The Segment Support Map: Scalable Mining of Frequent Itemsets

Laks V.S. Lakshmanan
Concordia University
and IIT — Bombay
laks@cs.concordia.ca

Carson Kai-Sang Leung
The University of
British Columbia
kleung@cs.ubc.ca

Raymond T. Ng*
The University of
British Columbia
rng@cs.ubc.ca

ABSTRACT

Since its introduction, frequent set mining has been generalized to many forms, including online mining with Carma, and constrained mining with CAP. Regardless, scalability is always an important aspect of the development. In this paper, we propose a novel structure called *segment support map* to help mining of frequent itemsets of the various forms. A light-weight structure, the segment support map improves the performance of frequent-set mining algorithms by: (i) obtaining sharper bounds on the support of itemsets, and/or (ii) better exploiting properties of constraints. Our experimental results show the effectiveness of the segment support map.

Keywords

Scalable data mining, association rule, frequent sets

1. INTRODUCTION

Since its introduction [1], the problem of mining association rules, and the more general problem of finding frequent sets, from large databases has been the subject of numerous studies. Those studies can be broadly divided into two categories:

1. *Scalability*: The central question considered is how to compute the conventional association rules as efficiently as possible. Studies in this category can be further classified into three subgroups: (i) fast algorithms based on the levelwise Apriori framework [3; 11]; (ii) partitioning [17; 19], and sampling [22]; (iii) incremental updating and parallel algorithms [2; 6; 8; 16].

2. *Functionality*: The central question considered is what (kind of rules) to compute. Studies in this category can be further classified into two "generations". Studies in the first generation basically considered the data mining exercise in isolation. Examples include multi-level association rules [9], quantitative and multi-dimensional rules [7; 14; 21], correlations and causal structures [5; 20], mining long patterns [4], and ratio rules [12]).

   Studies in the second generation have explored how data mining can best interact with other key components in the broader picture of knowledge discovery. One key component is the DBMS, and some studies (e.g., the integration of association rule mining with relational DBMS [18], query flocks [23]) explored how association rule mining can handshake with the DBMS most effectively. Another component, which is arguably even more important when it comes to knowledge discovery, is the *human user*. Studies of this kind try to provide much better support for: (i) user interaction (e.g., dynamically change the parameters midstream); and (ii) user guidance and focus (e.g., limit the computation to what interests the user). With respect to user interaction, Hidber [10] proposed a "continuous/online" mining algorithm, called Carma, which provides the user with continuous feedback on the numerous frequent sets being computed, and permits the user to change the support threshold dynamically. With respect to user focus, Ng et al. [15; 13] proposed a constrained frequent set mining framework within which the user can use a rich set of constraints to guide the mining to find only those rules satisfying the constraints. The CAP algorithm was developed, which exploits the constraints to give as much pruning as possible.

The contribution of this work is to study how scalability can be enhanced in a mining environment, particularly one centered around the human user. To truly engage the user in the data mining process, it is imperative that responses to the user be delivered as "real time" as possible. Towards this objective, we introduce in this paper a novel structure called *segment support map (SSM)*. A light-weight and easy-to-compute structure, the SSM improves the efficiency of Cama by enabling the algorithm to obtain sharper upper bounds on the support of itemsets, which in turn help reduce the number of candidate itemsets that need to be counted for support. Experimental results show that the SSM can improve the efficiency significantly.

Furthermore, even though we present the SSM as a feature to enhance Carma, we show at the end of this paper that the SSM can be applied to the conventional mining of association rules using the Apriori algorithm (and hence, to many related data mining tasks using the Apriori algorithmic framework). Just as well, the SSM is also applicable to

*Person handling correspondence: Raymond T. Ng, Department of Computer Science, The University of British Columbia, 2366 Main Mall, Vancouver, BC, Canada V6T 1Z4; Phone: (604)822-2394; Fax: (604)822-5485.

```
Procedure Carma-PhaseI (TDB = ⟨t_1, ... , t_n⟩, support sequence ⟨σ_1, ... , σ_n⟩) {
      V = ∅; /* initialization */
(1)   for i from 1 to n { /* start scanning TDB */
(2)      for all v ∈ V with v ⊆ t_i, increment count(v);
(3)      for all v ⊆ t_i with v ∉ V { /* insert if appropriate */
(4)         if (for all w ⊂ v, w ∈ V and maxSupport(w) ≥ σ_i) { /* insert v */
(5)            add v to V;
(6)            firstTrans(v) = i; count(v) = 1;
(7)            if (v a singleton itemset) maxMissed(v) = 0;
(8)            else maxMissed(v) = min{ base, (maxMissed(w) + count(w) − 1) | w ⊂ v};
            } /* end if */
         } /* end for */
(9)      /* pruning step, details omitted here */ retain v if maxSupport(v) ≥ σ_i;
      } /* end for */
      return V;
}
```

