Social Network Mining and Analytics for Quantitative Patterns

Connor C.J. Hryhoruk, Carson K. Leung (✉), Adam G.M. Pazdor
Department of Computer Science, University of Manitoba Winnipeg, MB, Canada
✉ Carson.Leung@UManitoba.ca

***Abstract*—Frequent pattern mining has become popular in big data analytics and knowledge discovery as it discovers sets of items (e.g., merchandise items or events) that co-occur frequently. These frequent patterns are discovered by either horizontally by transaction-centric mining algorithms or vertically by item-centric mining algorithms. Regardless of the mining algorithms used, traditional frequent pattern mining algorithms focus on discovering Boolean frequent patterns, which reveal the presence or absence of specific items within the discovered patterns (e.g., whether or not a social entity follows another social entity in a social network or social networking site). However, in many real-life scenarios, the quantities of items within the patterns are crucial. For instance, the quantity of followed items (e.g., followed or liked posts) can significantly impact the socializability of a social entity toward another social entity in a social network. A very recent quantitative algorithm called Q-VIPER mined frequent quantitative patterns from traditional shopper market data by representing the big data as a collection of item-centric bitmaps. Each bitmap captures the presence or absence of a transaction containing the item, together with the quantity of that item in each transaction. It then mines quantitative frequent patterns vertically. It works well with small quantity. However, when dealing with large quantity, it generates a large number of sets of candidate *quantitative frequent patterns* (aka sets of item expressions, or *itemexpsets* for short). Given that large quantities are not unusual in numerous real-life applications, we design a scalable solution in this paper. The resulting *scalable* quantitative frequent pattern algorithm called SQ-VIPER significantly reduces the number of candidates to be generated, and thus speeds up the mining process. Evaluation results show that superiority of our SQ-VIPER over the existing Q-VIPER and MQA-M algorithms, which respectively mine quantitative frequent patterns vertically and horizontally. This also demonstrates the usefulness of SQ-VIPER in social network mining and analysis.**

Keywords—data science, data mining, frequent patterns, quantitative patterns, quantity, vertical mining, bitmap, social network

I. Introduction

Nowadays, big data [1-4] are everywhere. Huge amount of data are generated or collected at a rapid rate. With so much data and only a finite amount of time, the need for some sort of system to filter through the masses and find what a user wants is abundantly apparent. This calls for data science system [5], which makes good use of techniques like data mining [6], visual analytics [7] and machine learning [8-11] for discovering useful knowledge from big data. This, in turn, helps make recommendations or decisions in various real-life application areas such as transportation analytics [12-16], health informatics [17-23], and social network analysis [24- 26].

As an important data mining task, *frequent pattern mining* [27-30] discovers from big data those frequently co-occurring sets of items (e.g., popular merchandise items, co-located events). These discovered frequent patterns can be served as building blocks in the formation of interesting association rules. Specifically, antecedents and consequences of association rules are frequent patterns. The rules reveal associative relationships or correlation between frequent patterns in the antecedents and consequences of the rules. These tasks of frequent pattern mining and association rule mining serve as the foundation for numerous real-life applications, including business marketing, biological pattern discovery, human population analysis, and web log mining.

Frequent patterns can discovered by either horizontally by transaction-centric mining algorithms or vertically by item-centric mining algorithms. For example, transaction-centric mining algorithms—like the Apriori algorithm [31, 32]—horizontally discover frequent patterns from a collection of transactions capturing the presence or absence of items in each transaction. As another example, item-centric mining algorithms—like VIPER (Vertical Itemset Partitioning for Efficient Rule-extraction) algorithm [33]—vertically discover frequent patterns from a collection of bitmaps capturing which transactions contain a specific item. Such a bitmap representation can be advantageous because the size of bitmap collection is independent of the density of the data. Instead, the size of bitmap only depends on the number of items and transactions. Dense data contain more 1s than 0s, and vice versa for sparse data. These bitmap based algorithms were observed to be efficient as they take advantage of bitwise operations in the mining process.

Traditional frequent pattern mining algorithms— regardless of their mining direction (horizontal or vertical)— discover Boolean frequent patterns capturing only the presence or absence of items within the discovered patterns. While traditional frequent pattern mining and association rule mining are useful in many contexts, they have a major limitation. This limitation is that in traditional frequent pattern mining, we assume that every transaction either contains an item or does not contain the item. In other words, an item is contained in a transaction 0 or 1 times. For this reason, we can also refer to traditional frequent pattern mining as *Boolean* frequent pattern mining. However, in many real-world scenarios, a transaction can contain an item more than one time. For example, a person at a grocery store may buy multiple apples. To address this shortcoming, the notion of quantitative association rule mining or *quantitative* frequent pattern mining [34, 35] was studied. Quantitative frequent pattern mining is essentially an extension of frequent pattern mining to allow transactions to contain an item more than once. Rather than just trying to find items (which commonly

ASONAM '23, November 6–9, 2023, Kusadasi, Turkey

ACM ISBN 979-8-4007-0409-3/23/11…$15.00
https://doi.org/10.1145/3625007.3627338

occur in transactions), there is a demand for discovering commonly occurring quantities of items. For example, in Boolean frequent pattern mining, we may discover that bananas are a frequently purchased item. In quantitative frequent pattern mining, we may discover that customers frequently purchase at least five bananas at a time. As another example, the quantity of items may also affect profits of selling the items within the discovered patterns.

Discovering quantitative frequent patterns and quantitative association rules would lead to more interesting results than we would with Boolean association rule mining. In addition to receiving information about which items commonly occur together in transactions, we also obtain information regarding how many of each of those items tend to occur in transactions. MQA-M algorithm [34] extends the Apriori algorithm to mine *quantitative frequent patterns* (aka *itemexpsets*) horizontally.

