity, the latter DQN is called $DQN_{scratch}$.

In Fig. 4, DQN_{tgt} is compared to $DQN_{scratch}$ on the target partition of each dataset. Here, the increase of the number of unique itemsets extracted by DQN_{tgt} over 500 episodes is plotted in orange, and such an increase for $DQN_{scratch}$ is drawn in blue. That is, Fig. 4 visualises how fast each of DQN_{tgt} and $DQN_{scratch}$ is trained to extract itemsets in the target partition. As illustrated in this figure, in all the 12 cases, DQN_{tgt} finally extracts more itemsets or at least the same number of itemsets to $DQN_{scratch}$. Especially, the 7 cases depicted by the dotted-line rectangles objectively indicate that DQN_{tgt} extracts many itemsets of a target type more quickly than $DQN_{scratch}$. This suggests the potential effectiveness of agent transfer to realise DQN_{tgt}'s efficient itemset extraction with the help of DQN_{src}.

In addition, let us focus on Chess, Accidents_10%, Connect in the HUI extraction and Pumsb and Connect in the AR extraction in Fig. 4. In each of these cases, DQN_{tgt} extracts no or few itemsets at the beginning, but once it starts to extract itemsets, it becomes to extract many itemsets very fast. This can be thought as follows: Although parameters transferred from DQN_{src} to DQN_{tgt} are useful, they are not directly compatible with a target partition. But, once these parameters are adapted to the target partition, their usefulness brings in extracting many itemsets very quickly. To improve agent transfer in terms of this parameter adaptation, we plan to explore model-based RL that uses an internal model summarising an environment (i.e., dataset) and adaptively updates this model depending on changes of the environment [33].

C. RUNTIME ANALYSIS

Table 6 presents the runtimes of *GIM-RL-Fusion* for each itemset extraction task. These runtimes are measured on a desktop PC equipped with Intel Core i9-9900K (3.60GHz), 32GB RAM and NVIDIA GeForce RTX 2080Ti. *GIM-RL-Fusion* implemented with Pytorch is run on Ubuntu 20.04-LTS. All the source codes for *GIM-RL-Fusion* as well as the other agents are available on our Github repository, as described in Appendix C. As can be seen from Table 6, *GIM-RL-Fusion* requires at least more than 30 minutes to finish one task. But, we believe that this slowness is surpassed by *GIM-RL-Funsion*'s great flexibility that any type of itemsets can be extracted as long as a reward for the type can be defined. In other words, *GIM-RL-Funsion* can extract any user-defined type of itemsets with no need to develop a specialised data structure or algorithm for the type.

TABLE 6. *GIM-RL-Fusion*'s runtimes expressed in the form of "hh:mm:ss." Column "Acc. / Pumsb" means that Accident_10% is used for the HUI extraction, and Pumsb is used for the FI and AR extractions.

Type	Chess	Mushroom	Acc. / Pumsb	Connect
HUI	00:34:09	00:31:42	04:40:53	12:45:10
FI	00:31:56	00:33:34	00:50:06	00:37:29
AR	00:50:20	00:50:10	01:07:26	00:55:21

The main reason for *GIM-RL-Fusion*'s slow runtime is the need of scanning a dataset to compute a state s_k and a reward r_k, as illustrated by the non-filled arrows in Fig. 1. Here, s_k and r_k are computed at each step and *GIM-RL-Fusion* (or more generally GIM-RL) involves E episodes each of which consists of K steps, as seen from Algorithm 1 in Appendix A. Thus, s_k and r_k are computed in total EK times.

Furthermore, M values in s_k are based on interestingness measure values for different itemsets, each of which is created by changing one item's inclusion in the itemset $X(b_k)$ defined by the current bit-vector b_k. Since the computation of an interestingness measure value requires one scan of the dataset, the computation of s_k involves M scans. Moreover, r_k needs the interestingness measure value for the updated itemset $X(b_{k+1})$, so its computation also involves one dataset scan. Hence, the computation of s_k and r_k at each step demands $M + 1$ dataset scans. Letting *scan_time* be an approximate time required for one scan of the dataset, the computational complexity of *GIM-RL-Fusion* can be expressed as $O(EK(M + 1) \times scan\_time)$. Compared to *scan_time*, the other processes shown in Algorithm 1 of Appendix A require negligible times.

The reduction of *scan_time* is obviously crucial for speeding up *GIM-RL-Fusion*. But, we will not develop a specialised data structure or algorithm to accomplish fast scan of a dataset. Instead, inspired by the work in [34], we will explore to encode the dataset into a neural network. Its input is an itemset given as a query, and the output is an approximate interestingness measure value for the itemset. This kind of speed-up of dataset scan is especially essential to extend *GIM-RL-Fusion* for more complex patterns like sequential or trajectory patterns. While an itemset can be defined by a bit-vector, the representation of a sequential pattern needs a "bit-matrix" where one row is a bit-vector representing an itemset observed at one time point, and the one of a trajectory pattern requires a "bit-tensor" where each bit-matrix at one time point indicates the x-y location of an object. The above-mentioned dataset encoding is necessary for fast querying a dataset in terms of these sequential or trajectory patterns.

V. CONCLUSION AND FUTURE WORK

In this paper, we introduced GIM-RL that offers a unified RL framework to extract various types of itemsets only by changing a reward definition. The general effectiveness of GIM-RL is verified through the experiments on the HUI, FI and AR extractions. The experimental results also suggest one remarkable potential of GIM-RL, namely agent transfer, which realises efficient itemset mining on a dataset with the help of an agent trained on another related dataset.

