A Hybrid Partitioning Strategy for Distributed FCA

Muneeswaran P[0000−0003−4099−3003], Jyoti[0000−0003−1595−3173], and Sriram Kailasam[0000−0002−2218−8660]

Indian Institute of Technology Mandi, India
cegmunees@gmail.com, jjangra2@gmail.com, India
sriramk@iitmandi.ac.in

Abstract. Several parallel and distributed FCA algorithms have been proposed to speedup the discovery of concepts. Some require the context to be replicated on all machines while others partition the context horizontally across different machines. The former suffers from memory bottlenecks for large contexts as every machine retains the entire context in memory. The latter approaches are based on Map-Reduce paradigm wherein to compute a valid concept requires all-all communication between different machines, incurring huge communication overheads. It has been shown that DFS (Depth-First-Search) based approaches can exploit database reduction strategy for concept sub-tree exploration. This significantly decreases the cost of closure computation in the concept sub-tree and results in higher speedup. The existing Map-Reduce based approaches cannot use DFS exploration strategy. In this paper, we propose a hybrid partitioning strategy for distributed FCA that uses horizontal partitioning for groups of low support attributes and vertical partitioning for high support attributes. This not only alleviates memory bottlenecks but also minimizes communication overheads. For vertically partitioned sub-context, inter-machine communication is only required for concepts below a certain support threshold. We further minimize the communication cost by using group-id based auxiliary structure. This hybrid partitioning also allows DFS based concept exploration, thereby making it more scalable and efficient.

Keywords: Hybrid partitioning · Large context · Distributed FCA

1 Introduction

Formal Concept Analysis (FCA) is a method of data analysis that aims at extracting natural clusters (formal concepts) from object and attribute data table (a.k.a. context). The sequential algorithms for computing concepts (e.g. Close-By-One (CbO) [8], LCM [1], and DCI-Closed [9]) do not scale to large contexts as the total number of concepts can be extremely large. In the worst-case, it is exponential in the number of objects/attributes. To speedup the discovery of concepts, several parallel and distributed FCA algorithms have been proposed [5, 3, 2, 11, 4]. These algorithms either replicate the input context on all machines

 Published in Francisco J. Valverde-Albacete, Martin Trnecka (Eds.): Proceedings of the 15th International Conference on Concept Lattices and Their Applications, CLA 2020, pp. 71–82, 2020.

or partition the context horizontally across different machines. The algorithms that use replicated context [3] suffer from memory bottlenecks for large contexts as every machine needs to retain the entire context in memory. The other algorithms [2, 11] partition the context horizontally across multiple machines. Here, computing a concept that is valid in the entire context involves two steps: (1) each machine first computes the local closure in its partition. Then it exchanges the local closure with other machines using broadcast. (2) At the end of broadcast, each machine computes the valid concept by intersecting the local closures from all the machines. The above two steps are repeated until all valid concepts are discovered. MR-Ganter [2] implements the above procedure using a sequence of Map-Reduce (MR) jobs whereas MR-Ganter+ [11] uses Twister [10], a MR-based framework for iterative computations.

It has been shown that DFS based approaches can exploit database reduction for concept sub-tree exploration. This significantly decreases the cost of closure computation in the concept sub-tree and results in higher speedup. The existing MR-based approaches cannot use DFS exploration strategy as it would make them highly inefficient. If they compute one concept at a time using DFS, they would incur large waiting time due to all-all communication between the machines. Hence, they explore the concept tree in a breadth-first manner and try to overlap communication and computation. However, they lose the benefits of database reduction available in DFS based approach. From experiments, we find that the distributed MR-based algorithms perform slower than even single node DFS based algorithms like LCM [1].

In this paper, we propose a hybrid partitioning strategy for distributed FCA that uses horizontal partitioning for groups of low support attributes and vertical partitioning for high support attributes (see Fig. 1). This partitioning strategy allows DFS based concept exploration, alleviates memory bottlenecks and minimizes communication overheads. Hence, it is more scalable and efficient than existing algorithms. We also propose some optimizations for reducing inter-machine communication cost by using group-id based auxiliary structure.