In 2022, a bitwise algorithm called Q-VIPER [36] was proposed to mine quantitative frequent patterns (i.e., itemexpsets). Such an algorithm represents the data as a collection of bitmaps captures the presence or absence of a transaction containing the item, as well as the quantity of that item in each transaction. With this representation, the algorithm then vertically mines quantitative frequent patterns. When compared the existing MQA-M algorithm (which was built for quantitative frequent pattern mining), their evaluation results show that Q-VIPER requires shorter runtime to mine frequent patterns. While Q-VIPER manages to mine quantitative frequent patterns and be able to mine patterns occur in relatively low quantities, its performance degrades when handling large quantities. This is partially due to the way the algorithm generates candidate itemexpsets. Unfortunately, it is not uncommon to involve large quantities in many real- life scenarios. Consequently, better solutions for mining frequent patterns with large (and small) quantities are in demand.

To response to the demand, we present in this paper a scalable solution—i.e., a vertical bitwise algorithm to mine quantitative frequent patterns (i.e., itemexpsets) vertically. The resulting *scalable* quantitative frequent pattern algorithm algorithm called SQ-VIPER significantly improves Q-VIPER so that SQ-VIPER is capable to mine frequent patterns with both large and small quantities.

Note that, in the context of social network mining and analysis, a transaction capturing purchased merchandise items in the traditional shopper market data can be considered as a record for a social entity who follows or likes another social entities' posting. Traditional Boolean frequent pattern mining helps reveal who likes who (e.g., a social entity A frequently like postings of social entities B and C). The quantitative frequent pattern mining helps reveal the degree of interest in the following/liking relationship. For example, it may reveal that the social entity A likes 30 (i.e., 60%) of social entity B's postings but only likes 5 (i.e., 10%) of social entity C's postings.

Our evaluation results on both benchmark shopper market data and Stanford SNAP social network data show that our SQ-VIPER algorithm requires shorter runtime than existing Q-VIPER and MQA-M when mining quantitative frequent patterns. Our *key contributions* in this paper include design and implementation of our SQ-VIPER algorithm, as well as its practical applications in social network mining and analysis.

We organize the rest of this paper as follows. Section II presents the mathematical framework for quantitative frequent pattern mining and discusses related algorithms (e.g., Apriori, MQA-M, VIPER). Section III introduces our SQ-VIPER algorithm. Section IV shows our evaluation results, and Section V draws the conclusions.

II. Background and Related Works

A. Vertical Boolean Frequent Pattern Mining with the VIPER Algorithm

Recall from Section I that the **VIPER** (Vertical Itemset Partitioning for Efficient Rule-extraction) algorithm [33] is an example of vertical item-centric frequent pattern mining algorithms, with which data are represented as a collection of bitmaps. Each bitmap for an item captures which transactions contain the specific item. A bit of 1 in the *i*-th position indicates the presence of the item in the *i*-th transaction, whereas a bit of 0 in the *i*-th position indicates the absence of the item from the *i*-th transaction.

A key difference between the horizontal transaction database and the vertical transaction database is that the former refers to the standard representation of transactions, where a set of items is associated with each transaction [31, 32]. The Apriori algorithm uses the horizontal representation. On the other hand, one can represent the transaction database in a "vertical" format [33]. A bitmap for an item can represent a transaction database in a vertical format by putting a "1"-bit in the *i*-th position indicates the presence of the item in the *i*-th transaction and a "0"-bit the *i*-th position indicates the absence of the item from the *i*-th transaction. For example, for transactions $T_1 = \{a, b\}$ and $T_2 = \{b\}$, the corresponding vertical representation of the transaction database is $bitmap(a)$ = [10 ...] and $bitmap(b)$ = [11 ...] . Each bitmap is of the same length, and its length equals to the number of transactions. The VIPER algorithm makes use of the vertical representation.

Like the Apriori algorithm, let C_k and L_k be the sets containing candidate itemsets and frequent itemsets respectively of size k. First, the VIPER algorithm determines which itemsets are in L_1. It then computes the support of any itemset simply by counting (or summing) the number of "1"-bits in its corresponding bitmap. Mathematically, for an itemset X, $\sup(X) = \sum_{\text{bit } i} bitmap(X, i)$, where bitmap(X, *i*) denotes the *i*-th position of bitmap of X. After computing the support for every item occurring in the transaction database, L_1 contains each item with a support greater than or equal to $minsup$.

After determining L_1 , the main loop of the VIPER algorithm is be executed. The first part of the loop involves generating C_k from L_{k-1} . This uses the same candidate generation method in the Apriori algorithm (i.e., performing a self-join on L_{k-1} and pruning the resulting set). Next, it forms bitmap corresponding to each itemset in C_k. Suppose that, for some itemset $X \in C_k$, W is an itemset containing the first $(k-2)$ items in X, y is the second last item in X, and z is the last item in X . Then, $X = W \cup \{y\} \cup \{z\}$. The algorithm computes the bitmap of X as the cross-product of $(W \cup \{y\})$ and $(W \cup \{z\})$, i.e., bitmap(X, *i*) = bitmap(W$\cup${y}, *i*) × bitmap(W$\cup${z}, *i*)}. Next, it computes the support of each pattern in C_k by summing the number of "1"-bits in the

resulting bitmap(X). The frequent patterns in L_k are computed as the candidate patterns in C_k with a support that is at least $minsup$. At the end of a loop iteration, increase k by 1 and continue iterating through the main loop (if necessary). The loop stops iterating when L_{k-1} is empty. In a similar fashion to the Apriori algorithm, all the frequent patterns ($\cup_k L_k$) are returned by the algorithm.

