Peak-Jumping Frequent Itemset Mining Algorithms

Nele Dexters[1], Paul W. Purdom[2], and Dirk Van Gucht[2]

[1] Departement Wiskunde-Informatica, Universiteit Antwerpen, Belgium,
nele.dexters@ua.ac.be,
[2] Computer Science Department, Indiana University, USA,
pwp@cs.indiana.edu, vgucht@cs.indiana.edu

Abstract. We analyze algorithms that, under the right circumstances, permit efficient mining for frequent itemsets in data with tall peaks (large frequent itemsets). We develop a family of level-by-level peak-jumping algorithms, and study them using a simple probability model. The analysis clarifies why the jumping idea sometimes works well, and which properties the data needs to have for this to be the case. The link with Max-Miner arises in a natural way and the analysis makes clear the role and importance of each major idea used in this algorithm.

1 Introduction

The frequent itemset mining (FIM) problem [1, 2] is to determine combinations of items that occur together frequently: frequent itemsets (FIS). In a store with n items visited by s shoppers, the question is which of the 2^n possible itemsets are bought together by at least k shoppers, with k the threshold.

The *support* of itemset I, $\mathit{supp}(I)$, is the number of shoppers that buy all items from I. Itemset I is *frequent* if $\mathit{supp}(I) \geq k$, and *infrequent* otherwise. If $J \subseteq I$, then $\mathit{supp}(J) \geq \mathit{supp}(I)$. This is the *monotonicity property* of the support function. If I contains m items and (1) if I is frequent, then all of its 2^m subsets are also frequent, and, (2) if I is infrequent, then all of its 2^{n-m} supersets are also infrequent. These principles are at the core of FIM algorithms because of their potential to reduce the search space. Max-Miner [3] is the canonical example of an algorithm that uses pruning principle (1) and Apriori [2] of principle (2).

Datasets with large FIS are called *peaky*. Algorithms designed to run on peaky data, such as Max-Miner, MAFIA [4], certain versions of Eclat [10] and FP-Growth [8], and GenMax [7] usually avoid outputting every FIS. Instead, they can concentrate only on the maximal FIS. The basic idea for dealing efficiently with large FIS is trying to jump up the peaks. Early in the processing, it is important to determine the frequency of selected large itemsets. When a large FIS is found, the challenge is to quickly avoid processing its subsets.

This paper uses simple probabilistic analysis to show that if you want (1) to do FIM, (2) to handle peaky data efficiently, and (3) to use only simple ideas, you are more or less forced into an algorithm that is very close to Max-Miner [3]. An alternate view is that if any key idea is omitted from Max-Miner, then its performance will be much poorer for some simple real-world datasets.

2 Two Shopper, Two Item Types Data Model

The two shopper, two item types data model is based on the following notation:

- s_i shoppers of type i $(i = 1, 2)$,
- n_j items of type j $(j = 1, 2)$, and
- probability p_{ij} that a shopper of type i buys an item of type j.

By convention, p_{11} is the largest of the probabilities.

This shopping model brings out the most important aspects of peak-jumping algorithm performance. The probability that a shopper of type i $(i = 1, 2)$ buys m_1 different items of type 1 and m_2 of type 2, regardless of what else is bought:

$$P_i(m_1, m_2) = p_{i1}^{m_1} p_{i2}^{m_2}. \tag{1}$$

The expected support of an itemset that contains m_1 particular items of the first kind and m_2 of the second:

$$s_1 P_1(m_1, m_2) + s_2 P_2(m_1, m_2), \tag{2}$$

$$s_1 p_{11}^{m_1} p_{12}^{m_2} + s_2 p_{21}^{m_1} p_{22}^{m_2}. \tag{3}$$

3 The Perfect Jump Algorithm

3.1 Perfect Jumps

Suppose an algorithm jumps and finds that some itemsets on a high level are frequent. The question is how to make good use of this information. In principle, monotonicity can be used. If we jump from level l to level x $(x > l)$ and find some FIS on level x, then the only candidates that need to be considered on level $l+1$ are formed from pairs of sets on level l that are not subsets of the FIS on level x. Otherwise, the candidate is frequent based on monotonicity. Unfortunately, there is no fast general algorithm to carry out this subsumption test.

This paper focusses on the idea: a jump to level x is successful if and only if *every* candidate on level x is frequent. We call this a *perfect jump*. Now, every candidate between levels $l+1$ and x is frequent. By only allowing perfect jumps, the performance of the algorithm is limited, but the time for processing levels l and x is proportional to the sum of the number of candidates on the two levels.