Figure 1: PhaseI of Algorithm Carma

the CAP algorithm. In fact, not only does the SSM help to provide better pruning by the frequency/support constraint, but it also exploits any *anti-monotone* constraint in a similar fashion.

Among all the scalability studies listed above, the proposed technique is the most related to the partitioning algorithms. In Park et al.'s work [17], a hash table is used to partition itemsets of the same cardinality (e.g., 2) into bins. But the SSM is different in at least two key aspects. First, the SSM partitions transactions, not itemsets. Second, the SSM is intended to be a static data structure, whereas the hash table is created dynamically. In Savasere et al.'s work [19], the Partition algorithm divides transactions into partitions, and computes *all* itemsets that are frequent locally within the partition. In contrast, the SSM is a data structure, not an algorithm. As will be shown later, the SSM is valuable to many algorithms, including the conventional Apriori algorithm, Carma and CAP.

The paper is organized as follows. In the next section, we give an overview of Carma. Section 3 presents an overview of the SSM. In Section 4, we show how to use the SSM in Carma. Section 5 shows the experimental results. In Section 6, we discuss the usefulness of the SSM in other frequent-set mining algorithms like Apriori and CAP. Finally, conclusions are presented in Section 7.

2. BACKGROUND: OVERVIEW OF CARMA

To allow the user to dynamically adjust the support threshold, Carma [10] is divided into PhaseI and PhaseII. During PhaseI, Carma constructs a lattice V, called the *support lattice*. For each itemset $v \in V$, Carma maintains three integer counters: (i) $firstTrans(v)$, storing the transaction index at which v was inserted into the lattice V; (ii) $count(v)$, storing the support of v *since* v was inserted; and (iii) $maxMissed(v)$, storing an upper bound on the support of v *before* v was inserted.

Suppose that the i-th transaction t_i has just been read. Then $maxSupport(v) = (maxMissed(v) + count(v))/i$ gives an upper bound on the support of v in the first i transactions. At this point after reading t_i, two main operations can take place. On the one hand, as shown in Step (2) of Figure 1, Carma increments $count(v)$ for each itemset v currently in V and is a subset of transaction t_i. On the other hand, as shown in Step (3), if v, which is a subset of t_i, is not currently in the lattice V, then v is inserted provided that the following condition is met:

$$\forall\, w \subset v,\ w \in V \text{ and } maxSupport(w) \geq \sigma_i \qquad (1)$$

where σ_i is the support threshold at the point after t_i has just been read. If the above condition is met and v is inserted, the three counters associated with v are initialized in Steps (6), (7) and (8). Of the three counters, the initialization of $maxMissed(v)$ requires the most attention. If v is a singleton itemset, the condition in (1) guarantees that this transaction is the first transaction containing v, in which case $maxMissed(v)$ should be initialized to 0. If v is not a singleton itemset, then $maxMissed(v)$ is initialized based on: (i) the counters associated with all subsets w of v, and (ii) a quantity called *base*, which is an estimate defined according to the support threshold sequence. Here we omit the precise definition of *base*, because it is complicated and immaterial to the rest of the paper.

From time to time, Carma may invoke the pruning step (i.e., Step (9)) to prune the lattice V during Phase I; details of the pruning step is omitted. Basically, for each non-singleton itemset v currently in V with $maxSupport(v) < \sigma_i$, Carma removes v from V. Finally, to complete the description of Carma, PhaseII re-scans the transaction database TDB to get precise counts for all itemsets $v \in V$.

3. SEGMENT SUPPORT MAP

To provide for a human-centered and exploratory environment for data mining, it is imperative that the system be efficient and be able to deliver responses to the user as "real time" as possible, so as not to lose the attention of the user. Towards this objective, we introduce in this section a light-weight structure called *segment support map (SSM)*. The structure consists of the support count for all singleton itemsets in each segment in the transaction database.