B. Quantitative Association Rule Mining

For quantitative association rule mining [34, 35], suppose that $I = \{i_1, i_2, \cdots, i_m\}$ is the set of all items that can be found in a transaction database for some positive integer m. Then, a transaction can be represented as $T = \{(e_1, f_1), (e_2, f_2), \cdots, (e_t, f_t)\}$ for some positive integer t, where each $e_i \in I$, such that $e_i \neq e_j$ whenever $i \neq j$, and each f_i is a positive integer. The quantitative transaction database is $D = (T_1, T_2, \cdots, T_n)$, which is the set of all transactions. Each transaction has a unique numeric identifier TID.

An ***itemexp*** (short for ***item-expression***) is an ordered triplet of the form $(p, \otimes, q)$, where $p \in I$, $\otimes \in \{=, \geq, \leq\}$, and q is a positive integer. We represent an itemexp as $(p \otimes q)$. Then, an ***itemexpset***—i.e., a set of item-expression— can be defined as a set $X = \{x_1, x_2, \cdots, x_k\}$ for some positive integer k, where each $x_i = (p_i, \otimes_i, q_i)$ is an itemexp such that $p_i \neq p_j$ whenever $i \neq j$.

For a transaction $T = \{(e_1, f_1), (e_2, f_2), \cdots, (e_t, f_t)\}$ and an itemexpset $X = \{x_1, x_2, \cdots, x_k\}$ with $x_i = (p_i, \otimes_i, q_i)$, T ***satisfies*** X if for every $i \in \{1,2, \cdots, k\}$, there exists some $j \in \{1,2, \cdots, t\}$ such that $p_i = e_j$ and the expression $(f_j \otimes_i q_i)$ is true. For example, if $T = \{(a, 2), (b, 3), (c, 1)\}$ and $X = \{(a = 2), (b \geq 1)\}$, then T satisfies X. However, if $T = \{(a, 2), (b, 3), (c, 1)\}$ and $X = \{(a \leq 2), (c \geq 2)\}$, then T does not satisfy X because X requires $c \geq 2$ but c only occurs once in T.

If an itemexpset X contains an itemexp of the form $(p \leq q)$ where $p \in I$ and q is a positive integer, then for a transaction T to satisfy X, the item p must still occur in T at least once, even though $0 < q$. In other words, the number of occurrences of item p in T must be in the interval $[1, q]$. For example, if $T = \{(a, 1)\}$ and $X = \{(b \leq 2)\}$, then T does not satisfy X, even though the number of occurrences of b in T is at most 2. By including this restriction, many itemexpsets are prevented from being considered where an item can occur zero times.

For an itemexpset X, the ***support*** of X—denoted sup(X)— is defined as the number of transactions in D that satisfy X. Now, let *minsup* be some non-negative real number. Then, X is a frequent (large) itemexpset if $\sup(X) \geq minsup$. For two itemexpsets X and Y, the association rule $X \Rightarrow Y$ is ***interesting*** if (a) there are no common items between X and Y, (b) $\sup(X \cup Y) \geq minsup$, and (c) for some confidence value $minconf \in [0, 1]$, $\frac{\sup(X \cup Y)}{\sup(X)} \geq minconf$. For example, it is possible for $\{(a \geq 2)\} \Rightarrow \{(b \leq 3), (c = 1)\}$ to be an interesting association rule if the support and confidence values are satisfied. However, $\{(a = 5)\} \Rightarrow \{(a \geq 3)\}$ cannot be an interesting association rule, since the item a appears on both sides of the rule.

C. Horizontal Quantitative Frequent Pattern Mining with the MQA-M Algorithm

The **MQA-M** (Mining Quantitative Association rules with Multiple comparison operators) [34] is an algorithm for mining quantitative frequent patterns. The MQA-M algorithm is very similar to the Apriori algorithm except that it is generalized to handle quantitative transaction databases.

For any positive integer k, let C_k be the set of candidate itemexpsets containing k itemexps and let L_k be the set of candidate itemexpsets containing k itemexps. Like in the Apriori algorithm, $L_k \subseteq C_k$. The MQA-M algorithm starts by generating C_1. Suppose that $item\_\max[p]$ represents the maximum number of times an item p appears in a transaction. For example, if the quantitative database consists of transactions $T_1=\{(a, 1)\}$ and $T_2=\{(a, 3)\}$, then $item\_max[a] = 3$ because the highest number of times a appears in a transaction is 3. Then, for each item p appearing in the quantitative transaction database, add every itemexpset of the form $\{(p, \otimes, q)\}$ to C_1, where $\otimes \in \{=, \geq, \leq\}$ and $q \in \{1, \cdots, item\_max[p]\}$. The algorithm computes the support of each itemexpset in C_1 by iterating through the transactions and checking each itemexpset in C_1 to see if it should increment the support of that itemexpset. It increments the support of an itemexpset if the transaction satisfies that itemexpset. Let k=1. Then, L_1 becomes the set of all itemexpsets with a support that is at least *minsup*. The algorithm removes some itemexpsets from L_1 using two pruning rules [34].

Similar to the Apriori algorithm, the MQA-M also has a main loop. It first runs the loop with $k = 2$. The loop body begins with generating C_k from L_{k-1}. C_k is initially generated using a self join on L_{k-1}, like in the Apriori algorithm. If 2 itemexpsets in L_{k-1} have the same first $(k - 2)$ itemexps, then it generates an itemexpset in C_k consisting of those $(k - 2)$ itemexps and the last itemexp in the 2 itemexpsets in L_{k-1}. However, it imposes an additional restriction that it does not create an itemexpset in C_k where there are 2 itemexps referring to the same item. For example, if L_1 contains $\{(a = 1)\}$ and $\{(a \geq 2)\}$, it does not form $\{(a = 1), (a \geq 2)\}$ in C_2. After the join step, it prunes itemexpsets from C_k with a subset containing $(k - 1)$ itemexps where that subset is not in L_{k-1}. It gets L_k from C_k using the same procedure that was used to obtain L_1. Using the two aforementioned pruning rules, it removes some itemsets from L_k. At the end of the loop body, it increments k and repeats the previous steps (if necessary). The loop terminates when L_{k-1} is empty. Afterwards, it returns $\cup_k L_k$, which contains all the interesting frequent itemexpsets.