Rest of the paper is organized as follows: Section 2 introduces the relevant background and related work; Section 3 discusses the hybrid partitioning strategy and the optimizations for reducing inter-machine communication cost; Section 4 describes the performance results and Section 5 discusses the conclusion and future work of this paper.

2 Background and Related Work

2.1 FCA: Notations and Definitions

In this section, we introduce the relevant notations and terminology of FCA.

Definition 1 *A formal context* $\mathbb{K} = (G, M, I)$*, consists of two sets* G *and* M*, and a binary relation* $I \subseteq G \times M$*.* G *represents the set of objects and* M *represents the set of attributes. If* $g \in G$ *and* $m \in M$ *are in relation* I*, we write* $(g, m) \in I$ *or* gIm*. It can be read as "object* g *has attribute* m*".*

Formal concept is computed using two operators $^{\uparrow}$ and $^{\downarrow}$. $^{\downarrow\uparrow}$ is a closure operator [6]. These operators are defined as below:

Definition 2 *Let $A \subseteq G$ and $B \subseteq M$, then: $A^{\uparrow} = \{m \in M \mid \forall g \in A : (gIm)\}$, and $B^{\downarrow} = \{g \in G \mid \forall m \in B : (gIm)\}$*

Definition 3 *An object subset $A \subseteq G$ is closed in (G, M, I) if $A = A^{\uparrow\downarrow}$. An attribute subset $B \subseteq M$ is closed in (G, M, I) if $B = B^{\downarrow\uparrow}$.*

A formal concept is described by an extent and an intent as shown below:

Definition 4 *A duplet $\langle A, B\rangle$ is called formal concept of context $\mathbb{K} = (G, M, I)$, if $A \subseteq G$ and $B \subseteq M$, $A^{\uparrow} = B$, and $A = B^{\downarrow}$. The set A (object set) is called the extent and the set B (attribute set) is called the intent of the concept.*

2.2 Basic Linear time Closed itemset Miner (LCM)

Linear time Closed itemset Miner (LCM) [1] is state-of-the-art sequential algorithm for concept discovery. LCM algorithm induces a search tree on concepts and explores it in a depth-first-search manner. Given a concept intent, the LCM procedure adds attributes which are not part of the intent (a.k.a. list of markers) and computes the resulting closure. These closures form the next-level concepts. To prevent generating the same concept multiple times (duplicate concepts), LCM orders the attributes; it considers a particular concept valid only if it is generated in the sub-tree of the leftmost attribute in that concept. This redundancy check is performed during closure computation, by testing whether the newly added attributes in the closure contain any attribute to the left of the marker. If yes, it is considered a duplicate, else it is considered a valid concept. LCM bounds the time complexity of concept mining to a function linear in the number of concepts and its memory usage is linear in the size of the context. However, due to its sequential nature, it does not scale to large contexts. We have designed a Group based LCM algorithm that can scale to large contexts.

2.3 Related Work

There are both parallel and distributed implementations of the LCM algorithm. P-LCM [5] is a parallel version of LCM based on shared memory model; the task queue is shared among all the workers. When a worker becomes idle, it pulls tasks from the shared queue. This approach works in a multi-core environment, but requires redesign to work in distributed shared nothing architectures largely prevalent in the cloud [12]. Distributed-LCM [6] uses a master-worker based architecture. The master performs static load balancing to assign seed concepts to each worker. It also maintains meta-data about the load on each worker. The workers generate the concepts in the sub-tree below the assigned seed concept. When they become idle, they request the master to assign a donor and fetch work from the donor. This master-aided dynamic load balancing scheme distributes the skewed concept tree of LCM evenly among the workers. Distributed-LCM

provides linear speedups for many contexts, however, it is not scalable to large contexts as it uses context replication on all the workers.