Before moving to detailed descriptions of our future work, let us clean up the main concept of how GIM-RL is formulated in the framework of RL by referring to Fig. 5. GIM-RL presented in this paper is based on the most basic RL where an environment produces a state s_k based on which an agent takes an action a_k. Then, from the environment, the agent receives a reward r_k as an evaluation of taking a_k at s_k as well as the updated state s_{k+1} to take the next action a_{k+1}. Under this iterative interaction, RL aims to train the agent so that it can take a sequence of actions to maximise cumulative rewards. GIM-RL is formulated by defining the following s_k, a_k and r_k, as highlighted in bold italic font in Fig. 5. First, an itemset is defined by a bit-vector indicating the inclusion
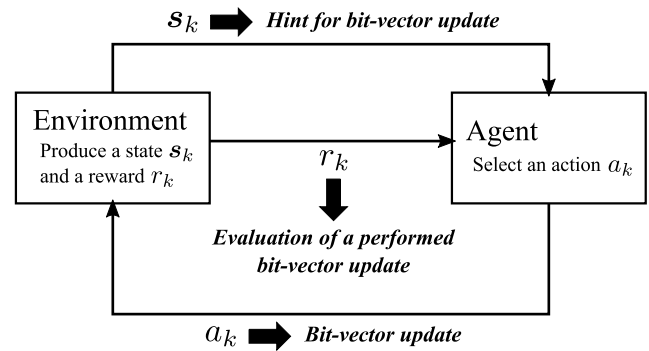

FIGURE 5. A conceptual illustration of how GIM-RL is formulated in the framework of RL.

of each of distinct items. Based on this, s_k is defined as a hint for how to update the bit-vector to form an itemset of a target type. In addition, a_k is defined as an update of the bit-vector based on s_k, and r_k is an evaluation of whether the updated bit-vector expresses an itemset of the target type. According to the above-mentioned formulation, GIM-RL trains an agent that maximises cumulative rewards, namely extracts as many itemsets of the target type as possible.

Since our main purpose in this paper is to empirically verify the possibility and potential of GIM-RL, its current implementation is simple and involves many research topics that need further investigations in the future. Below, some of them are outlined in association with the components of GIM-RL's formulation in Fig. 5. As an extension of an environment, dataset encoding described in Section IV-C aims to speed-up the computation of s_k and r_k. Besides, model-based RL in Section IV-B focuses on building a model that simulates the dynamics of an environment to generate "virtual" experiences, which are useful for faster agent training. Also, an agent can be enhanced by adopting a planning mechanism to infer itemsets that will be possibly obtained at the multiple-step-ahead future [35]. Moreover, an agent can be extended to take continuous-valued actions to extract itemsets that consist of items representing quantitative features (e.g., age, height and weight).

Apart from an environment and agent, it is also an important topic to design a new or more sophisticated state, action and reward. Especially, one invaluable attempt is to design a reward for extracting a previously unexplored type of itemsets. Another key topic is the adoption or development of a more advanced RL algorithm compared to Q-learning used in this paper (although the components in Fig. 5 are not explicitly associated with this topic). For example, training more accurate $Q(s, a; \Theta)$ is possible using double Q-learning where the selection and evaluation of an action are done by separate neural networks [36]. In addition, a policy gradient method that directly learns an action selection policy without relying on $Q(s, a; \Theta)$ is useful for training an agent that takes continuous-valued actions [9]. Finally, we believe that GIM-RL opens a new research direction towards so-called "learning-based itemset mining" which involves many interesting topics described above.

APPENDIX A
PSEUDO-CODE OF GIM-RL

Algorithm 1 shows a pseudo-code of GIM-RL to train an agent $Q(s, a; \Theta)$ that extracts itemsets satisfying $\varphi(X) \geq \xi$ from a dataset $\mathcal{D}$. Overall, as expressed by the double for-loop in lines 6-22, the agent is trained through E episodes consisting of K steps. As seen at lines 10-12, at each step, the agent receives a state s_k and computes q_k to decide an action a_k for updating the bit-vector b_k into b_{k+1}. After checking the quality of the itemset $X(b_{k+1})$ defined by b_{k+1} at lines 13-15, the environment generates a reward r_k and a new state s_{k+1} at lines 16 and 17. The tuple (s_k, a_k, r_k, s_{k+1}) is then stored into a replay memory $\mathcal{P}$ as one experience of the agent at line 18. Subsequently, the agent's parameters Θ are updated at line 19, while as written at line 21 the target network's parameters Θ^- are updated every e^- episodes by copying Θ to Θ^-. For *GIM-RL-Fusion* in Section IV, q_k at lines 10 and 11 is replaced with $q'_k = \lambda s'_k + (1 - \lambda)q_k$.

Algorithm 1 GIM-RL (Generic Itemset Mining Based on Reinforcement Learning)

Input Dataset $\mathcal{D}$, interestingness measure $\varphi(X)$, threshold ξ
Output A set $\mathcal{X}$ containing itemsets meeting $\varphi(X) \geq \xi$

1: Initialise $Q(s, a; \Theta)$ with He's parameter initialisation [37]
2: Initialise a target network as $Q(s, a; \Theta^-) = Q(s, a; \Theta)$
3: Initialise a replay memory as $\mathcal{P} \leftarrow \{\}$
4: Filter out items that individually have no possibility to be an element of itemsets of the target type.
5: $\mathcal{X} \leftarrow \{\}$
6: **for** $e = 1, \cdots E$ **do**
7:     Randomly initialise the bit-vector b_1
8:     Generate the first state s_1 based on b_1 and $\mathcal{D}$
9:     **for** $k = 1, \cdots K$ **do**
10:         Compute q_k by feeding s_k into $Q(s, a; \Theta)$
11:         Decide an action a_k by the ϵ-greedy strategy on q_k
12:         Update b_k into b_{k+1} by a_k
13:         **if** $\varphi(X(b_{k+1})) \geq \xi$ **then**
14:             $\mathcal{X} \leftarrow \mathcal{X} \cup \{X(b_{k+1})\}$ // An itemset is extracted
15:         **end if**
16:         Compute a reward r_k based on b_{k+1} and $\mathcal{D}$ (Section III-B)
17:         Compute s_{k+1} based on b_{k+1} and $\mathcal{D}$
18:         $\mathcal{P} \leftarrow \mathcal{P} \cup \{(s_k, a_k, r_k, s_{k+1})\}$
19:         Update Θ of $Q(s, a; \Theta)$ using experiences randomly sampled from $\mathcal{P}$
20:     **end for**
21:     Update Θ^- of $Q(s, a; \Theta^-)$ as $\Theta^- = \Theta$ (every e^- episodes)
22: **end for**
23: **return** $\mathcal{X}$