Let the transaction database TDB be divided *arbitrarily* into m partitions, called segments. A *segment support map (SSM)* is a structure consisting of $support_k(\{a\})$ for all *singleton* itemsets $\{a\}$, where $support_k(\{a\})$ denotes the support of $\{a\}$ in the k-th segment for $1 \leq k \leq m$. The support of $\{a\}$, by definition, is then $\sum_{k=1}^{m} support_k(\{a\})$. While the SSM only contains the segment supports of singleton itemsets, it can be used to give an upper bound on the support of an arbitrary itemset v:

$$estsup(v) = \sum_{k=1}^{m} min\{support_k(\{a\}) \mid a \in v\} \qquad (2)$$

where $estsup(v)$ denotes the estimated segment support of v. For example, suppose there are 4 segments in the SSM, and the (actual) segment supports for items a, b and c are as shown:

	seg 1	seg 2	seg 3	seg 4	TDB
$\{a\}$	20	10	5	20	55
$\{b\}$	5	20	20	20	65
$\{c\}$	10	20	10	10	50

By equation (2), $estsup(\{a, b\})$ is $min\{20, 5\}+$ $min\{10, 20\}$ + $min\{5, 20\}+$ $min\{20, 20\}$, for a total of 40.
Similarly, by equation (2), the support of $\{a, b, c\}$ is bounded from above by 30. On the other hand, if we did not use the SSM (i.e. the number of segments is 1), then the estimated support for $\{a, b\}$ would have been $min\{55, 65\} = 55$, while that for $\{a, b, c\}$ would have been $min\{55, 65, 50\} = 50$.
Clearly, the upper bound $estsup(v)$ provided by the SSM can be made tighter in two ways. The first way is to increase the number of segments m. The amount of storage space required is then increased linearly. In an extreme case, the number of segments m equals to the number of transactions n in the database. Then, the upper bound $estsup(v)$ is so tight that it becomes the *actual* support count of v.
The second way is to generalize the SSM to store not only the actual segment supports of singleton itemsets, but also those of itemsets of higher cardinalities (i.e. itemsets of sizes greater than one). For example, for the support of $\{a, b, c\}$, the actual segment supports based on $\{a, b\}$, $\{a, c\}$ and $\{b, c\}$ provide a tighter upper bound than those based on $\{a\}$, $\{b\}$ and $\{c\}$. The price is that the amount of storage space required is then increased exponentially with respect to the sizes of the stored itemsets. Thus, in this paper, we restrict our consideration to segment supports of singleton itemsets. In this way, we keep the SSM as a very light-weight structure. For instance, for a domain of 10 000 items, even 50 segments require the storage of only 500 000 integers. We also note that the SSM is a fixed structure that can be computed *once* at "compile-time", and can be used regardless of how the support threshold is changed dynamically *multiple times* during "exploration-time". Finally, there is no searching involved when the SSM is used. Once the singleton itemsets are enumerated based on some canonical ordering, the itemsets themselves (i.e. the first column of the above table) need not be stored, and direct addressing into the SSM makes the computation of equation (2) very efficient.

4. USING THE SSM IN CARMA

In this section, we show how the SSM can be applied to the transaction-wise frequent-set mining algorithm, Carma, to effect more pruning. In the rest of the paper, we use Carma(w/ SSM) to denote Carma with the SSM, and use Carma(w/o SSM) to denote the original Carma.
Recall from Section 2 and from Steps (4) and (9) of Figure 1 that the only source of pruning in Carma is based on the condition $maxSupport(w) < \sigma_i$. The term $maxSupport(v)$ is defined as $(maxMissed(v) + count(v))/i$. In turn, the term $maxMissed(v)$ is mainly initialized as in Step (8):

$$maxMissed(v) = min \; \{ base, \; (maxMissed(w) + count(w)) \mid w \subset v\} \qquad (3)$$