Suppose that on level l the FIS contain a_l distinct items. If all the FIS are coming from a single peak, the itemset causing the peak contains a_l items. Furthermore, the number of FIS on level l is $\binom{a_l}{l}$. When the number of FIS is not given by this formula, there is no possibility that a perfect jump will occur.

Suppose, in a perfect jump case, we jump to level x. There are $\binom{a_l}{x}$ candidates on level x. They are formed by taking each combination of x items from the a_l items that occur in the FIS on level l. If any early jump is successful, then the jumping algorithm can be much faster than an algorithm that does not try to jump. In particular, the jumping algorithm does not have to explicitly determine the frequency status of the $\sum_{j=l+1}^{x-1} \binom{a_l}{j}$ itemsets that occur strictly between levels l and x. Clearly, the biggest savings come from doing a jump from a level l, small compared to a_l, to a level x near a_l. Therefore, we will tacitly assume that jumps occur from small levels to high levels.

3.2 The Algorithm

Based on the concept of perfect jumps, we present the *Perfect Jump Algorithm.*

Step 1. [Start] Set $l = 0$. If the number of shoppers is greater than k then set the empty set to frequent, otherwise set the empty set to infrequent.
Step 2. [Check if done] If there are no FIS on level l then stop. Otherwise set $l = l+1$.
Step 3. [Process level l] Determine the frequency of the itemsets of size l that can be built from the FIS from level $l-1$. (See [2] for details on how to do this.)
Step 4. [Try to jump] Set a_l to the number of items that appear in the FIS on the current level. If the number of FIS on this level is strictly smaller than $\binom{a_l}{l}$ or $l \geq a_l/2$ then go to Step 2.
Step 5. [Try jumping] Set $x = a_l - l$. For every itemset that can be formed by choosing x of the a_l items, determine whether or not the item set is frequent. If every such itemset is frequent, then set $l = x + 1$ and go to Step 3.
Step 6. [Continue] Go to Step 2.

When the conditions for attempting to jump are met, the Perfect Jump (PJ) Algorithm tries jumping to level $a_l - l$ because the number of candidate itemsets to be tested at that level is the same as the number of FIS on level l. This balances the work attempting jumps with the other work, thereby ensuring that the algorithm is never slower than Apriori by more than a factor of two. In favorable cases, i.e., when a large jump occurs, the PJ Algorithm is much faster.

4 Analysis of the Perfect Jump Algorithm

The shopping model from Section 2 generates a distribution of datasets. Every dataset with $s_1 + s_2$ shoppers and $n_1 + n_2$ items will be generated with some probability, but some of these sets have much higher probability than others. We now consider a particular dataset in which the actual supports of the itemsets equal the expected supports from the probability model.

To have a perfect jump, we must have a tall peak. Since p_{11} is the largest probability, eqs. (1, 2) imply that there is no peak unless p_{11} is close to one.

We begin the analysis of the performance of the PJ Algorithm for the case where the peak at level $x = m_1 - l$ is formed only from m_1 items of type 1.

$$s_1 P_1(m_1 - l, 0) + s_2 P_2(m_1 - l, 0) \geq k, \tag{4}$$

$$s_1 p_{11}^{m_1 - l} + s_2 p_{21}^{m_1 - l} \geq k. \tag{5}$$

To meet the condition for a perfect jump the FIS on level l must be those itemsets that contain l items of type 1 and no items of type 2. Obeying eq. (5) already ensures that every itemset with l items of type 1 is frequent. To ensure that no other itemsets are frequent, we need that for every r $(1 \leq r \leq l)$,

$$s_1 P_1(l - r, r) + s_2 P_2(l - r, r) < k, \tag{6}$$

$$s_1 p_{11}^{l-r} p_{12}^{r} + s_2 p_{21}^{l-r} p_{22}^{r} < k. \tag{7}$$

The first term on the left side of eq. (7) is biggest at $r = 1$, since $p_{11} > p_{12}$. The second term is biggest at $r = 1$ when $p_{21} \geq p_{22}$ and at $r = l$ when $p_{21} \leq p_{22}$.

We now start a case analysis of the condition for a perfect jump to occur on level l. The cases are selected for the most interesting parameter settings.