MR-CbO [3] is the first distributed implementation of CbO [2] based on the Map-Reduce framework. It follows an iterative approach. In the map phase, each mapper(worker) computes the first level concepts from a given seed concept and passes them to the reduce phase for redundancy check. Those concepts which pass the redundancy check are considered as seed concepts for the next iteration. MR-CbO also requires the entire context to be replicated. Hence it is not scalable. To address this issue, MR-Ganter [2] partitions the context horizontally. Each mapper finds the local closure with respect to their partitioned context. In the next phase, global closure is computed from the local closures of the previous phase. This method suffers from communication overhead for computing the concepts valid in the entire context and for performing the redundancy check. Further, the MR-based approaches do not use DFS based exploration of the concept tree, thereby losing the benefits of database reduction.

Yoshizoe et al. [12] proposed an attribute-based partitioning of the context; wherein each partition consists of objects in the extent of the attribute. This allows discovery of all concepts that contain a particular attribute without any inter-machine communication. However, for large contexts, the partition corresponding to high support attributes would contain almost the entire context leading to memory bottleneck.

In summary, existing parallel and distributed versions of LCM either work on replicated context or horizontally partitioned context. The former suffers from memory bottlenecks for large contexts as every machine retains the entire context in memory. The latter approaches incur huge communication overheads and are not able to explore the concept tree using DFS based strategy.

3 `G-LCM` (group-based LCM)

We propose `G-LCM`, a group-based LCM algorithm that scales to large contexts. `G-LCM` uses a hybrid partitioning strategy to bound the communication cost as well as the memory consumption across machines.

3.1 Hybrid Partitioning in `G-LCM`

Hybrid partitioning is a technique that uses horizontal partitioning for low support attributes and vertical partitioning for high support attributes (above certain support threshold). The context is reordered according to the support of the attributes. For creating horizontal partitions, we start with the attribute having the lowest support and form groups of attributes in an increasing order of their support by fixing an upper bound on the total number of objects in a group.

Thus, the low support attributes are grouped into many horizontal partitions, each of which can be stored in a single machine. The high support attributes are grouped into a single vertical partition containing only the frequent attributes

in each of its objects. This enables us to compress the vertical partition since many objects in this group would become identical and can be coalesced.

We use 3 Hadoop Map-Reduce jobs to partition the input context. Job1 computes the support of attributes; job2 reorders the attributes based on their support and job3 partitions the context based on the grouping strategy.

3.1.1 Horizontal Partitioning: The sub-context induced by a group of attributes is referred as a horizontal partition. Consider a formal context $\mathbb{K} = (G, M, I)$ relabeled according to support of the attributes; the attribute with the lowest support is assigned the lowest index.

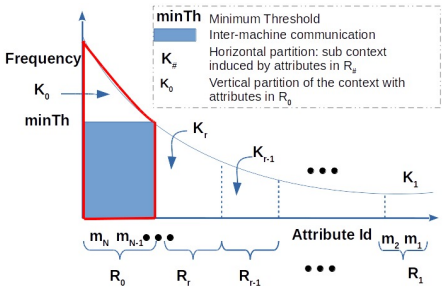

Fig. 1: Partitions of the context with N attributes. The number of horizontal partitions is r, i.e. $\{K_1, K_2, \ldots, K_r\}$. K_0 denotes the vertical partition.

Definition 5 *Suppose attributes are indexed and arranged in the ascending order of their support as m_1, m_2, ... $m_{|M|}$. The grouping strategy partitions the attributes linearly into (say $r+1$-groups): R_1, R_2, ...,R_r, and R_0.*

1. *Each horizontal partition, $R_i = \{m_k \mid m_k \in M,\ k \in [L(i), U(i)]\}$*
2. *Vertical partition $(R_0) = \{m_k \mid m_k \in M,\ support(m_k) \geq minTh\}$*

$$L(i) = \begin{cases} 1, & \textbf{\textit{if}}\ i = 1 \\ U(i-1)+1, & \textbf{\textit{if}}\ 1 < i \leq r \end{cases}$$

$$U(i) = \left\{ j \mid j < |M| \ and \sum_{k=L(i)}^{j} support(m_k) \leq \theta < \sum_{k=L(i)}^{j+1} support(m_k) \right\}$$

Where, θ is upper bound on the number of objects in a partition and $support(m_k)$ returns the cardinality of $m_k^{\downarrow}$.