However, there are two main weaknesses/problems with this pruning strategy of Carma.
The first problem is that the right-hand-side of the above equation is too loose an upper bound. And because of the recursive nature of the equation, a loose initialization of $maxMissed(v)$ has a compound effect that causes the bound for $maxMissed(u)$, for all supersets u of v, to be quite loose as well.
The second problem is with the division by i in the term $(maxMissed(v) + count(v))/i$. Basically, this is a uniform distribution assumption — assuming that the transactions supporting v are uniformly distributed, i.e. whatever happens in the first i transactions will continue to hold for the remaining transactions. In practice, this assumption is hardly true. For example, if the transaction database consists of supermarket transactions over a few months, items sold during the summer can be very different from those sold in the fall. Thus, pruning based on this assumption can be highly inaccurate. One consequence is that an itemset might be pruned too early, and needs to be re-inserted afterwards; another consequence is that an itemset might be kept for too long, while it should have been pruned earlier.
The SSM can record whatever variations that may exist in the support of an itemset from different parts of the transaction database. In this section, we show how it can be used to tighten the $maxMissed(v)$ bound, as well as to effect pruning while discarding the uniform distribution assumption.

4.1 Tightening the $maxMissed(v)$ Bound

Figure 2 depicts the various events during a transaction scan. Here we assume that the k-th segment contains h transactions. Suppose a certain itemset v is inserted after the j-th transaction of this segment (i.e., the transaction t_{q+j} where $j < h$) has been read. Exactly as in Carma(w/o SSM), Carma(w/ SSM) initializes $maxMissed(v)$ as per equation (3). But in the presence of the SSM, Carma(w/ SSM) can divide $maxMissed(v)$ into two components:

1. the part from the first $k - 1$ segments, which we refer to as $maxMissed_1(v)$; and

2. the part from within the first j transactions in the k-th segment, which we refer to as $maxMissed_2(v)$.

Let us examine $estsup(v)$ in equation (2) in the finer granularity of individual segments. Specifically, let $estsup_k(v)$ denote the estimated segment support of v (via the SSM) from the k-th segment, i.e.,

$$estsup_k(v) = min\{support_k(\{a\}) \mid a \in v\} \qquad (4)$$

Then, the first component $maxMissed_1(v)$ is simply:

$$maxMissed_1(v) = \sum_{u=1}^{k-1} estsup_u(v) \qquad (5)$$

which is the sum of the estimated segment supports of v from the first $k - 1$ segments.
The second component $maxMissed_2(v)$ is harder to estimate, because this corresponds to only part of a segment.

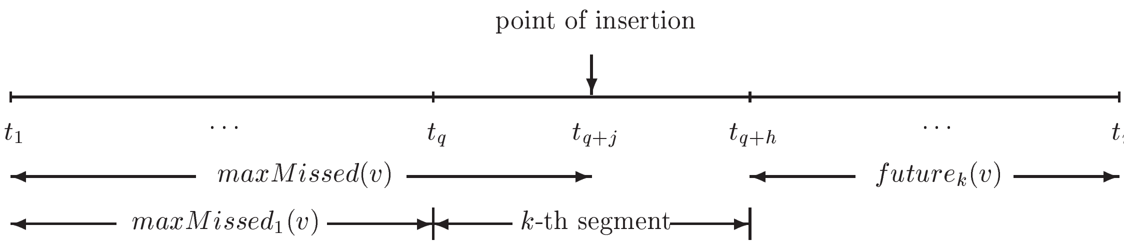


Figure 2: Using the SSM for $maxMissed(v)$ and $future_k(v)$

Clearly, $maxMissed_2(v)$ is bounded from above by j, the number of transactions in the k-th segment read so far. Depending on how far the current segment has been read, such a bound may be too loose. In this case, the estimated segment support of v — *just* from the k-th segment, i.e. $estsup_k(v)$ — may give a tighter bound. Thus, the second component of $maxMissed(v)$ can be computed as:

$$maxMissed_2(v) \quad = \quad min\{j,\ estsup_k(v)\} \tag{6}$$

Hence, Carma(w/ SSM) can combine all of equations (3), (4), (5) and (6) to give a better bound for $maxMissed(v)$. Note that this is a bound obtained *at the point of insertion* of v, i.e., after the transaction t_{q+j} has been read.

