Distributed Approximate Mining of Frequent Patterns

Claudio Silvestri
claudio@dsi.unive.it

Salvatore Orlando
orlando@dsi.unive.it

Dipartimento di Informatica
Università Ca’ Foscari
Via Torino, 155 - Venezia, Italy

ABSTRACT
This paper discusses a novel communication efficient distributed algorithm for approximate mining of frequent patterns from transactional databases. The proposed algorithm consists in the distributed exact computation of locally frequent itemsets and an effective method for inferring the local support of locally unfrequent itemsets. The combination of the two strategies gives a good approximation of the set of the globally frequent patterns and their supports. Several tests on publicly available datasets were conducted, aimed at evaluating the similarity between the exact result set and the approximate ones returned by our distributed algorithm as well as the scalability of the proposed method.

Categories and Subject Descriptors
H.2.8 [Database management]: Database Applications—Data mining.

General Terms

Keywords
Frequent itemsets, association mining, distributed data mining, approximate data mining.

1. INTRODUCTION
Association Rule Mining (ARM), one of the most popular topic in the KDD field [3, 10, 11, 19], regards the extractions of association rules from a database of transactions $D$. In this paper we are interested in the most computationally expensive phase of ARM, i.e the Frequent Pattern Mining (FPM) one, during which the set of all the frequent patterns (sorted itemsets) is built.

A pattern $x$ is frequent in dataset $D$ with respect to a minimum support $\minsup$, if its support is greater than $\minsup = \minsup \cdot |D|$, i.e. the pattern occurs in at least $\minsup$ transaction, where $|D|$ is the number of transaction in $D$. A k-pattern is a pattern composed of $k$ items, $\mathcal{F}_k$ is the set of all frequent k-patterns, and $\mathcal{F} = \bigcup \mathcal{F}_i$ is the set of all frequent patterns.

The computational complexity of the FPM problem derives from the exponential size of its search space $\mathcal{P}(M)$, i.e. the power set of $M$, where $M$ is the set of items contained in the various transactions of $D$. A way to prune $\mathcal{P}(M)$ is to restrict the search to itemsets whose subsets are all frequent. The APriori algorithm [4] and other derived algorithms [2, 6, 9, 14, 16, 17, 25, 20, 29] exactly exploits this pruning technique, based on the Apriori anti-monotonic principle.

Parallel (PDM) and distributed (DDM) data-mining are a natural evolution of data-mining technologies, motivated by the need of scalable and high performance systems. The main differences between these two approaches is that while in PDM data can be moved (centralized) to a tightly coupled parallel system before starting computation, DDM algorithms must deal with limited possibilities for data movement/replication, due either to specific policies or technical reasons like large network latencies [15]. Several parallel/distributed versions of sequential FPM algorithms have been proposed in the last years [5, 8, 13, 18, 23, 26, 27, 30], but most of them are not suitable for loosely coupled settings. Only a few papers discussing truly distributed FPM algorithms recently appeared in the literature [21, 22, 28].

In this paper we discuss a distributed algorithm for approximate mining of frequent patterns, $\mathcal{AP}_{\text{Interp}}$ (Approximate Partition with Interpolation), that exploit DCI [25], a state-of-the-art algorithm for FPM, as the sequential miner engine for local computations. The name “Approximate Partition” derives from the distributed computation method adopted, which is inspired by the Partition algorithm [20] and its distributed version [5].

We assume that our dataset $D$ is divided into several disjoint partitions $D_i, i = \{1, \ldots, n\}$, located on $n$ collaborating entities, where each transaction completely belongs to one of the partitions. In particular, we consider that the dataset is already partitioned, according to some business rules, among geographically distributed systems. Collaborating entities are loosely coupled, and even if available network bandwidths sometimes is not an issue, latency surely is. A fitting example is a set of insurance companies, which
are connected by the Internet and collaborate in order to
detect lots of messages with several barrier synchronizations.
Thus a small loss of accuracy is a fair trade-off for a reduced
number of communications/synchronizations.