Suppose that $\mathbb{K}$ is partitioned into k horizontal partitions each having group-id in $[1, r]$(see Fig. 1). While forming groups of attributes, we ensure that the total number of objects in their corresponding partitions is less than or equal to a threshold θ. Consider a toy context shown in Table 1 consisting of four

Table 1: Toy context

	DEL	NYC	LON	BOM
AirCanada (o1)	X			X
AirIndia (o2)	X	X	X	
Indigo (o3)		X		
Aeroflot (o4)	X	X	X	X

Table 2: Relabelled context

	BOM	LON	NYC	DEL
o1	X			X
o2		X	X	X
o3			X	
o4	X	X	X	X

Table 3: Group 1 sub-context

	BOM	LON	NYC	DEL
o1	X			X
o4	X	X	X	X

Table 4: Group 2 sub-context

	BOM	LON	NYC	DEL
o2		X	X	X
o4	X	X	X	X

attributes and four objects. The relabelled context after sorting the attributes according to support is shown in Table 2. Assume that the upper bound $\theta = 2$, then NYC and DEL are the frequent attributes. The two horizontal partitions are formed with group attribute set $\{BOM\}$ and $\{LON\}$ respectively. Group 1 contains objects 1 and 4 (See Table 3). Group 2 contains objects 2 and 4 (see Table 4).

Definition 6 *The sub-context induced by R_k, $k > 0$, is known as the horizontal partition, $\mathbb{K}_k = (G_k, M_k, I_k)$ and is defined as: $G_k = \bigcup_{m \in R_k}(m^{\downarrow})$, where $m^{\downarrow}$ is the set of objects in $\mathbb{K}$ containing attribute m, $M_k = \bigcup_{g \in G_k}(g^{\uparrow})$, where $g^{\uparrow}$ is the set of attributes in $\mathbb{K}$ containing object g, and $I_k = (G_k \times M_k) \cap I$. $^{\uparrow_k}$ and $^{\downarrow_k}$ denote the corresponding concept forming operators in $\mathbb{K}_k$.*

Lemma 1. *Every concept (A, B) generated by the horizontal partition $\mathbb{K}_k$ that includes at least one attribute $m \in R_k$ as part of its intent (i.e. $m \in B$) is valid in the entire context (G, M, I).*

Proof. Suppose (A, B) is not closed in $\mathbb{K}$. Then by extensivity of closure operator, $B \subseteq B^{\downarrow\uparrow}$. Let $n \in B^{\downarrow\uparrow} \setminus B$.
$\implies \forall g \in B^{\downarrow} : gIn$
Consider an attribute $m \in R_k$ s.t. $m \in B$
$\implies \forall g \in B^{\downarrow} : gIm$
$\implies B^{\downarrow} \subseteq m^{\downarrow}$
$\implies B^{\downarrow} \subseteq G_k \qquad (m \in R_k)$
$\implies B^{\downarrow} = B^{\downarrow_k} \qquad$ (By definition of I_k)
But $B^{\downarrow_k} = A \qquad$ (((A,B) is closed in $\mathbb{K}_k$)
$\implies \forall g \in A : gIn$
$\implies n \in B \qquad$ (hence, proved by contradiction)

3.1.2 Vertical Partitioning: Vertically partitioned context is defined w.r.t. a support threshold and is assigned group-id 0. It contains the union of occurrences

Table 5: Vertical partition

	NYC	**DEL**
AirCanada (o1)		X
AirIndia (o2)	X	X
Indigo (o3)	X	
Aeroflot (o4)	X	X

Table 6: Augmented context

	$m^{(1)}$	$m^{(2)}$
o1	X	
o2		X
o3		
o4	X	X

of all attributes whose support is greater than the threshold. Vertical partition with support threshold 3 is shown in Table 5.