However, at a later point in time, when the *entire k-th segment has been read*, a better bound may be possible. This is because in equation (6) above, $estsup_k(v)$ is used to estimate the support of v from the first j transactions in the k-th segment, when in fact the same bound applies to the *entire* segment, implying that we should be able to do better. Let $count_k(v)$ denote the *actual* support of v in the k-th segment *after v was inserted*. Then by the definition of the SSM, it must be the case that:

$$maxMissed_2(v) + count_k(v) \leq estsup_k(v) \tag{7}$$

Hence, while equation (6) is appropriate for initializing the term $maxMissed_2(v)$ *at* the point of insertion, the following equation can be used to *update*, and possibly *tighten*, $maxMissed_2(v)$ *after* the entire k-th segment has been processed:

$$maxMissed_2(v) = min\{j, (estsup_k(v) - count_k(v))\} \tag{8}$$

The value of $maxMissed(v)$ can be tightened accordingly.

4.2 Creating the $future_k(v)$ Bound

Recall that there are two weaknesses associated with the pruning based on the condition $maxSupport(v) < \sigma_i$, where the term $maxSupport(v) = (maxMissed(v) + count(v))/i$. So far we have addressed the first problem by tightening the $maxMissed(v)$ bound. Next we turn our attention to the second problem of making the (strong) assumption of uniform distribution.

Suppose that the transaction scan is at the point when the k-th segment has just been processed. Given an itemset v that has been inserted and is being processed, let $future_k(v)$ denote an upper bound on the support of v from all the future/remaining segments, i.e., $(k{+}1)$-th, $\ldots$, m-th segments. With the SSM, this is defined as:

$$future_k(v) = \sum_{u=k+1}^{m} estsup_u(v) \tag{9}$$

By putting all the pieces together, the old pruning condition, which is based on the value of $maxSupport(v) = (maxMissed(v) + count(v))/i$, can now be replaced by a new condition $(maxMissed(v) + count(v) + future_k(v))/n$, where n is the total number of transactions, at the point when the k-th segment has just been processed. This new condition can be used in Step (9) in Figure 1. Note that both $maxMissed(v)$ and $future_k(v)$ are upper bounds, and $count(v)$ is the actual count up to the point of pruning. Thus, unlike the old condition, the new condition guarantees that a pruned itemset will never need to be re-inserted, unless the support threshold is reduced.

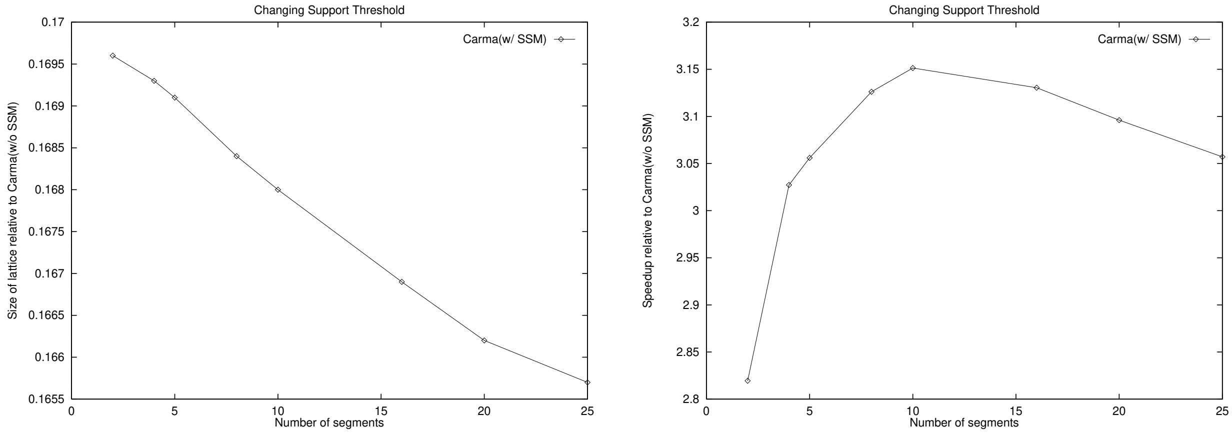
4.3 Skipping Confirmed Infrequent Itemsets

In addition to helping to tighten the $maxSupport(v)$ and $maxMissed(v)$ bounds for pruning, the SSM can enhance Carma in better exploiting the frequency constraint. It is well-known that the frequency constraint $support(S) \geq minsup$ is anti-monotone. If any itemset S is frequent, then all its subsets $S' \subseteq S$ are also frequent. Contrarily, any itemset that already has a support below the user-defined threshold $minsup$ should be tossed away, because adding more items to the itemset will never increase its support count, and thus will never make it satisfy the frequency constraint. Currently, Carma exploits anti-monotonicity by making sure that an itemset v is added to the support lattice, only if all its subsets w are already in the lattice.