We mention the following three implementation details: First, the search space reduction before extracting itemsets is done at line 4. Specifically, for the extraction of HUIs, items whose upper bound utilities (transaction weighted utilisations [5], [6], [10], [11]) are less than the threshold ξ are discarded, and for the extractions of FIs and ARs, items whose supports are less than the minimum support threshold (ξ for FIs, and ξ_1 for ARs) are eliminated. Second, at line 7, b_1 in each episode is obtained by the random bit-vector initialisation described in Section III-A. For this initialisation, we assume that the more frequently an item occurs in $\mathcal{D}$, the more likely it is to be included in HUIs, FIs or ARs. The above-mentioned search space reduction and random bit-vector initialisation need to be modified when targeting another type of itemsets like infrequent itemsets. Last, to help the agent explore a variety of itemsets, line 11 shows that a_k is decided using the ϵ-greedy strategy. Here, a_k is chosen as the action corresponding to the highest value in q_k with the probability $1 - \epsilon$, while a_k is set to the action to change the value of a randomly selected dimension in b_k with the probability ϵ.
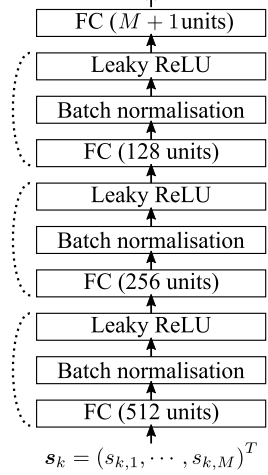
APPENDIX B
DQN ARCHITECTURES

Figs. 6 (a) and (b) show the DQN architecture for extracting HUIs and ARs and the one for FIs, respectively. These architectures are common with respect to the input and output layers. The input layer accepts an M-dimensional state vector $s_k = (s_{k,1}, \cdots, s_{k,M})^T$ where $s_{k,m}$ ($1 \leq m \leq M$) represents an estimated usefulness of changing the mth item's inclusion in the itemset $X(b_k)$. The output layer produces an $(M+1)$-dimensional vector $q_k = (q_{k,1}, \cdots, q_{k,M+1})^T$ where $q_{k,m}$ ($1 \leq m \leq M$) and $q_{k,M+1}$ indicate an approximate quality of the action to change the mth item's inclusion and the one for the random bit-vector initialisation, respectively. As illustrated by the three dotted arcs in Fig. 6 (a), the DQN for the HUI and AR extractions has three blocks each of which consists of a Fully-Connected (FC) layer, a batch normalisation layer and an activation layer defined by leaky ReLU (Rectified Linear Unit). Similarly, as shown in Fig. 6 (b), the DQN for the FI extraction is comprised of one block with an FC layer having a large number of units. Based on our preliminary experiments, the structural complexity of a DQN may be related to the complexity of a target type. That is, FIs are simpler than HUIs and ARs, so a simpler DQN seems enough for extracting FIs compared to the one for HUIs and ARs.

We describe details of how to compute $s_{k,m}$ in s_k. First of all, $\varphi(X)$ has a huge value range. For example, assuming that $\varphi(X)$ outputs the support of an itemset X, one itemset may have a support of 1000 while the support of a rare itemset may be 1. Thus, if $s_{k,m}$ is defined directly as $\varphi(X(b'_k))$ where $X(b'_k)$ is the itemset created by changing the mth item's inclusion in $X(b_k)$ (please see Section III-A), $s_{k,m}$'s value range can be huge, and s_k is often very biased in the sense that only some dimensions have very large values. To alleviate this bias, $s_{k,m}$ is computed by normalising $\varphi(X(b'_k))$ as follows:

$$s_{k,m} = \log\left(\frac{\varphi(X(b'_k))}{Z} + 1\right), \qquad (4)$$

a) HUI and AR extractions

$$\boldsymbol{q}_k = (q_{k,1}, \cdots, q_{k,M}, q_{k,M+1})^T$$

| FC ($M+1$ units) |
| Leaky ReLU |
| Batch normalisation |
| FC (128 units) |
| Leaky ReLU |
| Batch normalisation |
| FC (256 units) |
| Leaky ReLU |
| Batch normalisation |
| FC (512 units) |

$$\boldsymbol{s}_k = (s_{k,1}, \cdots, s_{k,M})^T$$

b) FI extraction

$$\boldsymbol{q}_k = (q_{k,1}, \cdots, q_{k,M}, q_{k,M+1})^T$$

| FC ($M+1$ units) |
| Leaky ReLU |
| Batch normalisation |
| FC (4096 units) |

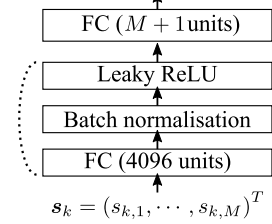$$\boldsymbol{s}_k = (s_{k,1}, \cdots, s_{k,M})^T$$

FIGURE 6. DQN architectures used for the HUI, FI and AR extractions.

where Z is a normalisation factor to put $\varphi(X(\boldsymbol{b}_k'))/Z$ between 0 and 1. For this, using Z is common in itemset mining, for instance, in FI extraction, Z is the number of transactions in $\mathcal{D}$ to compute the "relative" support of an itemset [1], [3], [4], and in HUI extraction, relative utilities are calculated by setting Z to the sum of utilities for all transactions (i.e., sum of transaction utilities) in $\mathcal{D}$ [6], [10], [11], [24]. These definitions of Z are also used in Eq. 4, which then takes log of $(\varphi(X(\boldsymbol{b}_k'))/Z + 1)$ to further reduce value differences among $s_{k,1} \cdots, s_{k,M}$ in $\boldsymbol{s}_k$.

One natural extension of $\boldsymbol{s}_k$ is performed for AR extraction involving two interestingness measures $\varphi_1(X)$ and $\varphi_2(X)$ that output the support and confidence of X, respectively. Specifically, we compute a $2M$-dimensional state vector $\boldsymbol{s}_k$. Here, $s_{k,m}$ in the first M dimensions is computed based on $\varphi_1(X(\boldsymbol{b}_k'))$ and Z being the number of transactions in $\mathcal{D}$. On the other hand, $s_{k,m}$ in the last M dimensions is computed based on $\varphi_2(X(\boldsymbol{b}_k'))$ and $Z = 1$, because $\varphi_2(X(\boldsymbol{b}_k'))$ representing the confidence of the rule defined by $X(\boldsymbol{b}_k)$ and $X(\boldsymbol{b}_k')$ already lies between 0 and 1. Finally, $\boldsymbol{s}_k$ is fed into a batch normalisation layer to normalise values by $\varphi_1(X(\boldsymbol{b}_k'))$ and $\varphi_2(X(\boldsymbol{b}_k'))$, and the resulting normalised $\boldsymbol{s}_k$ is used as the input of the DQN architecture in Fig. 6 (a). For *GIM-RL-Fusion*, the first M dimensions based on $\varphi_1(X(\boldsymbol{b}_k'))$ are used to create $\boldsymbol{s}_k'$.