III. Our SQ-VIPER Algorithm for Vertical Quantitative Frequent Pattern Mining

A. Vertical Representation of Quantitative Data

To represent quantitative transaction databases in a vertical format, for each item that occurs in the transaction database, we store it as a set of pairs. Each pair contains a transaction ID associated with that item and the number of occurrences of the item in the transaction. Since we are storing a pair, we can call these sets "pairsets". For example, if we have the transactions $T_1 = \{(a, 2)\}$ and $T_2 = \{(a, 4)\}$, then the transaction database can be represented vertically using $pairset(a) = \{(T_1, 2), (T_2, 4)\}$.

It is useful to convert the quantitative transaction database to this vertical format when implementing the **SQ-VIPER**

(Scalable Quantitative mining algorithm with Vertical Itemset Partitioning for Efficient Rule-extraction) algorithm. We can also define *bitmaps* for quantitative association rule mining. For any itemexpset X, $bitmap(X)$ is defined as the set of transaction IDs corresponding to transactions which satisfy X. Continue with the above example. We can determine that $bitmap(\{(a \geq 2)\}) = [11 \ldots]$ while $bitmap(\{(a = 4)\}) = [01 \ldots]$. When X is an itemexpset containing at least 2 itemexps, we can break down X as $X = W \cup \{y\} \cup \{z\}$, where W is an itemexpset with 2 fewer elements than X and y and z are itemexps. Like bitmap for Boolean frequent pattern mining, we have

- a recursive equation to compute the cross-product to indicate the presence (or absence of transactions that contain the specific item, and
- another recursive equation to compute the minimum to indicate the quantity of the specific item.

We use this recursive definition to generate bitmaps for itemexpsets containing at least two "1"-bits when running the algorithm. The support of an itemexpset X can be computed simply by counting or summing the number of "1"-bits in its bitmap.

B. Key Observations from the Existing Q-VIPER Algorithm

The existing Q-VIPER (and MQA-M) algorithms compute $item\_max[item]$ as the maximum number of times item appears in a transaction, over all transactions in the transaction database. Then, it generates candidate 1- itemexpsets such that each itemexpset consists of a single itemexp of the form $(item, operation, quantity)$, where $item$ is an item in the transaction database, $operation \in \{=, \geq, \leq\}$, and $quantity \in \{1, \cdots, item\_max[item]\}$. For small maximum quantity (say, 4), the algorithm generates $item\_max[item] \times 3 = 12$ candidate 1-itemexpsets (i.e., 12 candidate singleton itemexpset) because it generates 3 candidates (one for each operation =, ≥, and ≤) for each quantity from 1 to $item\_max[item]$. The number of candidates increases when $item\_max[item]$ increases. For example, if we have the transactions $T_1 = \{(a, 502)\}$ and $T_2 = \{(a, 504)\}$, the algorithm would generates $item\_max[item] \times 3 = 504 \times 3 = 1512$ candidate 1-itemexpsets.

In many real-life scenarios, it is not uncommon to deal with large quantities. As such, a large number of candidates will be generated. Unfortunately, many of these generated candidates will be pruned later by the pruning rules, which prune redundant frequent 1-itemexpsets. These rules check groups of itemexpsets having the same (items, operation, support)-combination and select the tightest itemexpset. Continue with our example. Among 1512 candidate 1- itemexpsets, 1506 will be pruned by the pruning rules and only 6 are tightest itemexpsets.

Moreover, redundant candidates are generated not only at the singleton level (k=1) but also higher cardinality levels (k>1). Continue with the above example, the 6 frequent 1- itemexpsets (3 for item a and 3 for item b) lead to 18 candidate 1-itemexpsets. Among them, 9 will be pruned by the pruning rules and the remaining 9 are tightest itemexpsets.

C. Our SQ-VIPER Algorithm

With the above observations, we conducted a careful in-depth analysis. The analytical results inspire our algorithm called SQ-VIPER. Specifically, **our SQ-VIPER algorithm** discovers quantitative frequent patterns vertically as follows. For any integer $k \geq 1$, we define C_k to be the set of candidate itemexpsets with size k and L_k to be the set of frequent itemexpsets with size k. First, we convert the quantitative transaction database into a vertical format if it is not already. Then, it computes all itemexpsets in C_1 by only generating the boundary cases (instead of all cases from quantity 1 to *item_max*[*item*] as in Q-VIPER). By doing so, we do not generate many redundant candidate 1-itemexpsets that will be pruned by the pruning rules. Specifically, the number of candidate 1-itemexpsets generated by our SQ-VIPER for each item i with v_i distinct quantity values within 1 to *item_max*[i]:

$$\#C_1 = \sum_{\text{item i}} (v_i \times 3) \quad (1)$$

where 3 indicates the three operators =, ≥, and ≤.

After computing C_1, we compute the bitmap associated with each itemexpset in C_1. The bitmaps can easily be computed from the vertical representation of the quantitative transaction database. We then compute the support of each itemexpset in C_1 by counting or summing the number of "1"- bits in its corresponding bitmaps. The itemexpsets in L_1 are itemexpsets in C_1 with a support ≥ *minsup*.