With the SSM, additional pruning based on anti-monotonicity can be effected as follows. After transaction t_i has just been read, Carma(w/ SSM) only needs to consider those itemsets that do not include any *confirmed infrequent* single item as candidates. For example, suppose $t_i = \{a, b, ..., k, l\}$, and knowing from the SSM of these 12 items that a, b and c have a support below σ_i. Let t_i^{SSM} denote the subset of t_i not containing any confirmed infrequent single item. Then, for all subsequent computation regarding t_i, we only need to consider $t_i^{SSM} = t_i - \{a, b, c\}$, because no itemset containing a, b or c can be frequent. Specifically, for subsets of size j of t_i, we need to process only $\binom{9}{j}$, as opposed to $\binom{12}{j}$, itemsets. Accumulating for all $j \geq 2$, this simple optimization step can bring about considerable saving.

5. EXPERIMENTAL RESULTS

The experimental results cited below are based on a transaction database TDB of 100k records, and a domain of 10k items. The TDB was generated by the program developed at IBM Almaden Research Center [3]. The average transaction size is 10 items, and the average cardinality of a frequent set is 4. Unless otherwise specified, we used a support threshold of 0.1%. All experiments were run in a time-



(a) Relative Size of Lattice

(b) Relative Speedup

Figure 3: SSM-based Pruning: Changing Support Threshold

sharing environment in a 600 MHz machine. The speedup shown is with respect to total CPU and I/O time.

In this experiment, we compared the results for two algorithms (implemented in C): Carma(w/o SSM) and Carma(w/ SSM). The former does not include the SSM, whereas the latter does. The difference is to highlight the effectiveness of SSM-based pruning.

5.1 SSM-based Pruning: Constant and Changing Support Threshold

We evaluated how the number of segments in the SSM can benefit Carma(w/ SSM), with a query consisting of the frequency constraint $support(S) \geq minsup$. We conducted two sets of experiments: (i) one experiment with a fixed support threshold of $minsup = 0.1\%$, and (ii) another with support threshold $minsup$ varied from 0.075% to 0.125% then to 0.1%. These values were chosen to correspond to a similar set of experiments shown for Carma in Hidber's work [10]. The results of these two sets of experiments turned out to be almost the same. For lack of space, we only show the results based on changing support threshold.

The x-axis in Figure 3 shows the number of segments varying from 2 to 25. The y-axis of Figure 3(a) shows the size of the lattice computed by Carma(w/ SSM) relative to that by Carma(w/o SSM). The size of the lattice corresponds to the number of itemsets that were counted. The smaller the size, the more effective the pruning was. As expected, the larger the number of segments in the SSM, the larger the number of itemsets that were pruned. For instance, with 10 segments, the number of itemsets counted was about 1/6 of that required without the SSM.

The y-axis of Figure 3(b) gives the speedup of Carma(w/ SSM) relative to Carma(w/o SSM) in terms of total runtime. With increasing number of segments, while the relative size of the lattice decreases monotonically in Figure 3(a), the relative speedup shows a peak when the number of segments is 10 in Figure 3(b). The peak occurs when the reduction in the lattice size is no longer significant enough to offset the cost of processing an extra segment. The result shows that while the SSM is a light-weight structure (e.g., 10 segments requiring 100 000 integers for 10k items, for a total space of 0.2MB using 2 bytes per integer), the pruning effect is spectacular. In absolute terms, Carma(w/o SSM) took about 10 seconds total time; whereas Carma(w/ SSM) took less than 3 seconds with 10 segments. This is very encouraging because this shows that we are making significant progress towards the eventual goal of providing real time response/completion of query evaluation.

5.2 SSM-based Pruning: "Seasonal" Transaction Database

In Section 4.2, we mentioned that the SSM is well suited to handle transaction databases that do not follow the uniform distribution assumption, and there are many databases of this kind. An example is the supermarket database consisting of "seasonal" transactions. By using the IBM Almaden program, we generated a transaction database in which the effect of the "seasonal" nature has been simulated. The following table shows the speedup of Carma(w/ SSM) relative to that of Carma(w/o SSM).