APPENDIX C
HYPER-PARAMETER SETTING

Referring to Algorithm 1 in Appendix A, the number of episodes E is set to 500 for the HUI extraction and 1000 for the FI and AR extractions in Section IV-A, and to 500 for all the experiments of agent transfer in Section IV-B. The number of steps in one episode is always $K = 500$. The size of a replay memory $\mathcal{P}$ is set to 10000. Each update of an agent's parameters Θ is based on Eq. 2 where the

discount factor γ is set to 0.95. This update of Θ is done using RAdam [38] (with the initial learning rate $\alpha = 0.001$) as an optimiser on a mini-batch of 512 experiences randomly sampled from $\mathcal{P}$. Furthermore, e^- which is the episode period to update a target network's parameters Θ^- is set to 5, that is, Θ^- is updated every $e^- = 5$ episodes.

λ in $\boldsymbol{q}_k' = \lambda \boldsymbol{s}_k' + (1-\lambda)\boldsymbol{q}_k$ of *GIM-RL-Fusion* is dynamically changed. As described in Section IV-A, at the beginning of training an agent, λ is large in order to facilitate finding positive itemsets based on $\boldsymbol{s}_k'$, and is gradually reduced to prioritise the agent's exploitation based on the trained $Q(\boldsymbol{s}, a; \Theta)$. Inspired by the ϵ-greedy strategy, we implement this dynamic change of λ as follows:

$$\lambda = \lambda_{end} + (\lambda_{start} - \lambda_{end}) \exp\left(-\frac{k_{total}}{\Delta}\right), \qquad (5)$$

where λ_{start} and λ_{end} are the maximum and minimum values of λ, respectively. k_{total} is the total number of steps the agent has experienced, that is, $k_{total} = e \times K + k$ for the kth step in the eth episode. In Eq. 5, λ is λ_{start} when $k_{total} = 0$, and gradually converges to λ_{end} as k_{total} increases. Δ is a hyper-parameter to control the speed of this convergence.

First, Δ is set to 200 in all the experiments in Section IV. In Section IV-A, $\lambda_{start} = 0.999$ and $\lambda_{end} = 0.5$ are used for extracting HUIs and ARs, and $\lambda_{start} = 0.999$ and $\lambda_{end} = 0.6$ are for FIs. For these settings of λ_{start} and λ_{end}, we think that $\boldsymbol{s}_k$ provides "dataset-dependent" information for changing each item's inclusion in $X(\boldsymbol{b}_k)$ by actually checking transactions in a dataset, while $\boldsymbol{q}_k$ represents more general information obtained from a DQN. According to this thought, HUIs and ARs are more complex than FIs, so the smaller $\lambda_{end} = 0.5$ is used to put a higher priority on $\boldsymbol{q}_k$.

For the experiments of agent transfer in Section IV-B, λ_{start} and λ_{end} are set depending on source and target datasets, and itemset types. For the HUI extraction, $\lambda_{start} = 0.999$ and $\lambda_{end} = 0.5$ are used to train an agent on a source dataset, and $\lambda_{start} = \lambda_{end} = 0.5$ (i.e., λ is constant at 0.5) is used on a target dataset in order to take more advantage of $\boldsymbol{q}_k$ obtained from the source dataset. For the FI extraction, $\lambda_{start} = 0.999$ and $\lambda_{end} = 0.6$ are used for both source and target datasets, to take more account of dataset-dependent information for the target dataset. For the AR extraction, we use $\lambda_{start} = \lambda_{end} = 0.5$ on a source dataset to obtain general information about items as $\boldsymbol{q}_k$, and then $\lambda_{start} = 0.999$ and $\lambda_{end} = 0.5$ are used to gradually adapt $\boldsymbol{q}_k$ to a target dataset by considering dataset-dependent information. We will explore a more sophisticated approach to define λ_{start} and λ_{end} based on statistical characteristics of source and target datasets.

Finally, all the source codes (including the codes for data download) used in this paper are available on our Github repository.[4]
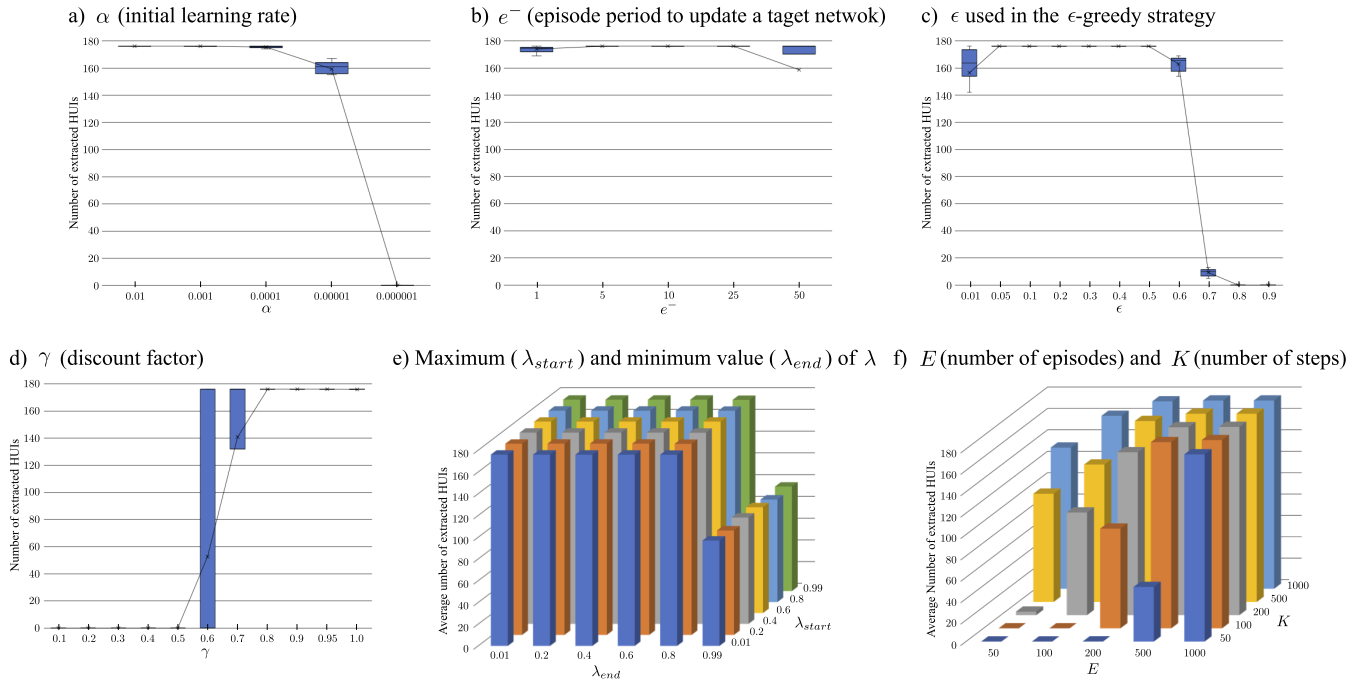
---