Next, we increment $k = 2$ and begin executing the main loop. We generate C_k from L_{k-1} by performing a self join on redundant L_{k-1}. If there are 2 itemexpsets in L_{k-1} where the first $(k-2)$ itemexps in those itemexpsets are the same and the last itemexp in those itemexpsets refer to different items, then we check if these two itemexpsets involve different items and they appear in the transactions. With vertical representation, this means the i-th bit of two frequent (k-1)-itemexpsets are both "1". By doing so, we do not generate many redundant candidate k-itemexpsets that will be pruned by the pruning rules. Specifically, the number of candidate k-itemexpsets generated by our SQ-VIPER for each item i with v_i distinct quantity values within 1 to *item_max*[i]:

$$\#C_k = \prod_{\text{k items}} \sum_i (v_i \times 3) \quad (2)$$

This main loop is repeated for $k > 2$ until L_k is empty. The output of our algorithm is $\bigcup_k L_k$, which contains all interesting frequent itemexpsets.

Example 1. Suppose that L_2 contains $\{(a \geq 502), (b = 63)\}$ and $\{(a \geq 504), (b = 63)\}$ before pruning and that those itemexpsets have the same support. Using the original pruning rules that were used in MQA-M, neither itemexpset would be pruned. However, using the pruning rules in either Q-VIPER or SQ-VIPER, $\{(a \geq 502), (b = 63)\}$ would be pruned from L_2.

TABLE I. CANDIDATE AND FREQUENT ITEMEXPSETS OF SIZE 1 BY OUR IMPROVED ALGORITHM SQ-VIPER

C_1			L_1	
X	bitmap(X)	sup(X)	X	sup(X)
{(a = 502)}	[100]	1	{(a = 502)}	1
{(a = 504)}	[010]	1	{(a = 504)}	1
{(a ≥ 502)}	[110]	2	{(a ≥ 502)}	2
{(a ≥ 504)}	[010]	1	{(a ≥ 504)}	1
{(a ≤ 502)}	[100]	1	{(a ≤ 502)}	1
{(a ≤ 504)}	[110]	2	{(a ≤ 504)}	2
{(b = 10)}	[001]	1	{(b = 10)}	1
{(b = 63)}	[110]	2	{(b = 63)}	2
{(b ≥ 10)}	[111]	3	{(b ≥ 10)}	3
{(b ≥ 63)}	[110]	2	{(b ≥ 63)}	2
{(b ≤ 10)}	[001]	1	{(b ≤ 10)}	1
{(b ≤ 63)}	[111]	3	{(b ≤ 63)}	3

Example 2. Suppose we set $minsup$ = 2 and have 3 transactions in a quantitative transaction database: T1 {a:502, b:63}, T2 {a:504, b:63}, and T3 {b:10}. Note that $item\_max[a]$ = 504 and $item\_max[b]$ = 63 because the highest number of occurrences of a in a transaction is 504 and the highest number of occurrences of b in a transaction is 63. As such, the original Q-VIPER would generate 504×3 = 1512 candidate 1-itemexpsets for item a and 63×3 = 189 candidate 1-itemexpsets for item b, for a total of 1701 candidate 1-itemexpsets. In contrast, by consider only distinct quantity values, our algorithm SQ-VIPER generates much fewer candidates. Specifically, it generates only 2×3 = 6 candidate 1-itemexpsets for item a (due to 2 distinct values for a's quantities 502 & 504) and 2×3 = 6 candidate 1-itemexpsets for item b (due to 2 distinct values for b's quantities 10 & 63), for a total of 12 candidate 1-itemexpsets (i.e., 0.71% of the number of candidate 1-itemexpsets generated by the original Q-VIPER). This confirms the number of candidates generated by SQ-VIPER as per Eq. (1). Regardless whether one use the original Q-VIPER or our algorithm SQ-VIPER, only 12 candidate 1-itemexpsets are essential. These are exactly the same 12 candidate 1- itemexpsets generated by our SQ-VIPER. As the extra 1689 candidate 1-itemexpsets generated by the original Q-VIPER would be pruned by the pruning rules, it would be redundant to generate these 1689 candidate 1-itemexpsets in the first place. See Table I.

Example 3. Continue Example 2 for k = 2 . With 6 frequent 1-itemexpsets for item a and 6 frequent 1-itemexpsets for item b, the original Q-VIPER would generates 6×6 =36 candidate 2-itemexpsets. In contrast, by considering only existing combinations in the transactions, our algorithm SQ-VIPER generates much fewer candidates. Specifically, among 12 frequent 1-itemexpsets, (a:502) and (a:504) only co-occur with (b:63) but not (b:10). As such, SQ-VIPER do not generate and candidate 2-itemexpsets with (b=10), (b≥10) or (b≤10). This reduces the number of candidates to 18. See Table II. Once again, our SQ-VIPER generates only essential itemexpsets (instead of generating many redundant itemexpsets as in the original Q-VIPER).

TABLE II. CANDIDATE AND FREQUENT ITEMEXPSETS OF SIZE 2 BY OUR ALGORITHM SQ-VIPER

C_2			L_2	
X (where θ∈{=, ≥, ≤})	bitmap(X)	sup(X)	Before pruning	After pruning
{(a=502), (b θ 63)}	[100] × [110]^T = [100]	1	✓	✓
{(a=504), (b θ 63)}	[010] × [110]^T = [010]	1	✓	✓
{(a≥502), (b θ 63)}	[110] × [110]^T = [110]	2	✓	✓
{(a≥504), (b θ 63)}	[010] × [110]^T = [010]	1	✓	✓
{(a≤502), (b θ 63)}	[100] × [110]^T = [100]	1	✓	✓
{(a≤504), (b θ 63)}	[110] × [110]^T = [110]	2	✓	✓

IV. EVALUATION

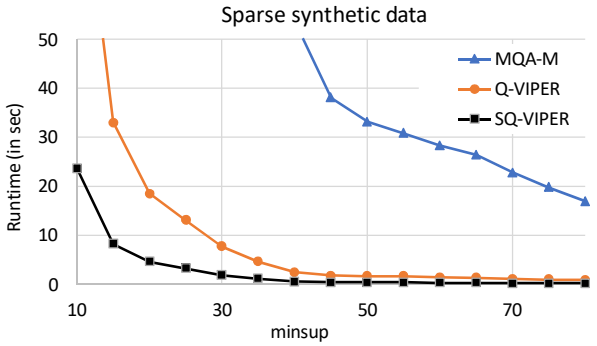
To evaluate our SQ-VIPER algorithm, we compared it with the existing MQA-M algorithm [34] and Q-VIPER algorithm [36]. The performance of the algorithms is assessed using four different quantitative transaction databases:

1. **sparse synthetic dataset**, with $prob$=0.2; and
2. **dense synthetic dataset**, with $prob$=0.8.
3. modified **chess** dataset; and
4. modified **mushroom** dataset.

