A tight upper bound on the number of candidate patterns

Floris Geerts Bart Goethals Jan Van den Bussche

Limburg University, Belgium

Abstract

In the context of mining for frequent patterns using the standard levelwise algorithm, the following question arises: given the current level and the current set of frequent patterns, what is the maximal number of candidate patterns that can be generated on the next level? We answer this question by providing a tight upper bound, derived from a combinatorial result from the sixties by Kruskal and Katona. Our result is useful to reduce the number of database scans.

1 Introduction

The frequent pattern mining problem [1] is by now well known. We are given an ordered set of items $\mathcal{I}$ and a database $\mathcal{D}$ of subsets of $\mathcal{I}$ called transactions. A *pattern* is some set of items; its *support* in $\mathcal{D}$ is defined as the number of transactions in $\mathcal{D}$ that contain the pattern; and a pattern is called *frequent* in $\mathcal{D}$ if its support exceeds a given minimal support threshold. The goal is now to find all frequent patterns in $\mathcal{D}$.

The standard Apriori algorithm [2] for solving this problem is based on the observation that all subsets of a frequent pattern are also frequent. A pattern is thus potentially frequent, also called a *candidate* pattern, if its support is yet unknown, but all of its subsets are already known to be frequent. In every step of the algorithm, all candidate patterns are generated and their supports are then counted by performing a complete scan of the transaction database. This is repeated until no new candidate patterns can be generated.

The search space of this problem, all subsets of $\mathcal{I}$, is clearly huge. Apart from this huge search space, one of the greatest costs of an Apriori-like algorithm is the number of scans it has to make through the database, because these databases tend to be very large, and hence they do no fit into main memory. In the standard Apriori algorithm, the number of scans through the database equals the maximal size of a candidate pattern. Several improvements on the Apriori algorithm try to reduce the number of scans through the database by estimating the number of candidate patterns that can still be generated.

At the heart of all these techniques lies the following purely combinatorial problem, that must be solved first before we can seriously start applying them: *given the current set of frequent patterns at a certain pass of the algorithm, what is the maximal number of candidate patterns that can be generated in the passes yet to come?*

Our contribution is to solve this problem by providing a hard and tight combinatorial upper bound. By computing our upper bound after every pass of the algorithm, we have at all times a watertight guarantee on the size of what is still to come, on which we can then base various optimization decisions, depending on the specific algorithm that is used.

In the next Section, we will discuss existing techniques to reduce the number of database scans, and point out the dangers of using existing heuristics for this purpose. using our upper bound, these techniques can be made watertight. In Section 3, we derive our upper bound, using a combinatorial result from the sixties by Kruskal and Katona. In Section 4, we show how to get even more out of this upper bound by applying it recursively. In Section 5, we discuss several issues concerning the implementation of the given upper bounds on top of Apriori-like algorithms. In Section 6, we give experimental results, showing the effectiveness of our result in estimating, far ahead, how much will still be generated in the future. Finally, we conclude the paper in Section 7.

2 Related Work

Nearly all frequent pattern mining algorithms developed after the proposal of the Apriori algorithm, rely on its levelwise candidate generation and pruning strategy. Other strategies try to push certain constraints into the candidate pattern generation as deeply as possible to reduce the number of candidate patterns that must be generated. The first succesful attempt to generate frequent patterns without candidate generation was recently proposed by Han et al. in [6]. However, this approach requires that a compressed form of the database fits in main memory. The possibility of a disk-resident variation of the algorithm was discussed, but its performance against Apriori-like algorithms was not further examined.

The first heuristic proposed to reduce the number of database scans was used in the AprioriHybrid algorithm [2, 3]. This algorithm uses Apriori in the initial passes and switches to AprioriTid if it expects it to run faster. This AprioriTid algorithm does not use the database at all for counting the support of candidate patterns. Rather, an encoding of the candidate patterns used in the previous pass is employed for this purpose. The AprioriHybrid algorithm switches to AprioriTid when it expects this encoding of the candidate patterns to be small enough to fit in main memory. The size of the encoding grows with the number of candidate patterns. Therefore, it calculates the size the encoding would have in the current pass. If this size is small enough and there were fewer candidate patterns in the current level than the previous pass, the heuristic decides to switch to AprioriTid.