[4] https://github.com/Wisteria30/GIM-RL

FIGURE 7. Transitions of numbers of HUIs extracted from Chess by changing one or two hyper-parameters of *GIM-RL-Fusion*.

APPENDIX D
EFFECTS OF HYPER-PARAMETERS

This section is devoted to examining effects of GIM-RL's hyper-parameters on itemset mining results. Due to the computational cost and the presentation brevity, the following experiments especially address the HUI extraction on Chess, where the maximum number of extractable HUIs is 176 based on the threshold ξ in Table 2 of Section IV-A. In addition, *GIM-RL-Fusion* is particularly used because it stably achieved the top or nearly top performance on all the experiments.

Fig. 7 illustrates transitions of numbers of HUIs extracted by changing one or two hyper-parameters of *GIM-RL-Fusion*. Note that all hyper-parameters except the changed ones are kept the same to the ones in Section IV-A (please see the previous section for the concrete values of the unchanged hyper-parameters). Each of Figs. 7 (a) to (d) presents a transition of numbers of HUIs extracted using hyper-parameter values shown on the horizontal axis. Here, considering the randomness involved in *GIM-RL-Fusion*, a box-plot is adopted by running it 10 times for each hyper-parameter value. The line is drawn by connecting the average numbers of HUIs extracted in 10 runs for neighbouring hyper-parameter values.[5] Figs. 7 (e) and (f) employ three-dimensional histograms that are individually based on hyper-parameter pairs defined by the bottom plane. The vertical axis of each histogram indicates the average of numbers of HUIs extracted in 10 runs

with each hyper-parameter pair (the standard deviation is omitted for the presentation brevity).

Let us focus on Figs. 7 (b) regarding e^- to handle the episode period for updating a target network, (c) regarding ϵ to control the probability of a random action in the ϵ-greedy strategy, and (e) regarding λ_{start} and λ_{end} to adjust the mixing weight between an extended state s'_k and a vector of estimated action qualities q_k. For these hyper-parameters, all the 176 HUIs are extracted as long as they are set to moderate values like 5-25 for e^-, 0.05-0.5 for ϵ and 0.01-0.8 for λ_{start} and λ_{end}. The initial learning rate α in Fig. 7 (a) needs special attention like usual training of a neural network. Fig. 7 (a) indicates that setting α to a very small value slows down agent training, consequently a much smaller number of HUIs can be extracted in the same episode and step numbers. Fig. 7 (d) treats a discount factor γ to control the degree of lowering rewards at future steps. More intuitively, γ manages the future range that an agent considers to select an action at the current step. Fig. 7 (d) exhibits that more HUIs are extracted as the further future is considered using larger γ. For Fig. 7 (f) concerning E and K to control the episode and step numbers, it is clear that the number of HUIs decreases as these hyper-parameters are set to smaller values. On the other hand, setting them to larger values leads to extracting all the HUIs although it lengthens the runtime of *GIM-RL-Fusion*.

In Fig. 7, all the HUIs are extracted by multiple values of a hyper-parameter. Thus, more detailed analysis is conducted by adopting another evaluation criteria about the efficiency of itemset mining. Based on Section IV-C, when targeting a particular dataset, the time needed for each step of GIM-RL is assumed to be nearly the same, because the number of

---

[5]The line lying outside the box for $e^- = 50$ in Fig. 7 (b) is caused by one exceptionally small number of extracted HUIs. In addition, the long box for $\gamma = 0.6$ in Fig. 7 (d) results from the mixture of runs where all the HUIs are extracted and the ones where no HUI is extracted.
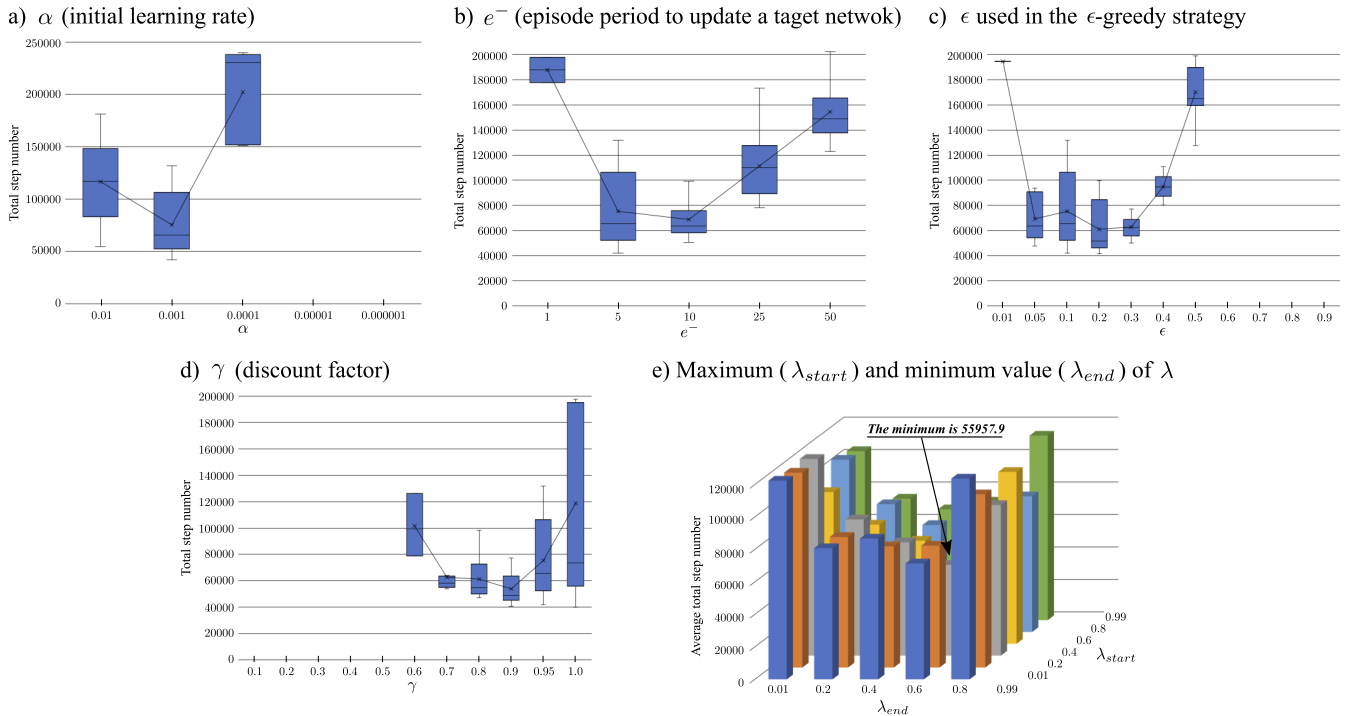
FIGURE 8. Transitions of total step numbers required to extract all the HUIs in Chess by changing one or two hyper-parameters of *GIM-RL-Fusion*.