Here, in the two synthetic datasets, we assume that there are n transactions and $|I|$ different items. Each item has a probability $prob$ of occurring in a particular transaction, where $0 \le prob \le 1$. If the item appears in the transaction, then the number of occurrences of that item follows a $Poisson(\lambda)$ distribution plus 1. We set n = 1000, $|I|$ = 50, and λ = 1. The values of $prob$ for these two quantitative transaction databases are 0.2 and 0.8. These quantitative transaction databases considered as sparse and dense, respectively

In contrast, the original chess and mushroom datasets were two real-life datasets from UCI ML Repository [37]. We modified them by making them quantitative transaction databases. Whenever an item occurs in a transaction, instead of it only occurring once, its number of occurrences follows a $Poisson(\lambda = 1)$ distribution plus 1.

The three algorithms for quantitative frequent itemset mining (i.e., MQA-M [34], Q-VIPER [36] and our SQ- VIPER) have been implemented in the Python language. The algorithms were run on a Windows 10 Nitro AN515-55 laptop using an Intel® Core™ i5-10300H CPU at 2.50 GHz and 8.00 GB RAM. To keep the comparisons between the three algorithms fair, many common procedures (e.g., for generating candidate itemexpsets, determining frequent itemexpsets, pruning frequent itemexpsets) are shared among the algorithms. When we implement the MQA-M algorithm, we use the pruning rules used in Q-VIPER and SQ-VIPER rather than the pruning rules originally used with MQA-M. This allows the simulations to emphasize the differences between the algorithms.



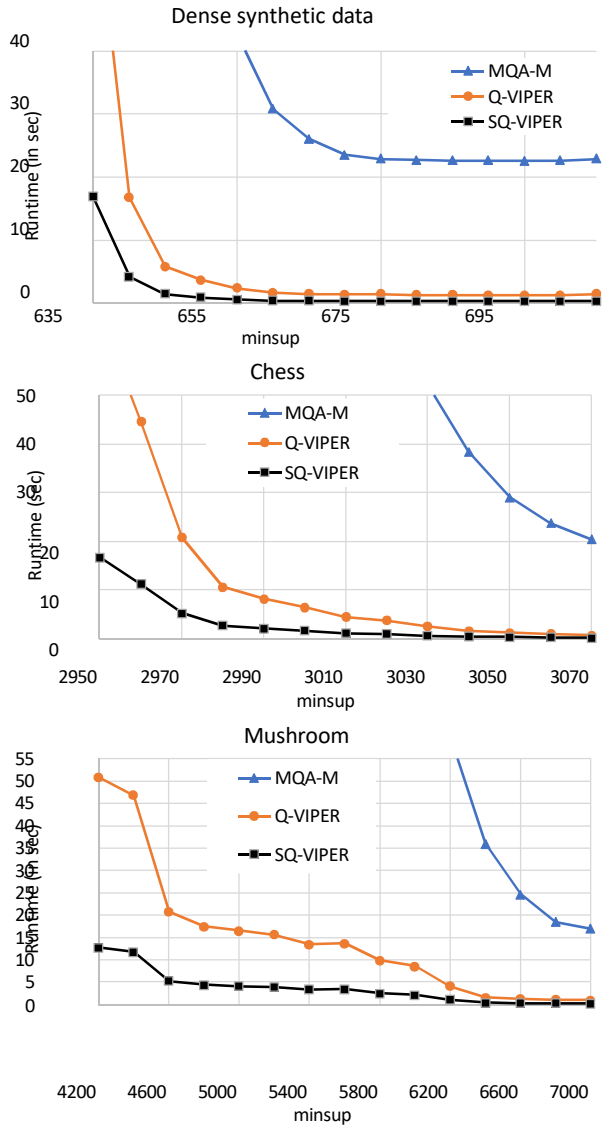


Fig. 1. The runtimes of the existing MQA-M [34] and Q-VIPER [36] algorithm when compared with our SQ-VIPER algorithms for quantitative frequent itemset mining for various *minsup* values.

We run the main code once for every quantitative transaction database. For each of those quantitative transaction databases, we use a sequence of *minsup* values. The sequence depends on the quantitative transaction database being used so that we see interesting results and that the algorithms do not take too long to run. For each combination of a quantitative transaction database and a value for *minsup*, the MQA-M, Q-VIPER and SQ-VIPER algorithms are be run and timed.

Fig. 1 show the runtimes of each of the three algorithms for a variety of values of *minsup* for each of the four quantitative transaction databases. The runtime (in seconds) is shown on the *y*-axis while the value of *minsup* is given on the *x*-axis. In all cases, our SQ-VIPER outperforms both existing Q-VIPER and MQA-M algorithms.