For the first case, assume that $p_{21} = 0$ and $p_{22} = 0$. To have a perfect jump we need to have

$$p_{11}^{m_1 - l} \geq \frac{k}{s_1} \tag{8}$$

and

$$p_{12} < \frac{k}{s_1 p_{11}^{l-1}}. \tag{9}$$

If we write $p_{11} = 1 - \epsilon$, then logarithm of eq. (8) says

$$(m_1 - l)\,(-\epsilon + \text{higher order terms}) \geq \ln(k/s_1), \tag{10}$$

$$\epsilon \leq \frac{-\ln(k/s_1)}{m_1 - l} \qquad \text{(neglecting higher order term).} \tag{11}$$

Essentially, to have a perfect jump from a low level, p_{11} must be close to 1 and p_{12} must be less than k/s_1. For a particular dataset eq. (9) says that threshold k has to be set large enough to filter out the noise from the items that are not part of the peak, and eq. (8) determines whether there is any value of k that works. The value of l plays only a small role in what is happening[3]. Thus, if this noise resulting from shoppers buying the peak items and occasional not-peak items causes trouble on level 1, it probably will cause trouble on the higher levels too.

In a second case, we assume that p_{21} and p_{22} are not both 0 but neither is close to 1. In this case, we may not be able to make perfect jumps for small values of l, but since in this case the second term in eq. (7) decreases rapidly with l and the second term in eq. (5) only helps, we will be able to make a perfect jump with a moderate value of l.

For a third case, we assume that p_{21} is large. When this is the case, both types of shoppers contribute to the peak. We need both p_{12} and p_{22} to be small (close to 0) to have a perfect jump to a peak based on items of just type 1.

A next case is considered, when three or four of the probabilities are large. We now may have a peak based on all the items of both types. When s_1 is small, s_2 is large, and p_{22} is large, we may have a peak based just on items of type 2.

In a last case, when p_{11} and p_{22} are near 1 but p_{12} and p_{21} are near 0, the model is generating data with two peaks. The PJ Algorithm needs an additional idea before it can do well on such data (See Section 6).

To summarize this section, the PJ Algorithm will do well on data with a single peak so long as the subset of shoppers that are buying most of the peak items are unlikely to buy any non-peak items. This is the case even in the presence of other shoppers, so long as their buying patterns do not lead to other peaks.

[3] If p_{11} is not extremely close to 1 (eq. (11)), we don't have a peak. If p_{11} is quite close to 1, then p_{11}^{l-1} is close to 1 for small l.

5 The Perfect Jump Algorithm with Lower Bounds

If we know the frequency of all FIS on levels $l-1$ and l, we can obtain a lower bound on the frequency of the candidates on higher levels using [3]:

$$Supp(I \uplus J) \geq \sum_{j \in J} Supp(I \uplus \{j\}) - (|J|-1)Supp(I), \tag{12}$$

in which $\uplus$ denotes a union on disjoint sets. For us, I is a set on level $l-1$ and J is the set of items other than those in I that make up the peak.

We can modify the PJ Algorithm so that Step 5 replaces the actual count on level $x = m_1 - l$ with the bound given by eq. (12) (where $|I| = l-1$ and $|J| = m_1 - l - |I|$). To analyze the conditions for a successful perfect jump using this modified algorithm with data with a peak based only on items of type 1, we still need eq. (9), but we need to replace eqs. (4, 5) with

$$\begin{aligned} &s_1[(m_1-2l+1)P_1(l,0) - (m_1-2l)P_1(l-1,0)] \\ &\qquad +s_2[(m_1-2l+1)P_2(l,0) - (m_1-2l)P_2(l-1,0)] \geq k, \qquad (13) \\ &s_1 p_{11}^{l-1}[(m_1-2l+1)p_{11} - (m_1-2l)] \\ &\qquad +s_2 p_{21}^{l-1}[(m_1-2l+1)p_{21} - (m_1-2l)] \geq k. \qquad (14) \end{aligned}$$

For the case where $p_{21} = 0$, if we write $p_{11} = 1 - \epsilon$, and carry out the calculations to first order in ϵ, we obtain

$$(1-\epsilon)^{l-1}[1-(m_1-2l+1)\epsilon] \geq \frac{k}{s_1}, \text{ leading to } \epsilon \leq \frac{1-k/s_1}{m_1-l}. \tag{15}$$

Suppose we write $k/s_1 = 1 - \delta$. For datasets associated with the probability model, when k is almost as large as s_1 (δ near 0), the PJ Algorithm needs p_{11} equally close to 1 whether it uses estimates or exact counts in order for the jump to be successful. When k is less than s_1 by a few powers of ten, the version of the algorithm that uses exact counts will succeed in making perfect jumps with values of p_{11} that are much further from 1.

6 The Perfect Jump Algorithm with Connected Components

Items with intermediate probability interfere with the performance of the PJ Algorithm. When this interference is coming from a second type of shoppers, it can often be eliminated by breaking the data into disjoint components.