AP_interp, our approximate FPM algorithm, computes in-
dependently local solution for each node and then merges
local results. Instead of making a second pass, as Distributed
Partition does, we propose an approximate support inference
heuristic to be used during the merge phase. Experimental
tests shows that the solution produced by AP_interp is a good
approximation of the exact global result. Unfortunately, this
method may also generate a few false positives, whose ap-
proximate supports is however very close to the exact one.

So the support of the rules extracted from these false posi-
tive patterns should not bother analysts. This is especially
true when a positive result just indicate a case which need
the attention of the operator for further investigation, as in
the case of fraud detection: if a pattern with support slightly
higher than the threshold is interesting, probably a slightly
lower one will be interesting too. A single synchronization
is required in order to collect the knowledge of \(F_1\), used
for interpolation, and \(F_2\), used by slaves for global pruning.
This is particularly important in the described distributed
setting, where the network latency is often a more critical
factor than the available bandwidth, and the reduced num-
ber of communications is worth a small reduction in the
accuracy of results.

In this paper we also introduce a novel similarity measure,
derived from the one proposed in [17], used along with the
original one in order to assess the quality of the algorithm
output.

This paper is organized as follow. Section 2 introduces
some similarity measures we will use in order to evaluate the
quality of the approximate results, while Sec. 3 describes
the AP_interp algorithm. In Sec. 4 we report and discuss
our experimental results. Finally in Sec. 5, we draw some
conclusions.

2. DEFINITIONS

The method we are proposing yields approximate results.
In particular AP_interp computes pattern supports which
may be slightly different from the exact ones, thus the re-
result set may miss some frequent pattern (false negatives)
or include some unfrequent pattern (false positives). In order
to evaluate the accuracy of the results we need a measure of
similarity between two pattern sets. A widely used one has
been introduced in [17], and is based on support difference.

**Definition 1 (Similarity).** Let \(A\) and \(B\) respectively
be the reference (correct) result set and the approximate
result set. \(sup_A(x) \in [0,1]\) and \(sup_B(y) \in [0,1]\), where \(x \in A\)
and \(y \in B\), correspond to the relative support found in \(A\)
and \(B\) respectively. Note that since \(B\) corresponds to the frequent
patterns found by the approximate algorithm under observa-
tion, \(A - B\) thus corresponds to the set of false negatives,
while \(B - A\) are the true positives.

The Similarity is thus computed as

\[
\text{Sim}_{\alpha}(A,B) = \frac{\sum_{x \in A \cap B} \max\{0, 1 - \alpha \ast |sup_A(x) - sup_B(x)|\}}{|A \cup B|}
\]

where \(\alpha \geq 1\) is a scaling parameter, which increase the ef-
tect of the support dissimilarity. Moreover, \(\frac{1}{\alpha}\) indicates the
maximum allowable error on (relative) pattern supports. We
will use the notation \(\text{Sim}()\) to indicate the default case for \(\alpha\), i.e. \(\alpha = 1\).

This measure of similarity is thus the sum of at most \(|A \cap
B|\) values in the range \([0,1]\), divided by \(|A \cup B|\). Since \(|A \cap
B| \leq |A \cup B|\), similarity lies in \([0,1]\) too.

When a pattern appears in both sets and the difference
between the two supports is greater than \(\frac{1}{\alpha}\), it does not im-
prove similarity, otherwise similarity is increased according
to the scaled difference. If \(\alpha = 20\), then the maximum al-
lowable error in the relative support is \(1/20 = 0.05 = 5\%\).
Supposing that the support difference for a particular pat-
tern \(\alpha\) is 4\%, the numerator of the similarity measure will be
increased by a small quantity: \(1 - (20 \ast 0.04) = 0.2\). When \(\alpha\)
is 1 (default value), only patterns whose support difference
is at most 100\% contribute to increase similarity. On the
other hand, when we set \(\alpha\) to a very high value, only pat-
terns with a very similar supports in both the approximate
and reference sets will contribute to increase the similarity
measure.