distinct items M is fixed and the time *scan_time* required for one dataset scan should be approximately constant. Hence, we define the efficiency of GIM-RL's itemset mining as the total number of steps required to extract a certain number of itemsets, especially, all the itemsets. For simplicity, this number is abbreviated as a *total step number*. To be precise, it is computed as $e' \times K + k'$ if all the itemsets are extracted at the k'th step in e'th episode.

Fig. 8 shows transitions of total step numbers required to extract all the HUIs in Chess by changing each of α, e^-, ϵ, γ and the pair of λ_{start} and λ_{end} (a graph for E and K is not created because they obviously have no influence on total step numbers). Since a total step number can be calculated only in the case where all the HUIs are extracted, Fig. 8 focuses on hyper-parameter values leading to this case in Fig. 7. That is, nothing is depicted in Fig. 8 for hyper-parameter values with which all the HUIs are not extracted in Fig. 7. In each of Figs. 8 (a) to (d), *GIM-RL-Fusion* is run 10 times and a box-plot of total step numbers is drawn based on the runs where all the HUIs are extracted. The three-dimensional histogram in Fig. 8 (e) displays the average of total step numbers obtained in 10 runs using each pair of λ_{start} and λ_{end}.

Fig. 8 shows that even if multiple hyper-parameter values result in extracting all the HUIs, total step numbers for them are quite different. That is, GIM-RL's runtime to extract all the itemsets significantly varies depending on hyper-parameter values. And, analysis of total step numbers is useful for finding hyper-parameter values with which many itemsets can be efficiently extracted in a short runtime. With respect to this, it is often infeasible or takes very long time to extract all the itemsets in a huge real-world dataset, so

TABLE 7. A list of abbreviations.

Abbreviation	Expansion
FI	Frequent Itemset
AR	Association Rule
HUI	High Utility Itemset
FP-tree	Frequent Patten tree
FP-growth	Frequent Pattern growth
RL	Reinforcement Learning
GIM-RL	Generic Itemset Mining based on Reinforcement Learning (our proposed approach)
DQN	Deep Q-Network
EC	Evolutionary Computation
GA	Genetic Algorithm
SPMF	Sequential Pattern Mining Framework
FC layer	Fully-Connected layer
leaky ReLU	leaky Rectified Linear Unit

computing total step numbers for different hyper-parameter values is impossible or impractical. Nevertheless, random sampling of transactions in the original dataset can be performed to create a moderate-size dataset for which total step numbers are computed reasonably. This way, even for real-world datasets, it is possible to tune hyper-parameters so as to find a good tradeoff between GIM-RL's runtime (i.e., total step number) and the number of extracted itemsets.

APPENDIX E
SMALL REMARK ABOUT MUSHROOM
The Web page[6] of SPMF library [32] provides two versions of Mushroom, one for the HUI extraction and the other for the FI and AR extractions. Regarding the number of distinct items

[6] http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php

TABLE 8. A list of mathematical symbols defined to explain the framework of GIM-RL in Section III.

Symbols related to itemset mining

Symbol	Description
$\mathcal{D}$	A dataset containing N transactions, $\mathcal{D} = \{T_1, \cdots, T_N\}$
N	The number of transactions in $\mathcal{D}$
$\mathcal{I}$	The set of M distinct items in $\mathcal{D}$, $\mathcal{I} = \{i_1, \cdots, i_M\}$
M	The number of items in $\mathcal{I}$
T_n	The nth transaction in $\mathcal{D}$, $T_n = \{i_{n,1}, \cdots, i_{n,|T_n|}\} \subseteq \mathcal{I}$
$i_{n,l}$	The lth item in T_n, $i_{n,l} \in \mathcal{I}$
i_m	The mth item in $\mathcal{I}$
X	An itemset, $X \subseteq \mathcal{I}$
$\varphi(X)$	X's relevance to a target type (i.e., interestingness measure value)
ξ	A threshold to check the sufficiency of X's relevance (X is extracted if $\varphi(X) \geq \xi$)
$p(i_{n,l})$	The item-specific utility of $i_{n,l}$ in T_n to define HUIs
$q(i_{n,l})$	The quantity of $i_{n,l}$ in T_n to define HUIs
$sup(X)$	X's support (frequency) to define FIs and ARs
X_a	An itemset defining the antecedent of an AR
X_c	An itemset defining the consequent of an AR
min_sup	A minimum support threshold to define FIs and ARs
min_conf	A minimum confidence threshold to define ARs