Definition 7 *The vertically partitioned context* $\mathbb{K}_0 = (G_0, M_0, I_0)$ *w.r.t. support threshold* $(minTh)$ *is defined as:* $M_0 \subseteq M$ *s.t.* $\forall m \in M_0, support(m) \geq minTh$, $G_0 = G$ *and* $I_0 = (G_0 \times M_0) \cap I$.

3.2 Concept discovery from vertical partition of the context

As the vertical partition does not include all the attributes, it can lead to discovery of invalid concepts in certain cases. For example closure of {NYC, DEL} in vertical partitioned context is {NYC, DEL} but the actual closure in the input context is {LON, NYC, DEL}. From this example, it is clear that the vertical partition may generate invalid concepts. Hence, we augment the vertically partitioned context with auxiliary information that will help check validity of the concept. We define the augmented context for the vertical partition.

Definition 8 *An augmented context* $\mathbb{K}_{ac} = (G_{ac}, M_{ac}, I_{ac})$ *is defined as:* $G_{ac} = G$, $M_{ac} = \{m^{(k)} \mid 1 \leq k \leq r\}$, *where* r *is the number of horizontal partitions and* $gI_{ac}m^{(k)}$ *iff* $\exists m \in R_k$ *s.t.* gIm.

Table 6 shows the augmented context for the toy example.

Lemma 2. *Let* $^{\uparrow_a}$ *denote the* $^{\uparrow}$ *operator in* $\mathbb{K}_{ac}$. *A concept* (A_v, B_v) *generated by the vertical partition is valid if for every horizontal partition* $\mathbb{K}_k : m^{(k)} \in {A_v}^{\uparrow_a}$ *there exists no attribute* $m \in R_k$ *s.t.* $A_v \subseteq m^{\downarrow_k}$.

Proof. Assume there exists an attribute $m \in R_k$ s.t. $A_v \subseteq m^{\downarrow_k}$.
$\implies \forall g \in A_v : gIm$
$\implies m \in A_v^{\downarrow}$. But $A_v^{\downarrow} = B_v$ and $m \notin B_v$. Hence proved.

For each concept generated by the vertical partition, we intersect the group-ids of the objects in the extent (using the augmented context). If the intersection returns an empty set, then the concept is valid, else an additional check is required to test the validity of the concept. For example, suppose that we make 2 groups in our toy context; attributes in group 0 (vertical partition) contain NYC and DEL, while attributes in group 1 (horizontal partition) contain BOM and LON. Group 0 includes all 4 objects. In the augmented context, each object will contain group-id 1 except object 3. Hence, for concept intent {DEL} in the

augmented context, the group-id intersection returns {1}. This however does not mean that {DEL} is an invalid concept intent. We need to test its validity w.r.t. attributes in group 1. In this case, we find that no attribute in group 1 includes the extent of {DEL}, and therefore {DEL} is a valid concept intent.

Thus, group-id based intersection may generate false positives but no false negatives. To understand how false positives are handled, let us assume a scenario in which, machine M_0 is discovering concept in vertical partition; where as machine M_1 and M_2 are discovering concepts in group 1 and 2 (horizontal partitions) respectively. In machine M_1 and M_2, a hash table is created with keys as the low support attributes (m) of the corresponding group and value being the corresponding attribute's object ids ($m^{\downarrow}$). To check the validity of the concept (A_v, B_v) discovered in the vertical partition, a validity check is issued to all the machines whose id is returned by the group-id based intersection. Upon receiving this query, each machine checks in its hash table whether the extent A_v is a subset of the value stored against any attribute id of that group. If a match is found, then the concept discovered by the vertical partition is invalid. This check can be implemented efficiently using a Cuckoo filter.