It is worth noting that the presence of several false pos-
itives and negatives in the approximate result set \(B\) con-
tributes to reduce our similarity measure, since this entails
an increase in \(A \cup B\) (the denominator of the \(\text{Sim}_{\alpha}\) formula)
with respect to \(A \cap B\). Moreover, if a pattern has an actual
support which is slightly less than \(\text{minsup}\) but the approxi-
mate support (\(sup_B\)) is slightly greater than \(\text{minsup}\), simi-
larity is decreased even if the computed support was almost
correct. This could be an undesired behavior. While a false
negative can constitute a big issues, because some poten-
tially important association rules will be not generated at
all, a false positive with a support very close to the exact
one could be tolerated by an analyst.

In order to overcome this issue we propose a new simi-
larity measure, \(\text{fpSim}\) (where \(fp\) stand for false positive).
Since this measure consider every pattern included in the
approximate result set \(B\) (instead of \(A \cap B\)), it can be used
in order to assess whether false positives have an approxi-
mate support value close to the exact one or not. A high
value of \(\text{fpSim}\) compared with a smaller value of \(\text{Sim}\) sim-
ply means that in the approximate result set \(B\) there are
several false positive with a true support close to \(\text{minsup}\).

**Definition 2 (fpSimilarity).** Let \(A\) and \(B\) respec-
tively be the reference (correct) result set and the approxi-
mate result set. \(sup_B(x) \in [0,1]\), where \(x \in B\), corre-
responds to the support found in result sets \(B\), while \(sup(x) \in [0,1]\)
is the actual support of the same pattern. \(\text{fpSimilarity}\) is thus
computed as

\[
\text{fpSim}_{\alpha}(A,B) = \frac{\sum_{x \in A \cap B} \max\{0, 1 - \alpha \ast |sup(x) - sup_B(x)|\}}{|A \cup B|}
\]

where \(\alpha \geq 1\) is a scaling parameter. We will use the notation
\(\text{Sim}()\) to indicate the default case for \(\alpha\), i.e. \(\alpha = 1\).

Note that the numerator of this new measure considers
all the patterns found in the set \(B\), thus also false positives.
Hence finding a pattern with a support close to the true
one is considered a "good" result in any case, even if this
pattern is not actually frequent. For example, suppose that
minimum support threshold is 50\% and \(x\) is an unfrequent
pattern such that \(sup(x) = 49.9\). If \(sup_B(x) = 50\%\), it will
result to be a false positive. However, since \(sup_B(x)\) is very
close to the exact support \( \text{sup}(x) \), the value of \( fpSim_\alpha() \) will be increased.

In Definition 2 we used \( \text{sup}(x) \) instead of \( \text{sup}_\delta(x) \) to indicate the actual support of itemset \( x \) since it is possible, as in the example case, that a pattern is present in \( B \) even if it is not frequent (hence not present in \( A \)).

In both definitions above we used \( \text{sup}(x) \) to indicate the (relative) support, ranging from 0 to 1. In the remainder of the paper, in particular in the algorithm description, we will also use the notation \( \sigma(x) = \text{sup}(x) \cdot |D| \) to indicate the support count (absolute support), ranging from 0 to the total number of transactions.

When bounds on the support of each pattern are available, an intrinsic measure of the correctness of the approximation is the average width of the interval between the upper bound and the lower bound.

**Definition 3 (Average Support Range).** Let \( B \) be the approximate result set, \( \text{sup}(x) \) the exact support for pattern \( x \) and \( \text{sup}(x)_\text{lower} \) and \( \text{sup}(x)_\text{upper} \) the lower and upper bounds on \( \text{sup}(x) \), respectively. The average support range is thus defined as:

\[
\text{ASR}(B) = \frac{1}{|B|} \sum_{x \in B} \text{sup}(x)_\text{upper} - \text{sup}(x)_\text{lower}
\]

Note that, while this definition can be used for every approximate algorithm, how to compute \( \text{sup}(x)_\text{lower} \) and \( \text{sup}(x)_\text{upper} \) is algorithm specific. In the next section we will present a way that is suitable for the class of algorithms containing the one we are proposing.