Symbols related to RL for itemset mining

Symbol	Description
$\boldsymbol{b}_k$	An M-dimensional bit-vector to define the itemset examined at the kth step, $\boldsymbol{b}_k = (b_{k,1}, \cdots, b_{k,M})^T$
$b_{k,m}$	The mth binary value in $\boldsymbol{b}_k$ representing the inclusion of the mth item i_m, $b_{k,m} \in \{0,1\}$
$X(\boldsymbol{b}_k)$	The itemset defined by $\boldsymbol{b}_k$
$\boldsymbol{s}_k$	A state defined as an M-dimensional vector suggesting how to change $\boldsymbol{b}_k$ to form an itemset of a target type, $\boldsymbol{s}_k = (s_{k,1}, \cdots, s_{k,M})^T$
$s_{k,m}$	The mth value in $\boldsymbol{s}_k$ implying the usefulness for changing $b_{k,m}$ in $\boldsymbol{b}_k$ (i.e., i_m's inclusion in $X(\boldsymbol{b}_k)$)
a_k	An action taken by an agent at the kth step to update $\boldsymbol{b}_k$ into $\boldsymbol{b}_{k+1}$, $a_k \in \mathcal{A}$
$\mathcal{A}$	A set of possible actions including the change of each item's inclusion and the random bit-vector initialisation
r_k	A reward indicating an evaluation score of taking a_k at $\boldsymbol{s}_k$
$Q^*(\boldsymbol{s}, a)$	The optimal Q function quantifying the quality of taking an action a at a state $\boldsymbol{s}$
K	The number of steps to update a bit-vector
γ	A discount factor to lower future rewards
π	A policy for action selection
$Q(\boldsymbol{s}, a; \Theta)$	A Q function approximated by a neural network (i.e., DQN) defined by a set of parameters Θ ($Q(\boldsymbol{s}, a; \Theta)$ is interchangeably called an agent)
$\boldsymbol{q}_k$	An $(M+1)$-dimensional vector collecting $Q(\boldsymbol{s}, a; \Theta)$s for all the $M+1$ actions at the kth step, $\boldsymbol{q}_k = (q_{k,1}, \cdots, q_{k,M+1})^T$
E	The number of episodes each consisting of K steps
$\mathcal{P}$	A replay memory storing an agent's recent experiences $(\boldsymbol{s}_k, a_k, r_k, \boldsymbol{s}_{k+1})$s
$Q(\boldsymbol{s}, a; \Theta^-)$	A target network with parameters Θ^- to compute the target value $r_k + \gamma \max_{a' \in \mathcal{A}} Q(\boldsymbol{s}_{k+1}, a'; \Theta^-)$

in a source partition M_{src} and the one in a target partition M_{tgt} in Table 5 of Section IV-B, these two versions have a small difference. Mushroom for the HUI extraction is divided into the source parition with $M_{src} = 98$ and the target one with $M_{tgt} = 109$, and the source and target partitions of Mushroom

for the FI and AR extractions are characterised by $M_{src} = 78$ and $M_{tgt} = 106$, respectively. Only the latter M_{src} and M_{tgt} are shown in Table 5 for ease of understanding.

APPENDIX F
ABBREVIATIONS AND SYMBOLS
Below, a list of abbreviations (Table 7) and two lists of mathematical symbols (Tables 8 and 9) are provided to make this paper easier to understand. Especially, Tables 8 and 9 are created by categorising symbols based on whether they are defined to explain the framework of GIM-RL in Section III or describe details of our implementation and experiments in and after Section IV.

TABLE 9. A list of mathematical symbols defined to describe details of our implementation and experiments in and after Section IV.

Symbol	Description
e^-	The episode period to update a target network
α	The initial learning rate in RAdam optimiser
ϵ	A probability that an agent takes a random action (ϵ is used in two similar contexts, please see $State$-ϵ in Section IV-A and GIM-RL in Appendix A)
$\boldsymbol{s}'_k$	An extended state created by appending to $\boldsymbol{s}_k$ one dimension for the random bit-vector initialisation
$\boldsymbol{q}'_k$	$\boldsymbol{q}_k$ combined with $\boldsymbol{s}'_k$ to advance the training of an agent, $\boldsymbol{q}'_k = \lambda \boldsymbol{s}'_k + (1-\lambda)\boldsymbol{q}_k$
λ	A dynamic weight to combine $\boldsymbol{q}_k$ with $\boldsymbol{s}'_k$ through the passage of training an agent
λ_{start}	The initial value of λ corresponding to its maximum
λ_{end}	The convergence value of λ corresponding to its minimum
k_{total}	The total number of steps executed until the kth step in the eth episode, $k_{total} = e \times K + k$
Δ	A factor to control the speed of λ's convergence
M_{src}	The number of distinct items in a source partition
M_{tgt}	The number of distinct items in a target partition
DQN_{src}	A DQN trained on a source partition
DQN_{tgt}	A DQN trained by transferring DQN_{src} to a target partition
$DQN_{scratch}$	A DQN trained on a target partition from scratch
$scan\_time$	A time required to scan a dataset
$\mathcal{X}$	A set of extracted itemsets matching a target type
Z	A normalisation factor to compute $s_{k,m}$

REFERENCES

[1] P. Fournier-Viger, J. C. Lin, B. Vo, T. T. Chi, J. Zhang, and H. B. Le, "A survey of itemset mining," *WIREs Data Mining Knowl. Discovery*, vol. 7, no. 4, p. e1207, Jul. 2017.

[2] L. Geng and H. J. Hamilton, "Interestingness measures for data mining: A survey," *ACM Comput. Surv.*, vol. 38, no. 3, pp. 1–32, Sep. 2006.

[3] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *Proc. VLDB*, 1994, pp. 487–499.

[4] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," *Data Mining Knowl. Discovery*, vol. 8, no. 1, pp. 53–87, 2004.

[5] C. F. Ahmed, S. K. Tanbeer, B. S. Jeong, and Y. K. Lee, "Efficient tree structures for high utility pattern mining in incremental databases," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 12, pp. 1708–1721, Dec. 2009.

[6] V. S. Tseng, C.-W. Wu, B.-E. Shie, and P. S. Yu, "UP-growth: An efficient algorithm for high utility itemset mining," in *Proc. 16th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2010, pp. 253–262.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," in *Proc. NIPS Deep Learn. Workshop*, 2013.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[9] V. Francois-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," 2018, *arXiv:1811.12560*.