3.2.1 Cuckoo filter Cuckoo filter is a probabilistic data structure that checks the membership of an element in a set. It is mainly designed for applications that need to store many elements in the set and check the membership of the set with low false positive rate. Cuckoo filter is variant of cuckoo hash table that stores only *fingerprints* instead of (key, value) pairs which leads to space efficient filter and restricts false positive rate to less than 3% [7].

For each attribute ($m \in R_k$), we store its object IDs ($m^{\downarrow}$) in a cuckoo filter. To test the validity of a concept intent, we first determine the candidate horizontal partitions using the augmented context. Then for each candidate partition, we check the concept extent against the cuckoo filter of those attributes whose support is greater than the size of the extent. If the cuckoo filter returns negative for any object in the concept extent, it means that the concept intent does not include that attribute. Otherwise, the attribute can be present with high probability and needs to be verified by actual comparison of the concept extent with the concerned attribute's extent.

3.2.2 `G-LCM` algorithm for discovering the concepts in vertical partition Algorithm 1 shows how concepts are discovered in the vertical partition. In line number 2, algorithm iterates over the (M_0) attribute set. In line number 4, it checks whether the newly found concept is redundant or not using function *isNewConcept* similar to LCM (see Section 2.2). If the support of the new concept is below the threshold (line 5), then it performs the validity check locally using group-id based intersection function (line 6). If that returns a non-empty set (line 7) then for each group-id in *gIDList* (group-id list), a membership query is sent to the machine which holds that group-id context (lemma 2). On that machine, the cuckoo filter is used to check

the validity of the concept (line 8). As cuckoo filter may produce false positives, we do an actual comparison with the extent if the cuckoo filter returns true (lines 11 - 18). If the extent is having some common attributes in the group-id context then it is a true positive (line 14) else false positive (line 16).

Algorithm 1: `G-LCM` algorithm

Input: $\mathbb{K}_0$: Vertically partitioned context, minTh: Minimum threshold, $\mathbb{K}_{ac}$: Augmented Context.

```
1  // initialize flag=false, X = φ
2  for i := First_Attribute (M_0) to Last_Attribute (M_0) do
3      X = ((X ∪ i) ↓↑)
4      if isNewConcept(X) then
5          if support(X)< minTh then
6              gIDList = groupIDIntersection(Extent(X),K_ac)
7              if size(gIDList) > 0 then
8                  flag = cuckooFilterMembership(Extent(X),gIDList)
9              else
10                 Mark X as a valid concept // true negative
11             if flag==true then
12                 fpFlag= intersectInFreqAtt(Extent(X), gIDList)
13                 if fpFlag==true then
14                     // true positive
15                 else
16                     // false positive and SET flag=false
17             else
18                 // false positive
19         if !flag then
20             G-LCM(X, i)
```

In case the support of the newly found concept is greater than minimum threshold ($minTh$) then the extent of the concept does not have any common attribute from infrequent attributes. Thus, the concept found is valid (line 19). In the next section, we discuss experimental evaluation of this algorithm.

4 Results and Discussion

We implemented `G-LCM` algorithm in Java. The context is partitioned into horizontal and vertical partitions using Map-Reduce jobs. For horizontal partitions, we can use the distributed LCM algorithm from [6] to compute the concepts. Each horizontal partition is passed as a separate job to the framework. As each machine holds only the horizontal partition instead of the entire input context (see section 2.3), the memory bottleneck is resolved. According to lemma 1, each horizontal partition should generate only concepts containing at least one attribute of the group in its intent. To ensure this, we add the attributes of the corresponding group as first level markers (see section 2.2) to the null concept. From second level onward, the basic LCM procedure is followed.

Here, we report performance results for computing concepts in the vertical partition for susy and pumsb_star data-sets. The susy data-set [1] contains 190 attributes and 5 million objects with average object length as 19.

Table 7: Experimental evaluation for susy data-set.