Number of Segments	Relative Speedup
2	5.81
4	5.96
5	5.99
8	6.04
10	6.06
16	6.03
20	6.00
25	5.96

For the "seasonal" data, the average speedup is around 6 times, as opposed to around 3 times as shown in Figure 3(b). This shows that, mainly via the $future_k(v)$ bound, the SSM delivers additional benefit when the transaction database is not uniformly distributed and is "seasonal" in nature.

6. DISCUSSIONS

So far we have studied how the SSM help to improve the efficiency of Carma. But it is important to note that benefits of the SSM are not confined to Carma, it can be applied to other frequent-set mining algorithms. In this section, we show two examples: (i) how to use the SSM in Apriori, and (ii) how to use the SSM in CAP.

6.1 Using the SSM in Apriori

Recall from Section 3 that the SSM consists of the support count for all singleton itemsets in each segment in the transaction database. So, with the SSM, the Apriori algorithm does not need to generate the candidate 1-itemsets and test for support. It can find the frequent 1-itemsets by selecting those itemsets whose support count $\sum_{k=1}^{m} support_k(v)$ is at least the user-defined threshold $minsup$.

Moreover, such an optimization can be extended to higher levels. With the SSM, the Apriori algorithm can reduce the number of itemsets to be counted by discarding those itemsets $v \in C_k$ with $estsup(v) = \sum_{j=1}^{m} estsup_j(v)$ less than the user-defined threshold, for $k \geq 2$. These itemsets are confirmed infrequent. To illustrate, let us return to the example introduced in Section 3. Suppose the user-defined threshold $minsup$ is set to 42. Without the SSM, the Apriori algorithm needs to count the support for all 3 itemsets $\{a,b\}, \{a,c\}$ and $\{b,c\}$; but, with the SSM, the Apriori algorithm only needs to consider the itemset $\{b,c\}$. Similarly, the Apriori algorithm, with the SSM, can further reduce the number of itemsets to be counted in higher levels.

6.2 Using the SSM in CAP

For the frequent-set mining algorithms that we have discussed so far, they cannot handle constraints other than the frequency constraint $support(S) \geq minsup$. As a result, the user may need to wait for hours for numerous computed rules, out of which only a small fraction might be interested to the user. Thus the user incurs a high computational cost that is disproportionate to what the user wants and gets. To allow the user to express his focus, Ng et al. [15; 13] proposed: (i) a constrained frequent mining framework within which the user can use a rich set of constraints to guide the mining to find only those rules satisfying the constraints, and (ii) a mining algorithm, called CAP, which exploits the constraints to ensure that the computation effort is proportional to the selectivity of the constraints. These constraints include aggregation constraints, where the allowable aggregation operations are the basic ones supported by SQL, i.e. $min(), max(), sum(), count()$ and $avg()$.

The CAP algorithm exploits the constraints, and pushes them as deep "inside" the computation as possible. For instance, for a constraint C that is anti-monotone, if S does not satisfy C, then any superset S' of S is removed from counting because S' does not satisfy C.

With the SSM, the CAP algorithm can further exploit the constraints so as to further improve the performance. Recall from Section 6.1, the anti-monotonicity nature of the frequency constraint $support(S) \geq minsup$ can be exploited to speed up the mining process. The SSM we have seen so far can be easily extended to help exploit anti-monotone constraints other than the frequency constraint. Hence, in addition to the above optimizations for Apriori, the SSM has extra advantages to CAP as follows. Let us use $sum(S.Price) \leq 10$ as an example. With the SSM, all CAP needs to do is to add one column to the SSM, recording the $Price$ value of each singleton itemset. In other words, the CAP algorithm with SSM only needs to consider those itemsets that do not contain any *confirmed unsatisfiable* single item as candidates. For example, suppose that among the items in the domain, only a, d and e have $Price$ more than 10. Then, for all subsequent computations, we can discard any itemset containing a, d or e, because no itemset containing a, d or e can satisfy the anti-monotone constraint.

When combining with the optimization for the frequency constraint, the SSM brings further saving. Using the above example, and suppose that only a, b and c have a support below $minsup$. Then, for all subsequent computation, we can discard any itemset containing a, b, c, d or e, because no itemset containing these confirmed infrequent and/or confirmed unsatisfiable single items can satisfy both anti-monotone constraints.