V. Conclusions

In this paper, we presented our scalable algorithm for vertical quantitative frequent pattern mining called SQ- VIPER. This algorithm first represents the big data as a collection of bitmaps. Each item-centric bitmap captures the presence or absence of a transaction containing the item, as well as the quantity of that item in each transaction. With this representation, our algorithm then vertically mines quantitative frequent patterns. During the mining process, we only generate candidates at the boundaries rather than all quantity values. By doing so, it significantly reduce the candidate space and this reducing mining space and the runtime. Hence, it speeds up the social network mining and analysis process in revealing interesting social relationships (e.g., degree of interesting in social entities' postings). When compared the existing MQA-M and Q-VIPER algorithms (which were both built for quantitative frequent pattern mining), evaluation results show that our scalable quantitative vertical bitwise SQ-VIPER algorithm requires shorter runtime to mine frequent patterns. As *ongoing and future work*, we explore ways to further enhance the mining of quantitative frequent patterns and to extend this work to mine other quantitative patterns, especially for patterns in social networks.

Acknowledgment

This project is partially supported by NSERC (Canada) and University of Manitoba.

References

[1] P. Bellini, et al., "Data flow management and visual analytic for big data smart city/IoT," IEEE SmartWorld-ScalCom-UIC-ATC- CBDCom-IoP-SCI 2019, pp. 1529-1536.

[2] C.K. Leung, et al., "Scalable mining of big data," IEEE SmartWorld 2021, pp. 240-247.

[3] J. Zhao, et al., "Code refactoring from OpenMP to MapReduce model for big data processing," IEEE SmartWorld-ScalCom-UIC-ATC- CBDCom-IoP-SCI 2019, pp. 930-935.

[4] C. Zhu, et al., "5G wireless networks meet big data challenges, trends, and applications," IEEE SmartWorld-ScalCom-UIC-ATC-CBDCom- IoP-SCI 2019, pp. 1513-1516.

[5] K.E. Dierckens, et al., "A data science and engineering solution for fast k-means clustering of big data," IEEE TrustCom-BigDataSE-ICESS 2017, pp. 925-932.

[6] M.T. Alam, et al., "Mining frequent patterns from hypergraph databases," PAKDD 2021, Part II, pp. 3-15.

[7] C.K. Leung, et al., "Big data visualization and visual analytics of COVID-19 data," IV 2020, pp. 415-420.

[8] R. Froese, et al., "The border k-means clustering algorithm for one dimensional data," IEEE BigComp 2022, pp. 35-42.

[9] E. Madill, et al., "ScaleSFL: a sharding solution for blockchain-based federated learning," ACM BSCI 2022, pp. 95-106.

[10] C.K. Leung, et al., "Machine learning and OLAP on big COVID-19 data," IEEE BigData 2020, pp. 5118-5127.

[11] C.K. Leung, et al., "Personalized DeepInf: enhanced social influence prediction with deep learning and transfer learning," IEEE BigData 2019, pp. 2871-2880.

[12] P.P.F. Balbin, et al., "Predictive analytics on open big data for supporting smart transportation services," Procedia Computer Science 176, 3009-3018, 2020.

[13] C.C.J. Hryhoruk, et al., "Smart city transportation data analytics with conceptual models and knowledge graphs," IEEE SmartWorld 2021, pp. 455-462

[14] S. Kaur, et al., "A big data science solution for transportation analytics with meteorological data," IEEE BigDataSE 2022,

pp. 21-28.

[15] M. Kolisnyk, et al., "Analysis of multi-dimensional road accident data for disaster management in smart cities," IEEE IRI 2022, pp. 43-48.

[16] C.K. Leung, et al., "Urban analytics of big transportation data for supporting smart cities," DaWaK 2019, pp. 24-33.

[17] D. Deng, et al., "Spatial-temporal data science of COVID-19 data," IEEE BigDataSE 2021, pp. 7-14.

[18] J. de Guia, et al., "DeepGx: deep learning using gene expression for cancer classification," IEEE/ACM ASONAM 2019, pp. 913-920.

[19] D.L.X. Fung, et al., "Self-supervised deep learning model for COVID-19 lung CT image segmentation highlighting putative causal relationship among age, underlying disease and COVID-19," BMC Journal of Translational Medicine 19, 318:1-318:18, 2021.

[20] C.K. Leung, et al., "Smart data analytics on COVID-19 data," IEEE iThings-GreenCom-CPSCom-SmartData-Cybermatics 2021, pp. 372-379.

[21] C.K. Leung, C. Zhao, "Big data intelligence solution for health analytics of COVID-19 data with spatial hierarchy," IEEE DataCom 2021, pp. 13-20.

[22] A.M. Olawoyin, et al., "Privacy preservation of COVID-19 contact tracing data," IUCC-CIT-DSCI-SmartCNS 2021, pp. 288-295.

[23] C. Zhao, et al., "Analyzing COVID-19 epidemiological data," IEEE DASC-PICom-CBDCom-CyberSciTech 2021, pp. 985-990.

[24] D. Choudhery, C.K. Leung, "Social media mining: prediction of box office revenue," IDEAS 2017, pp. 20-29.

[25] J. Kang, et al., "Characterizing collective knowledge sharing behaviors in social network," IEEE SmartWorld-ScalCom-UIC-ATC-CBDCom- IoP-SCI 2019, pp. 869-876.

[26] C.K. Leung, et al., "Big data analytics of social network data: Who cares most about you on Facebook?" Highlighting the Importance of Big Data Management and Analysis for Various Applications, pp. 1- 15, 2018.

[27] C. Couronne, et al., "PrePeP: a light-weight, extensible tool for predicting frequent hitters," ECML-PKDD 2020, Part V, pp. 570-573.

[28] J. Fischer, J. Vreeken, "Sets of robust rules, and how to find them," ECML-PKDD 2019, Part I, pp. 38-54.

[29] C.K. Leung, S.K. Tanbeer, "Mining popular patterns from transactional databases," DaWaK 2012, pp. 291-302.

[30] F. Seiffarth, et al., "Maximal closed set and half-space separations in finite closure systems," ECML-PKDD 2019, Part I, pp. 21-37.