Other, less accurate, similarity measures can be borrowed from the Information Retrieval theory:

**Definition 4 (Recall & Precision).** Let \( A \) and \( B \) respectively be the reference (correct) result set and the approximate result set. Note that since \( B \) corresponds to the support of patterns observed in the approximate algorithm under observation, \( A \cap B \) thus corresponds to the set of false negatives, while \( B - A \) are the false positives.

Let \( P(A, B) \in [0,1] \) be the Precision of the approximate result, defined as follows:

\[
P(A, B) = \frac{B \cap A}{B}
\]

Hence the Precision is maximal \( P(A, B) = 1 \) if \( B \cap A = B \), i.e. the approximate result set \( B \) is completely contained in the exact one \( A \), and no false positives occurs.

Let \( R(A, B) \in [0,1] \) be the Recall of the approximate result, defined as follows:

\[
R(A, B) = \frac{B \cap A}{A}
\]

Hence the Recall is maximal \( R(A, B) = 1 \) if \( B \cap A = A \), i.e. the exact result set \( A \) is completely contained in the approximate one \( B \), and no false positive occurs.

According to our remarks above concerning the benefits of the \( fpSim \) measure (Def. 2), we have that a “good” approximate result should be characterized by a very high Recall, where the supports of the possible false positive patterns should be however very close to the exact ones. Conversely, in order to minimize the standard measure of similarity (Def. 1), we need to maximize both Recall and Precision, while keeping small the difference in the approximate supports of frequent patterns.

### 3. THE ALGORITHM

Our \( \text{AP}_{\text{interp}} \) (Approximate Partition with Interpolation) algorithm was inspired by Partition [20], a sequential algorithm which divides the dataset in several partitions processed independently.

#### 3.1 The Distributed Partition algorithm

The basic idea exploited by Partition is the following: each globally frequent pattern must be locally frequent in at least one partition. This guarantees that the union of all local solutions is a superset of the global solution. However, one further pass over the database is necessary to remove all false positives, i.e. patterns that result locally frequent but globally infrequent.

Obviously, Partition can be straightforwardly implemented in a distributed setting with a master/slave paradigm [5]. Each slave becomes responsible of a local partition, while the master performs the sum-reduction of local counters (first phase) and orchestrates the slaves for computing the missing local supports for potentially globally frequent patterns (second phase) to remove patterns having global support less than \( \text{minsup} \) (false positive patterns collected during the first phase).

While the Distributed Partition algorithm gives the exact values for supports, it has pros and cons with respect to other distributed algorithms. The pros are related to the number of communications/synchronizations: other methods as count-distribution [13, 29] require several communications/synchronizations, while the Distributed Partition algorithm only requires two communications from the slaves to the master, and a single one from the master to the slaves. The cons are the size of messages exchanged, and the possible additional computation performed by the slaves when the first phase of the algorithm produces false positives. Consider that, when low absolute minimum supports are used, it is likely to produce a lot of false positives due to data skew present in the various dataset partitions [17]. This has a large impact also on the cost of the second phase of the algorithm too: most of the slaves will participate in counting the local supports of these false positives, thus wasting a lot of time.

One naive work-around, that we will name Distributed One-pass Partition, consists in stopping Distributed Partition after the first-pass. So in Distributed One-pass Partition each slave independently computes locally frequent patterns and sends them to the master which sum-reduces the support for each pattern and writes in the result set only patterns having the sum of the known supports greater than (or equal to) \( \text{minsup} \). Distributed One-pass Partition has obvious performance advantages vs Distributed Partition. On the other hand it yields a result which is approximate. This is formalized in the following lemma.