Support (%)	No. of concepts below $minTh$	Group-id based intersection			Cuckoo Filter			Execution time (minutes)		No. of concepts above $minTh$
		TN_g	FP_g	TP_g	TN_c	FP_c	TP_c	$< minTh$	$\geq minTh$	
25	4230	4230	0	0	0	0	0	17.69	25.36	3745
30	2575	2575	0	0	0	0	0	10.38	20.86	2450
35	480	480	0	0	0	0	0	2.95	20.83	2207
40	520	520	0	0	0	0	0	0.77	19.71	2167
55	0	0	0	0	0	0	0	0	10.99	959

Table 8: Experimental evaluation for pumsb_star data-set.

Support (%)	No. of concepts below $minTh$	Group-id based intersection			Cuckoo Filter			Execution time (seconds)		No. of concepts above $minTh$
		TN_g	FP_g	TP_g	TN_c	FP_c	TP_c	$< minTh$	$\geq minTh$	
75	1	1	0		0	0	0	0.011	0.067	2
70	238	0	238	0	238	0	0	0.8	0.62	17
65	1317	0	1317	0	1317	0	0	4.454	1.3	42
60	5107	0	5107	0	5107	0	0	8.81	1.534	68
55	52218	0	52218	0	52218	0	0	51.16	2.424	116
50	436610	0	436610	0	436610	0	0	284.6	4.515	248
45	5237012	1971124	3265888	0	3265888	0	0	1028	10.692	713
40	14946492	7643577	7302915	0	7302915	0	0	2331.3	28.764	2610

We evaluate the efficacy of group-id based intersection and cuckoo filter for different values of support threshold. When group-id based intersection function returns an empty set, no inter-machine communication is necessary to test the validity of the concept. In Table 7, the true negative column (TN_g) captures the frequency of such occurrences in susy dataset. If the function returns a non-empty set ($gIDList$) then for each group-id in $gIDList$, we need to check whether the

[1] **FIMI** repository, http://fimi.cs.helsinki.fi/. Last accessed 1 March 2020

concept is valid/invalid. If the group has at least one attribute containing the extent, then the concept is invalid. As the group-id based intersection did not detect this, it is counted as False Positive ((FP_g)). Otherwise the concept is valid (TP_g). We observe that the group-id based intersection returns an empty set (TN_g) almost 100% times for susy data-set. This means that concepts discovered in the vertical partition are valid and there is no need for inter-machine communication.

For experiments on pumsb_star data-set [2](containing 49046 objects, 2088 attributes with average object length as 50.48), we found that group-id based intersection returned a non-empty set (FP_g) close to 50% (see Table 8 for pumsb_star data-set). Here, we need to communicate with a machine whose group-id is contained in the list returned by group-id based intersection. Still group-based auxiliary structure reduces the communication cost significantly as communication is required only with the machines storing the partition corresponding to that group and not all of them. On the machine, the validity check is performed using cuckoo filter. If cuckoo filter returns false (TN_c), then the concept is valid in the vertical partition. In such cases, no comparison with the actual context of the horizontal partition is necessary. From Table 8, we see that cuckoo filter always returns true negative (TN_c).

Our experiments show that group-id based auxiliary structure decreases the inter-machine communication cost for testing validity of discovered concepts by almost 100% for susy data-set and up to 50% for pumsb_star data-set. Cuckoo filter reported 0% false positives (FP_c) thereby further minimizing the cost of validity checks.

5 Conclusions and future work

In this paper, we proposed a novel hybrid partitioning technique that uses horizontal partitioning for low support attributes and vertical partitioning for high support attributes. This technique ensures that the size of each partition does not exceed single machine's memory. Thus, it becomes possible to use the distributed LCM algorithm [6] to compute concepts in parallel in each horizontal partition. We proposed a new approach for computing concepts using group-id based auxiliary structure for the vertical partition. The experimental evaluation shows significant reduction in the inter-machine communication cost. Thus, using horizontal and vertical partitioning, our algorithm can scale to large contexts. Our future work includes coming up with an optimal threshold that takes into consideration both memory capacity and combined execution time of computing concepts in vertical and horizontal partitions.