[10] M. Liu and J. Qu, "Mining high utility itemsets without candidate generation," in *Proc. 21st ACM Int. Conf. Inf. Knowl. Manage. (CIKM)*, 2012, pp. 55–64.

[11] W. Song and C. Huang, "Discovering high utility itemsets based on the artificial bee colony algorithm," in *Proc. PAKDD*, 2018, pp. 3–14.

[12] P. Fournier-Viger, C.-W. Wu, and S. V. Tseng, "Mining top-k association rules," in *Proc. Can. AI*, 2012, pp. 61–73.

[13] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 1717–1724.

[14] F. Li, K. Shirahama, M. A. Nisar, X. Huang, and M. Grzegorzek, "Deep transfer learning for time series data based on sensor modality classification," *Sensors*, vol. 20, no. 15, p. 4271, Jul. 2020.

[15] C. Lucchese, S. Orlando, and R. Perego, "Fast and memory efficient mining of frequent closed itemsets," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 1, pp. 21–36, Jan. 2006.

[16] T. Uno, M. Kiyomi, and H. Arimura, "LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets," in *Proc. FIMI*, 2004, pp. 1–11.

[17] R. U. Kiran, A. Kotni, P. K. Reddy, M. Toyoda, S. Bhalla, and M. Kitsuregawa, "Efficient discovery of weighted frequent itemsets in very large transactional databases: A re-visit," in *Proc. Big Data*, 2018, pp. 723–732.

[18] L. Cagliero and P. Garza, "Infrequent weighted itemset mining using frequent pattern growth," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 4, pp. 903–915, Apr. 2014.

[19] Y. S. Koh and S. D. Ravana, "Unsupervised rare pattern mining: A survey," *ACM Trans. Knowl. Discovery Data*, vol. 10, no. 4, pp. 1–29, 2016.

[20] S. Darrab, D. Broneske, and G. Saake, "Modern applications and challenges for rare itemset mining," *Int. J. Mach. Learn. Comput.*, vol. 11, no. 3, pp. 208–218, May 2021.

[21] J. Pei, J. Han, and L. V. S. Lakshmanan, "Pushing convertible constraints in frequent itemset mining," *Data Mining Knowl. Discovery*, vol. 8, no. 3, pp. 227–252, May 2004.

[22] A. Telikani, A. H. Gandomi, and A. Shahbahrami, "A survey of evolutionary computation for association rule mining," *Inf. Sci.*, vol. 524, pp. 318–352, Jul. 2020.

[23] J. Mata, J.-L. Alvarez, and J.-C. Riquelme, "Discovering numeric association rules via evolutionary algorithm," in *Proc. PKDD*, 2002, pp. 40–51.

[24] W. Song and C. Huang, "Mining high utility itemsets using bio-inspired algorithms: A diverse optimal value framework," *IEEE Access*, vol. 6, pp. 19568–19582, 2018.

[25] M. Boley, S. Moens, and T. Gärtner, "Linear space direct pattern sampling using coupling from the past," in *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2012, pp. 69–77.

[26] V. Dzyuba, M. van Leeuwen, and L. De Raedt, "Flexible constrained sampling with guarantees for pattern mining," *Data Mining Knowl. Discovery*, vol. 31, no. 5, pp. 1266–1293, Sep. 2017.

[27] C. K.-S. Leung, Q. I. Khan, Z. Li, and T. Hoque, "Cantree: A canonical-order tree for incremental frequent-pattern mining," *Knowl. Inf. Syst.*, vol. 11, no. 3, pp. 287–311, Apr. 2007.

[28] T. Calders, N. Dexters, J. J. M. Gillis, and B. Goethals, "Mining frequent itemsets in a stream," *Inf. Syst.*, vol. 39, pp. 233–255, Jan. 2014.

[29] P. Y. Taser, K. U. Birant, and D. Birant, "Multitask-based association rule mining," *Turkish J. Elect. Eng. Comput. Sci.*, vol. 28, no. 2, pp. 933–955, 2020.

[30] G. I. Webb, "Filtered-top-k association discovery," *WIREs Data Mining Knowl. Discovery*, vol. 1, no. 3, pp. 183–192, May 2011.

[31] S. Surampudi. (2017). Oracle data mining concepts. Oracle. [Online]. Available: https://docs.oracle.com/database/121/DMCON/toc.htm

[32] P. Fournier-Viger, J. C.-W. Lin, A. Gomariz, T. Gueniche, A. Soltani, Z. Deng, and H. T. Lam, "The SPMF open-source data mining library version 2," in *Proc. PKDD*, 2016, pp. 36–40.

[33] D. Ha and J. Schmidhuber, "Recurrent world models facilitate policy evolution," in *Proc. NIPS*, 2018, pp. 2450–2462.

[34] I. Bello, H. Pham, V. Q. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," in *Proc. ICLR Workshop*, 2017.

[35] R. Pascanu, Y. Li, O. Vinyals, N. Heess, L. Buesing, S. Racanière, D. Reichert, T. Weber, D. Wierstra, and P. Battaglia, "Learning model-based planning from scratch," 2017, *arXiv:1707.06170*.

[36] H. V. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. IJCAI*, 2016, pp. 2094–2100.

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1026–1034.

[38] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, "On the variance of the adaptive learning rate and beyond," in *Proc. ICLR*, 2020.

KAZUMA FUJIOKA received the B.E. degree in engineering from Kindai University, Japan, in 2021, where he is currently pursuing the M.E. degree at the Graduate School of Science and Engineering, Kindai University. His research interests include deep reinforcement learning and data mining.

KIMIAKI SHIRAHAMA received the B.E., M.E., and D.E. degrees in engineering from Kobe University, Japan, in 2003, 2005, and 2011, respectively. After working as an Assistant Professor at the Muroran Institute of Technology, Japan, he worked as a Postdoctoral Researcher at the Pattern Recognition Group, University of Siegen, Germany, from 2013 to 2018. Since 2018, he has been working as an Associate Professor at Kindai University, Japan. His research interests include multimedia data processing, machine learning, data mining, and sensor-based human activity recognition. He is a member of ACM SIGKDD, ACM SIGMM, the Institute of Image Information and Television Engineers in Japan (ITE), Information Processing Society of Japan (IPSJ), and the Institute of Electronics, Information and Communication Engineering in Japan (IEICE).

. . .