**Lemma 5 (Bounds on Support after First Pass).** Let \( P = 1, \ldots, N \) be the set of the \( N \) partition indexes. Then let \( \text{fpart}(x) = \{ j \in P | \sigma_j(x) \geq \text{minsup} \cdot |D_j| \} \) be the set of indexes of the partitions where the pattern \( x \) is frequent and
let \( f_{\text{part}}(x) = (P \setminus f_{\text{part}}) \) be its complement. The support for a pattern \( x \) is greater than or equal to the support computed by the \textit{Distributed One-pass Partition} algorithm:

\[
\sigma(x)^{\text{lower}} = \sum_{j \in f_{\text{part}}(x)} \sigma_j(x)
\]

and is less than or equal to \( \sigma(x)^{\text{upper}} \) plus the maximum support the same pattern can have in partitions where it is not frequent:

\[
\sigma(x)^{\text{upper}} = \sigma(x)^{\text{lower}} + \sum_{j \in f_{\text{part}}(x)} \text{minsup} \cdot |D_j| - 1
\]

Note that when a pattern does not result frequent in a partition, its actual local support can be at most equal to the local minimum support threshold minus one.

We can easily transform the two absolute bounds defined above into the corresponding relative ones:

\[
\sup(x)^{\text{upper}} = \frac{\sigma(x)^{\text{upper}}}{|D|}, \quad \sup(x)^{\text{lower}} = \frac{\sigma(x)^{\text{lower}}}{|D|}
\]

These bounds can be used to calculate the Average Support Range of Definition 3 (ASR(B)). Any approximate algorithm based on \textit{Distributed One-pass Partition} will yield results with at most this average error on all the supports.

The main issue with \textit{Distributed One-pass Partition} is that for every pattern the computed support is a very conservative estimate, since it always chooses the lower bounds to approximate the results. Generally any algorithm returning a support value between the bounds will have better chances of being more accurate. Following this idea we devised a new algorithm based on \textit{Distributed One-pass Partition} which uses a smart interpolation of support. Moreover, it is resilient to skewed item distributions.

### 3.2 The AP\textsubscript{interp} algorithm

\textit{AP}\textsubscript{interp}, the distributed algorithm we propose in this paper, tries to overcome some of the problems encountered by \textit{Distributed One-pass Partition} when the data skew between the data partitions is high. The more evident is that several false positives could be generated, increasing the resource utilization and the execution time of both \textit{Distributed Partition} and \textit{Distributed One-pass Partition}. \textit{AP}\textsubscript{interp} addresses this issue by means of global pruning based on partial knowledge of \( \mathcal{F}_2 \): each locally frequent pattern which contains a globally non-frequent 2-pattern will be locally removed from the set of frequent patterns before sending it to the master and performing next candidate generation.

Moreover this skew might cause a globally frequent pattern \( x \) to result infrequent on a given partition \( D_i \) only. In other words, since \( \sigma_i(x) < \text{minsup} \cdot |D_i| \), \( x \) will not be returned as a frequent pattern by the \( i \)-th slave. As a consequence, the master of \textit{Distributed One-pass Partition} can not count on the knowledge of \( \sigma_i(x) \), and thus can not exactly compute the global support of \( x \). Unfortunately, in \textit{Distributed One-pass Partition} the master might also deduce that \( x \) is not globally frequent, because \( \sum_{j \in f_{\text{part}}(x)} \sigma_j(x) < \text{minsup} \cdot |D| \). \textit{AP}\textsubscript{interp} thus allows the master to infer an approximate value for this unknown \( \sigma(x) \) by exploiting an interpolation method. The master bases its interpolation reasoning on the knowledge of:

- the exact support of each single item on all the partitions, and
- the average reduction of the support count of pattern \( x \) on all the partitions where \( x \) resulted actually frequent (and thus returned to the master by the slave), with respect to the support of the least frequent item contained in \( x \):

\[
\text{avg}_i(\text{reduct}(x)) = \frac{\sum_{j \in f_{\text{part}}(x)} \sigma_j(x)}{|f_{\text{part}}(x)|}
\]

where \( f_{\text{part}}(x) \) corresponds to the set of data partitions \( D_j \) where \( x \) actually resulted frequent, i.e. where \( \sigma_j(x) \geq \text{minsup} \cdot |D_j| \).