[2] **SPMF**, https://www.philippe-fournier-viger.com/spmf/. Last accessed 1 March 2020.

Acknowledgements

This research was supported by SPARC, a Government of India Initiative under grant no. SPARC/2018-2019/P682/SL. We thank Dr. Sergei Obiedkov from HSE Moscow for his helpful comments in coming up with the paritioning strategy.

References

1. Uno T., Asai T., Uchida Y., Arimura H., “An efficient algorithm for enumerating closed patterns in transaction databases,” in Discovery Science, DS 2004, Lecture Notes in Computer Science, vol 3245. Springer, 2004.
2. Ganter B., “Two basic algorithms in concept analysis,” In: Kwuida L., Sertkaya B. (eds) Formal Concept Analysis. ICFCA 2010. Lecture Notes in Computer Science, vol 5986. Springer, 2010.
3. Krajca P., Vychodil V. , “Distributed algorithm for computing formal concepts using map-reduce framework,” In: Adams N.M., Robardet C., Siebes A., Boulicaut JF. (eds) Advances in Intelligent Data Analysis VIII. IDA 2009. Lecture Notes in Computer Science, vol 5772. Springer, 2009.
4. Xu B., de Fréin R., Robson E., Ó Foghlú M, “Distributed formal concept analysis algorithms based on an iterative MapReduce framework,” In: Domenach F., Ignatov D.I., Poelmans J. (eds) Formal Concept Analysis. ICFCA 2012. Lecture Notes in Computer Science, vol 7278. Springer, 2012.
5. B. Negrevergne, A. Termier, J. Méhaut and T. Uno, "Discovering closed frequent itemsets on multicore: Parallelizing computations and optimizing memory accesses," In: 2010 International Conference on High Performance Computing and Simulation, Caen, 2010, pp. 521-528, 2010.
6. Shravan Patel, Umang Agarwal, and Sriram Kailasam, “A Dynamic Load Balancing Scheme for Distributed Formal Concept Analysis,” in 24^{th} IEEE International Conference on Parallel and Distributed Systems, Singapore, pp. 489-496, 2018.
7. Bin Fan, Dave G. Andersen, Michael Kaminsky, and Michael D. Mitzenmacher, "Cuckoo Filter: Practically Better Than Bloom," In Proceedings of the 10^{th} ACM International on Conference on emerging Networking Experiments and Technologies (CoNEXT ’14). ACM, New York, USA, pp.75–88, 2014.
8. Kuznetsov S.O., "Learning of Simple Conceptual Graphs from Positive and Negative Examples," In: Żytkow J.M., Rauch J. (eds) Principles of Data Mining and Knowledge Discovery. PKDD 1999. Lecture Notes in Computer Science, vol 1704. Springer, 1999.
9. C. Lucchese, S. Orlando and R. Perego, "Fast and memory efficient mining of frequent closed itemsets," in IEEE Transactions on Knowledge and Data Engineering, vol. 18, no. 1, pp. 21-36, Jan. 2006.
10. Ekanayake, J., Li, H., Zhang, B., Gunarathne, T., Bae, S.-H., Qiu, J., Fox, "G.: Twister: a Runtime for Iterative MapReduce," In: Hariri, S., Keahey, K. (eds.) HPDC, pp. 810–818, ACM, 2010.
11. Xu B., de Fréin R., Robson E., Ó Foghlú M, "Distributed Formal Concept Analysis Algorithms Based on an Iterative MapReduce Framework." In: Domenach F., Ignatov D.I., Poelmans J. (eds) Formal Concept Analysis ICFCA 2012, Lecture Notes in Computer Science, vol 7278. Springer, 2012.
12. K. Yoshizoe, A. Terada, K. Tsuda, "Redesigning pattern mining algorithms for supercomputers," Corr, 2015.