Define FIS I and J as *connected* when

$$|I| = |J| = |I \cap J| + 1. \tag{16}$$

The resulting connectedness relation is symmetric, but not transitive[4]. The reflexive transitive closure is an equivalence relation. Therefore, we can divide the

[4] This relation was previously used in some versions of the Eclat Algorithm [10].

FIS on level l into equivalence classes by putting into the same class any two frequent itemsets that are connected by a chain. Itemsets from two different equivalence classes never interact in FIM algorithms. Indeed, if I and J are FIS at level l and in different equivalence classes, then itemset $I \cup J$ can not be frequent, because if it were, there would be a chain connecting I and J.

Thus, the PJ Algorithm remains correct if we partition the FIS on level l into equivalence classes and process each equivalence class independently. On level 1, all FIS are in the same class, but on higher levels there can be many classes.

For our random model, it is interesting to consider the conditions where the itemsets made up of items of type 1 are in a different equivalence class than those of type 2. To have two classes on level l, all those sets made up of $l - t$ items of type 1 and t items of type 2 with $1 \leq t \leq l - 1$ must be infrequent. The expected number of such sets is

$$s_1 P_1(l - t, t) + s_2 P_2(l - t, t). \tag{17}$$

When this number is significantly below k the typical dataset from the model will have two equivalence classes. Thus the condition for each peak to be in its own equivalence class is the same eq. (7) except that eq. (7) permits $t = l$ while eq. (17) does not allow that case. The $t = l$ case is associated with a peak that will be in a separate component.

We see that by combining the idea of perfect jumps with partitions, we obtain a simple algorithm that is efficient at finding the maximal FIS in some interesting datasets with several large peaks.

7 The Max-Miner Algorithm

The PJ Algorithm with Connected Components is a fine algorithm except for datasets with several items of intermediate probability. One way to overcome this problem is to partition the search space of itemsets so that many of the partitions don't have itemsets with items of intermediate probability. This is the first key idea of the Max-Miner Algorithm [3].

Max-Miner is a level-wise FIM algorithm that works like Apriori, except that it also has peak-jumping capabilities. Throughout its running, it maintains a set $\mathcal{M}$ which at the end will contain all the maximal FIS.

At the start, Max-Miner picks a total ordering σ on the set of all items. It then begins its search for FIS. Assume that the algorithm has discovered the set $\mathcal{F}_{l-1}$ of all FIS at level $l - 1$. From these sets, it generates the set $\mathcal{C}_l$ of all candidates at level l. Based on σ and $\mathcal{F}_{l-1}$, it then partitions $\mathcal{C}_l$ as follows. The ordering σ induces a lexicographic order on the itemsets in $\mathcal{F}_{l-1}$, say $I_1 < I_2 < I_3 < \cdots$. For I_1, the algorithm finds all candidates that contain I_1 and puts them in a cell, called $\mathit{cell}(I_1)$. Then, from the *remaining* candidates, it finds all those that contain I_2 and puts them in $\mathit{cell}(I_2)$. It then determines $\mathit{cell}(I_3)$ etc. Notice that some of these cells may be empty. After this partitioning phase, the algorithm processes each non-empty cell as follows. For each itemset in cell $\mathit{cell}(I)$, the algorithm determines whether or not it is frequent. If it finds FIS, it constructs itemset J_I which consists of all the items that occur in *cell*(I), except those that are in I. Call *peak*(I) the set $I \cup J_I$. The algorithm then enters its peak-jumping phase. For each *peak*(I) it determines whether or not it is frequent (either by explicit counting or by using the lower bound formula). If it is frequent, *peak*(I) is added to $\mathcal{M}$, otherwise the FIS in *cell*(I) are added to $\mathcal{M}$. Then the algorithm moves from level l to level $l+1$.

When comparing Max-Miner with the PJ Algorithm, think of Max-Miner as a parameterized version of the PJ Algorithm (relative to I) which attempts only jumps from level 1 to the highest possible level (as determined by J_I).

In practice, Max-Miner uses a more subtle strategy to update $\mathcal{M}$. When it adds *peak*(I), it removes from $\mathcal{M}$ all the itemsets that are subsumed by it. And, when it considers adding the FIS in *cell*(I), it only adds those that are not subsumed by an existing itemset in $\mathcal{M}$. Clearly, under this update strategy, there is the guarantee that at the end, $\mathcal{M}$ consist exactly of all maximal FIS. Clearly this can be costly because each of the subsumption tests has to be done explicitly, and no general algorithm exists that can do this efficiently. Regardless of which update strategy is used, in the cases where *peak*(I) is frequent and J_I is large, Max-Miner can be expected to gain substantial performance improvements (see the experimental results in [3]).