The master can thus deduce the unknown support \( \sigma_i(x) \) on the basis of \( \text{avg}_i(\text{reduct}(x)) \) as follows:

\[
\sigma_i(x)^{\text{interp}} = \min_{\text{item}\in x} \left( \sigma_i(\text{item}) \times \text{avg}_i(\text{reduct}(x)) \right)
\]

It is worth remarking that this method works if the support of larger itemsets decrease similarly in all the dataset partitions, so that an average reduction factor (different for each pattern) can be used to interpolate unknown values. Finally note that, as regards the interpolated value above, we expect that the following inequality holds:

\[
\sigma_i(x)^{\text{interp}} < \text{minsup} \cdot |D_i| \quad (1)
\]

So, if we obtain that \( \sigma_i(x)^{\text{interp}} \geq \text{minsup} \cdot |D_i| \), this interpolated result can not be accepted. If it was true, the exact value \( \sigma_i(x) \) should have already been returned by the \( i \)-th slave. Hence, in those few cases where the inequality (1) does not hold, the interpolated value returned will be:

\[
\sigma_i(x)^{\text{interp}} = \left( \text{minsup} \cdot |D_i| \right) - 1
\]

We expect that the proposed interpolation schema yields a better approximation of exact results than \textit{Distributed One-pass Partition}. The support values computed by the latter algorithm are, in fact, always equal to the lower bounds of the intervals containing the exact support of any particular pattern. Hence any kind of interpolation producing an approximate result set, whose supports are between the interval bounds, should be, generally, more accurate than peeking always its lower bound. Obviously several other way of computing a support interpolation could be devised. Some are really simple as the average of the bounds while others are complex as counting inference, used in a different context in [24]. We chose this particular kind of interpolation because it is simple to calculate, since it is based on data that we already maintain for other purposes, and it is aware of the data partitioning enough to allow for accurate handling of datasets characterized by heavy data-skew on item distributions.

We can finally introduce the pseudo-code of \textit{AP}\textsubscript{interp}. As in \textit{Distributed Partition} we have a master and several slaves, each in charge of a horizontal partition \( D_i \) of the original dataset. We have a single synchronization, during which the slaves send information to the master about the counts of single items and locally frequent 2-itemsets. The master then communicates to the slaves an approximate global knowledge on \( \mathcal{F}_2 \), used by the slaves to prune candidates for the rest of the mining process. Finally, once received information about all locally frequent patterns, the master exploits the interpolation method sketched above for inferring unknown support counts.
Note that when a pattern is locally frequent in all the partitions, the master is able to exactly determine its support. Otherwise an approximate inferred support value is produced, along with an upper bound and a lower bound for that support.

Slave i:
1. Compute local Single_Counts\_i \( F_{i} \) and \( F'_{2} \).
2. Send local partial results to the master.
3. Receive the global approximation \( F'_{2} \) of \( F_{2} \).
4. Continue computation, by using \( F'_{2} \) for pruning candidates.
5. Send local results to the master. If computation is over, send an empty set.

Master:
1. Receive local partial results Single_Counts\_i and \( F'_{2} \) from all the slaves.
2. Compute the exact \( F_{1} \), on the basis of the local counts of single items.
3. Compute the approximate
\[
F'_{2} = \{ x \in \bigcup_{i} F_{i} \mid \sum_{i} \text{count}_i(x) \geq \text{minsup} \cdot |D| \}
\]
where if \( x \in F_{i} \) then \( \text{count}_i(x) \) is equal to \( \sigma_i(x) \), or is equal to \( \sigma_i(x) \) interp otherwise.
4. Send \( F'_{2} \) to all the slaves.
5. Receive local results from all the slaves (empty for slaves terminated before the third iteration).
6. Compute and return, for each \( k \), the approximate
\[
F'_{k} = \{ x \in \bigcup_{i} F_{i} \mid \sum_{i} \text{count}_i(x) \geq \text{minsup} \cdot |D| \}
\]
where if \( x \in F_{i} \) then \( \text{count}_i(x) \) is equal to \( \sigma_i(x) \), or is equal to \( \sigma_i(x) \) interp otherwise.