[31] R. Agrawal, et al., "Mining association rules between sets of items in large databases," ACM SIGMOD 1993, pp. 207-216.

[32] R. Agrawal, R. Srikant, "Fast algorithms for mining association rules," VLDB 1994, pp. 487-499.

[33] P. Shenoy, et al., "Turbo-charging vertical mining of large databases," ACM SIGMOD 2000, pp. 22-33.

[34] P.Y. Hsu, et al., "Algorithms for mining association rules in bag databases," Information Sciences 166(1-4), 31-47, 2004.

[35] R. Srikant, R. Agrawal, "Mining quantitative association rules in large relational tables," ACM SIGMOD 1996, pp. 1-12.

[36] T.J. Czubryt, et al., "Q-VIPER: quantitative vertical bitwise algorithm to mine frequent patterns," DaWaK 2022, pp. 219-233.

[37] M. Kelly, et al., UCI Machine Learning Repository. http://archive.ics. uci.edu/ml

[38] P. Bellini, et al., "Data flow management and visual analytic for big data smart city/IoT," IEEE SmartWorld-ScalCom-UIC-ATC- CBDCom-IoP-SCI 2019, pp. 1529-1536.

[39] C.K. Leung, et al., "Scalable mining of big data," IEEE SmartWorld 2021, pp. 240-247.

[40] J. Zhao, et al., "Code refactoring from OpenMP to MapReduce model for big data processing," IEEE SmartWorld-ScalCom-UIC-ATC- CBDCom-IoP-SCI 2019, pp. 930-935.

[41] Zhu, et al., "5G wireless networks meet big data challenges, trends, and applications," IEEE SmartWorld-ScalCom-UIC-ATC-CBDCom- IoP-SCI 2019, pp. 1513-1516.

[42] K.E. Dierckens, et al., "A data science and engineering solution for fast k-means clustering of big data," IEEE TrustCom-BigDataSE-ICESS 2017, pp. 925-932

relationship among age, underlying disease and COVID-19," BMC Journal of Translational Medicine 19, 318:1-318:18, 2021.

[43] C.K. Leung, et al., "Smart data analytics on COVID-19 data," IEEE iThings-GreenCom-CPSCom-SmartData-Cybermatics 2021, pp. 372-379.

[44] C.K. Leung, C. Zhao, "Big data intelligence solution for health analytics of COVID-19 data with spatial hierarchy," IEEE DataCom 2021, pp. 13-20.

[45] A.M. Olawoyin, et al., "Privacy preservation of COVID-19 contact tracing data," IUCC-CIT-DSCI-SmartCNS 2021, pp. 288-295.

[46] C. Zhao, et al., "Analyzing COVID-19 epidemiological data," IEEE DASC-PICom-CBDCom-CyberSciTech 2021, pp. 985-990.

[47] D. Choudhery, C.K. Leung, "Social media mining: prediction of box office revenue," IDEAS 2017, pp. 20-29.

[48] J. Kang, et al., "Characterizing collective knowledge sharing behaviors in social network," IEEE SmartWorld-ScalCom-UIC-ATC-CBDCom- IoP-SCI 2019, pp. 869-876.

[49] C.K. Leung, et al., "Big data analytics of social network data: Who cares most about you on Facebook?" Highlighting the Importance of Big Data Management and Analysis for Various Applications, pp. 1- 15, 2018.

[50] C. Couronne, et al., "PrePeP: a light-weight, extensible tool for predicting frequent hitters," ECML-PKDD 2020, Part V, pp. 570-573.

[51] J. Fischer, J. Vreeken, "Sets of robust rules, and how to find them," ECML-PKDD 2019, Part I, pp. 38-54.

[52] C.K. Leung, S.K. Tanbeer, "Mining popular patterns from transactional databases," DaWaK 2012, pp. 291-302.

[53] F. Seiffarth, et al., "Maximal closed set and half-space separations in finite closure systems," ECML-PKDD 2019, Part I, pp. 21-37.

[54] R. Agrawal, et al., "Mining association rules between sets of items in large databases," ACM SIGMOD 1993, pp. 207-216.

[55] R. Agrawal, R. Srikant, "Fast algorithms for mining association rules," VLDB 1994, pp. 487-499.

[56] P. Shenoy, et al., "Turbo-charging vertical mining of large databases," ACM SIGMOD 2000, pp. 22-33.

[57] P.Y. Hsu, et al., "Algorithms for mining association rules in bag databases," Information Sciences 166(1-4), 31-47, 2004.

[58] R. Srikant, R. Agrawal, "Mining quantitative association rules in large relational tables," ACM SIGMOD 1996, pp. 1-12.

[59] T.J. Czubryt, et al., "Q-VIPER: quantitative vertical bitwise algorithm to mine frequent patterns," DaWaK 2022, pp. 219-233.

[60] M. Kelly, et al., UCI Machine Learning Repository. http://archive.ics. uci.edu/ml

[61] P. Bellini, et al., "Data flow management and visual analytic for big data smart city/IoT," IEEE SmartWorld-ScalCom-UIC-ATC- CBDCom-IoP-SCI 2019, pp. 1529-1536.

[62] C.K. Leung, et al., "Scalable mining of big data," IEEE SmartWorld 2021, pp. 240-247.

[63] J. Zhao, et al., "Code refactoring from OpenMP to MapReduce model for big data processing," IEEE SmartWorld-ScalCom-UIC-ATC- CBDCom-IoP-SCI 2019, pp. 930-935.

[64] Zhu, et al., "5G wireless networks meet big data challenges, trends, and applications," IEEE SmartWorld-ScalCom-UIC-ATC-CBDCom- IoP-SCI 2019, pp. 1513-1516.

[65] K.E. Dierckens, et al., "A data science and engineering solution for fast k-means clustering of big data," IEEE TrustCom-BigDataSE-ICESS 2017, pp. 925-932.