If the initial ordering σ picked by Max-Miner is a randomly selected ordering, then the FIS at level $l-1$, i.e. those in $\mathcal{F}_{l-1}$, will also occur in some random order. When the algorithm then enters its partitioning phase of the candidates at level l, i.e. $\mathcal{C}_l$, it can be expected that for I in $\mathcal{F}_{l-1}$, the set J_I will have a mixture of low- to high-probability items. Even if J_I is large, the analysis in Section 4 strongly suggests that *peak*$(I) = I \cup J_I$ will be infrequent and the algorithm will not be able to gain much from its peak-jumping abilities. Overcoming this problem is the second key idea of Max-Miner. Rather than picking a random ordering, it picks an ordering based on the supports of the items. Specifically, the ordering is determined by arranging the items in *increasing* order of their supports. This ordering is much more likely to decrease the generation of many J_I's which have a mixture of low- to high probability items. Consequently, the algorithm considerably improves its opportunity to obtain significant performance improvement on account of peak-jumping.

Max-Miner searches for FIS in level-wise (breadth-first) order. If one changes to depth-first order, it is necessary to give up the full Apriori candidacy test. Doing so leads to Eclat-like versions of Max-Miner [4, 8, 10, 7]. However, with respect to the performance gain due to peak-jumping, these Eclat-like algorithms get essentially the same benefits as Max-Miner.

8 Discussion and Conclusion

When considering algorithms to process peaky data, it is necessary to consider the variation of probability of various items and the correlations in the data.

If the data has some low frequency items, some high frequency items, no intermediate frequency items, and no important correlations, then the data may have one peak. The P J Algorithm has good performance in this case.

If the data is the same as above but with important correlations, then there may be several peaks. The PJ Algorithm needs to be augmented with a division into independent components before it can handle such data efficiently.

These ideas still do not lead to an efficient algorithm if the dataset has several items with intermediate probability. Many datasets have the property that the occurrence of items is close enough to independent that the relative probability of items on one level is similar to that on other levels. In such cases, one can do jumps relative to an itemset that is missing the items of intermediate probability. This is the essential reason why Max-Miner types of algorithms are efficient.

This line of reasoning makes it clear that ordering items from least-frequent to most-frequent is critical to the success of Max-Miner. If for example, Max-Miner had processed items in random order, it is unlikely that it would produce an itemset that had all of the highly frequent but none of the moderately frequent items. Likewise, it will not do well on datasets where the correlation structure makes the frequency of items on one level greatly different from that on other levels. Similar conclusions can be drawn for other maximal FIM algorithms.

References

1. Rakesh Agrawal, Tomasz Imielinski, and Arun Swami: Mining Association Rules between Sets of Items in Large Databases. SIGMOD Record, ACM Press (1993) pp 207–216.
2. Rakesh Agrawal and Ramakrishnan Srikant: Fast Algorithms for Mining Association Rules. in Proc. of the 1994 Very Large Data Bases Conference (1994), pp 487–499.
3. Roberto J. Bayardo: Efficiently Mining Long Patterns from Databases. in Proc. of the 1998 ACM SIDMOD Int. Conf. on Management of Data, (1998) pp 85–93.
4. Douglas Burdick, Manuel Calimlim, and Johannes Gehrke: MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases. Proc. of the 17th Int. Conf. on Data Engineering (2001) pp 443–452.
5. Nele Dexters, Paul Purdom, Dirk Van Gucht: Analysis of Candidate-Based Frequent itemset Algorithms. Proc. ACM Symposium on Applied Computing, Data Mining Track, (2006) Dijon.
6. Bart Goethals: Frequent Set Mining. The Data Mining and Knowledge Discovery Handbook, Oded Maimon and Lior Rokach (Eds.) (2005), pp 277-397.
7. Karam Gouda and Mohammed J. Zaki. GenMax: An Efficient Algorithm for Mining Maximal Frequent Itemsets. Data Mining and Knowledge Discovery, 11, (2005) pp 1–20.
8. Jiawei Han, Jian Pei, Yiwen Yin: Mining Frequent Patterns without Candidate Generation. Proc. of the 2000 ACM SIGMOD Int. Conf. Management of Data, Dallas, TX (2000) pp 1–12.
9. Paul W. Purdom, Dirk Van Gucht, and Dennis P. Groth: Average Case Performance of the Apriori Algorithm. SIAM J. Computing, **33**(5),(2004) pp 1223–1260.
10. Mohammed J. Zaki: Scalable Algorithms for Association Mining. IEEE Transactions on Knowledge and Data Engineering, **12**(3), (2000) pp 372–390.