In the pseudo-code \( F'_{i} \) denotes the set of frequent \( k \)-patterns in partition \( i \) (or globally when \( i \) is not present). \( F'_{k} \) indicate an approximation of \( F_{k} \) and Single_Counts\_i is the support of all 1-patterns in partition \( i \).

For the sake of simplicity, some detail of the algorithm has been altered in the pseudo-code. In particular, points 4 and 5 of the slave pseudo-code are a over-simplification of the actual code: pattern are sent, asynchronously, as soon as they are available in order to optimize communication. Each slave terminates when, at iteration \( k \), less than \( k + 1 \) pattern are frequent; this is equivalent to checking emptiness of \( F'_{k+1} \), but more efficient. On the other side, the master continuously collects results from still active slaves and processes them as soon as all expected result sets of the same length arrive.

4. EXPERIMENTAL EVALUATION

4.1 Experimental environment

The experiments were performed on a cluster of seven high-end workstations, each equipped with an Intel Xeon 2 G Hz, 1 GB of RAM memory and local storage. In all our tests we mapped a single process (either master or slave) to each node. This system offers communications with good latency (a dedicated Fast Ethernet). However, since \( AP_{\text{interp}} \) requires just one synchronization and all communication are pipelined, its communication pattern should be suitable even for a distributed system characterized by an high latency network.

4.2 Experimental data

We performed several tests using datasets from the FIMI’03 contest [1]. We randomly partitioned each dataset twice and used the resulting partitions as input data for different slaves. In particular each dataset was divided in a roughly uniform way and also in a non uniform way (with a size difference of an order of magnitude between the smallest and the biggest partition). The first ones, the balanced partitioned datasets, were used in order to assess speedup for the tests on our parallel testbed. Table 1 shows a list of these datasets along with their cardinality. Each dataset is also identified by a short code, starting with U in case of unbalanced partitioning. The number of partitions is not reported in this table, since it depends on the number of slaves involved in the specific distributed test. For each dataset we computed the reference solution using DCI [25], an efficient sequential algorithm for frequent pattern mining (FPM).

Table 1: Datasets used in experimental evaluation. When a datasets is referenced by a keyword prefixed by U (see Reference column), this means that it was partitioned in an unbalanced way, with partitions of significantly different sizes.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Reference</th>
<th># Trans.</th>
</tr>
</thead>
<tbody>
<tr>
<td>accidents-bal</td>
<td>A</td>
<td>340183</td>
</tr>
<tr>
<td>accidents-unbal</td>
<td>UA</td>
<td>340183</td>
</tr>
<tr>
<td>kosarak-bal</td>
<td>K</td>
<td>990002</td>
</tr>
<tr>
<td>kosarak-unbal</td>
<td>UK</td>
<td>990002</td>
</tr>
<tr>
<td>mushroom-bal</td>
<td>M</td>
<td>8124</td>
</tr>
<tr>
<td>mushroom-unbal</td>
<td>UM</td>
<td>8124</td>
</tr>
<tr>
<td>retail-bal</td>
<td>R</td>
<td>88162</td>
</tr>
<tr>
<td>retail-unbal</td>
<td>UR</td>
<td>88162</td>
</tr>
<tr>
<td>pumbs-bal</td>
<td>P</td>
<td>49046</td>
</tr>
<tr>
<td>pumbs-unbal</td>
<td>UP</td>
<td>49046</td>
</tr>
<tr>
<td>pumbs-star-bal</td>
<td>PS</td>
<td>49046</td>
</tr>
<tr>
<td>pumbs-star-unbal</td>
<td>UPS</td>
<td>49046</td>
</tr>
<tr>
<td>connect-bal</td>
<td>C</td>
<td>67557</td>
</tr>
</tbody>
</table>

4.3 Experimental Results

The experiments were run for several minimum support values and for different partitioning on each dataset. In particular, except when showing the effects of minimum support and number of partition change, we reported results corresponding to three and six partitions and to the two smallest minimum support thresholds used, usually characterized by a difference of about one order of magnitude in execution time.