7. CONCLUSIONS

A key contribution of this paper is to optimize the performance of mining algorithms. To this end, we have proposed and studied the novel structure of SSM. While very light-weight, it helps to tighten the $maxSupport(v)$ and $maxMissed(v)$ bounds for pruning, and to better exploit the frequency/support constraint as well as other anti-monotone constraints. Experimental results reported here convincingly show the benefits of SSM. With a very small price to pay (e.g., 0.2MB of space for 10 000 items), the SSM can prune a much larger number of itemsets. Consequently, as an enhancement to Carma, the SSM can bring about a speedup that is a few times better than without using the SSM.

Furthermore, even though we present the SSM as a feature to enhance Carma, we have shown that the SSM can be applied to the conventional mining of association rules using the Apriori algorithm (and hence, to many related data mining tasks using the Apriori algorithmic framework). The same can be said about constrained mining, as is done in CAP.

In ongoing and future work, we are interested in exploring improvements to the SSM. For example, we are interested to investigate the benefits of storing the actual segment supports of higher cardinalities in the SSM. Along this direction, an interesting question to explore is that given a fixed amount of space for the SSM, which itemsets (of varying cardinalities) should be stored in it.

8. REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. 1993 SIGMOD*, pp. 207–216.

[2] R. Agrawal and J.C. Shafer. Parallel mining of association rules. *IEEE TKDE*, **8**(6), pp. 962–969, Dec. 1996.

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

[4] R.J. Bayardo. Efficiently mining long patterns from databases. In *Proc. 1998 SIGMOD*, pp. 85–93.

[5] S. Brin, R. Motwani, and C. Silverstein. Beyond market basket: Generalizing association rules to correlations. In *Proc. 1997 SIGMOD*, pp. 265–276.

[6] D.W. Cheung, J. Han, V.T. Ng, and C.Y. Wong. Maintenance of discovered association rules in large databases: An incremental updating technique. In *Proc. 1996 ICDE*, pp. 106–114.

[7] T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Data mining using two-dimensional optimized association rules: Scheme, algorithms, and visualization. In *Proc. 1996 SIGMOD*, pp. 13–23.

[8] E.-H. Han, G. Karypis, and V. Kumar. Scalable parallel data mining for association rules. In *Proc. 1997 SIGMOD*, pp. 277–288.

[9] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proc. 1995 VLDB*, pp. 420–431.

[10] C. Hidber. Online association rule mining. In *Proc. 1999 SIGMOD*, pp. 145–156.

[11] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A.I. Verkamo. Finding interesting rules from large sets of discovered association rules. In *Proc. 1994 CIKM*, pp. 401–408.

[12] F. Korn, A. Labrinidis, Y. Kotidis, and C. Faloutsos. Ratio rules: A new paradigm for fast, quantifiable data mining. In *Proc. 1998 VLDB*, pp. 582–593.

[13] L.V.S. Lakshmanan, R. Ng, J. Han, and A. Pang. Optimization of constrained frequent set queries with 2-variable constraints. In *Proc. 1999 SIGMOD*, pp. 157–168.

[14] R.J. Miller and Y. Yang. Association rules over interval data. In *Proc. 1997 SIGMOD*, pp. 452–461.

[15] R.T. Ng, L.V.S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proc. 1998 SIGMOD*, pp. 13–24.

[16] J.S. Park, M.-S. Chen, and P.S. Yu. Efficient parallel mining for association rules. In *Proc. 1995 CIKM*, pp. 31–36.

[17] J.S. Park, M.-S. Chen, and P.S. Yu. Using a hash-based method with transaction trimming for mining association rules. *IEEE TKDE*, **9**(5), pp. 813–825, Sept./Oct. 1997.

[18] S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with relational database systems: Alternatives and implications. In *Proc. 1998 SIGMOD*, pp. 343–354.

[19] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *Proc. 1995 VLDB*, pp. 432–443.

[20] C. Silverstein, S. Brin, R. Motwani, and J. Ullman. Scalable techniques for mining causal structures. In *Proc. 1998 VLDB*, pp. 594–605.

[21] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *Proc. 1996 SIGMOD*, pp. 1–12.

[22] H. Toivonen. Sampling large databases for association rules. In *Proc. 1996 VLDB*, pp. 134–145.

[23] D. Tsur, J.D. Ullman, S. Abiteboul, C. Clifton, R. Motwani, S. Nestorov, and A. Rosenthal. Query flocks: A generalization of association-rule mining. In *Proc. 1998 SIGMOD*, pp. 1–12.