Table 2 shows a summary of computation results for all datasets, obtained for three and six partitions using two different minimum support values. The first four columns contain the code of the dataset and the parameters of the test. The next two columns contain the number of frequent patterns contained in the approximate solution and the execution time. The average support range column contains the average distance between the upper and lower bounds for the support of the various patterns, expressed as a percentage of the number of transactions in the dataset (see Definition 3). The following columns show the precision and recall metrics and the number of false positives/negatives. As expected, there are really few false negatives and consequently the value of Recall is close to 100%, but the Precision is slightly smaller. Unfortunately, since these metrics do not take into
account the support, a false positive having true support really close to the threshold has the same weight than one having a very small support. The last columns contain the similarity measure for the approximate results introduced in Definitions 1 and 2. The very high value of the fpSim proves that false positives have a support close to the exact one (but smaller than the exact one, so that they are actually not frequent). This behavior, i.e. a lot of false positives with a value of fpSim close to 100%, is particularly evident for datasets K and UK.

Figure 1 shows a plot of the fpSim measure obtained for different datasets partitioned among a variable number of slaves. As expected the similarity is higher when the dataset is partitioned in few partitions. Anyway in most cases there is no significant decrease.

We have also compared the similarity of the approximate result obtained using support interpolation to the Distributed One-pass Partition one. The results are shown in Figure 2. The proposed heuristic for support interpolation does improve similarity, in particular for small minimum support values. Since no false positives are produced by Distributed One-pass Partition, in this case fpSim would be identical to Sim, thus this measure is plotted just for the APInterp algorithm.

Finally we have verified the speedup of the APInterp algorithm, using only uniformly sized partitions. Figure 3 shows the measured speedup when an increasing number of slaves is exploited. Note that when more slaves are used, the dataset has to be partitioned accordingly.

The APInterp algorithm performed worse on dense datasets, such as Connect, where too many locally frequent patterns are discarded when we add slaves. On the other hand in some cases we obtained also superlinear speedups. This should be due to the approximate nature of our algo-
rithm: the support of several pattern could be computed even if some slaves does not participate in the elaboration.

5. CONCLUSIONS

In this paper we have discussed \( \text{AP}_{\text{Interp}} \), a new distributed algorithm for approximate frequent pattern mining. \( \text{AP}_{\text{Interp}} \) exploits a novel interpolation method to infer unknown counts of some patterns, which are locally frequent in some dataset partitions. For dataset partitioning characterized by high data skew, the \( \text{AP}_{\text{Interp}} \) approach is able to strongly improve the accuracy of the approximate results.

Our tests prove that this method is particularly suitable for several (mainly sparse) datasets: it yields a good accuracy and scale nicely. The best approximate results obtained for the various datasets were characterized by a similarity above 99%. Even if some false positives are found, the high similarity value computed on the whole result set proves that the exact supports of these false positives are actually close to the support threshold, and thus of some interest to the analyst.

Moreover \( \text{AP}_{\text{Interp}} \) is communication efficient, since synchronization occurs just once as in a naive Distributed Partition. Furthermore the accuracy of the results is better than in Distributed One-pass Partition case. The main reason for this is that the Distributed One-pass Partition algorithm yields, for any patterns, a support value which is the lower bound of the interval in which the exact support is included. Hence the count estimated by our algorithm, which falls between the lower and upper bounds, is generally closer to the exact count than the lower bound. Furthermore the proposed interpolation schema does not increase significantly the overall space/time complexity and is resilient to heavy skew in the distribution of items.

6. ACKNOWLEDGEMENTS

This work was partially supported by the Italian MIUR project "Enhanced Content Delivery (ECD)".

The datasets used during the experimental evaluation are some of those used for the FIMI’03 (Frequent Itemset Mining Implementations) contest [1]. We thank the owners of this data and people who made them available in current format. In particular Karolien Geurts [12] for Accidents, Ferenc Bodon for Kosarok, Tom Brijs [7] for Retail and Roberto Bayardo for the conversion of UCI datasets.

7. REFERENCES