This heuristic (like all heuristics) is not waterproof, however. Take, for example, two disjoint datasets. The first dataset consists of all subsets of a frequent pattern of size 20. The second dataset consists of all subsets of 1 000 disjoint frequent patterns of size 5. If we merge these two datasets, we get $1\,140 + 10\,000$ patterns of size 3 and $4\,845 + 5\,000$ patterns of size 4. If we have enough memory to store the encoding for all these patterns, then the heuristic decides to switch to AprioriTid. This decision is premature, however, because the number of new patterns in each pass will start growing exponentially afterwards.

Another improvement of the Apriori algorithm, which is part of the folklore, tries to combine as many iterations as possible in the end, when only few candidate patterns can still be generated. The potential of such a combination technique was realized early on [2, 10], but the modalities under which it can be applied were never further examined.

Note that the reduction of database scans is not the only application which can profit from the upper bounds we will give in this paper. For example, the sampling algorithm proposed by Toivonen [11] performs only one single scan through the database by picking a random sample from the database, then finding all frequent patterns that probably hold in the whole database, and then verifying the results with the rest of the database. In the cases where the sampling method does not produce all frequent patterns, the missing patterns can be found by generating all candidate patterns and verifying their frequencies during a second pass through the database. Obviously, there exists the chance that the number of missed patterns can become very large. To prevent this from happening, one can apply our upper bounds to make sure that this number is feasible.

3 The basic upper bounds

In all that follows, L is some family of patterns of size k.

Definition 1. A *candidate pattern* for L is a pattern (of size larger than k) of which all k-subsets are in L. For a given $\ell \geq k$, we denote the set of all size-ℓ candidate patterns for L by $C_\ell(L)$.

For any $p \geq 1$, we will provide an upper bound on $|C_{k+p}(L)|$ in terms of $|L|$. The following lemma is central to our approach: (a simple proof was given by Katona [8])

Lemma 2. *Given n and k, there exists a unique representation*

$$n = \binom{m_k}{k} + \binom{m_{k-1}}{k-1} + \cdots + \binom{m_r}{r},$$

with $r \geq 1$, $m_k > m_{k-1} > \ldots > m_r$, and $m_i \geq i$ for $i = r, r+1, \ldots, k$.

This representation is called the *k-canonical representation of n* and can be computed as follows: The integer m_k satisfies $\binom{m_k}{k} \leq n < \binom{m_k+1}{k}$, the integer m_{k-1} satisfies $\binom{m_{k-1}}{k-1} \leq n - \binom{m_k}{k} < \binom{m_{k-1}+1}{k-1}$, and so on, until $n - \binom{m_k}{k} - \binom{m_{k-1}}{k-1} - \cdots - \binom{m_r}{r}$ is zero.

We now establish:

Theorem 3. *If*

$$|L| = \binom{m_k}{k} + \binom{m_{k-1}}{k-1} + \cdots + \binom{m_r}{r}$$

in k-canonical representation, then

$$|C_{k+p}(L)| \leq \binom{m_k}{k+p} + \binom{m_{k-1}}{k-1+p} + \cdots + \binom{m_{s+1}}{s+p+1},$$

where s is the smallest integer such that $m_s < s + p$. If no such integer exists, we set $s = r - 1$.

Proof. Suppose, for the sake of contradiction, that

$$|C_{k+p}(L)| \geq \binom{m_k}{k+p} + \binom{m_{k-1}}{k-1+p} + \cdots + \binom{m_{s+1}}{s+p+1} + \binom{s+p}{s+p}.$$

Note that this is in $k+p$-canonical representation. A theorem by Kruskal and Katona [5, 8, 9] says that

$$|L| \geq \binom{m_k}{k} + \binom{m_{k-1}}{k-1} + \cdots + \binom{m_{s+1}}{s+1} + \binom{s+p}{s}.$$

But this is impossible, because

$$\begin{aligned}
|L| &= \binom{m_k}{k} + \binom{m_{k-1}}{k-1} + \cdots + \binom{m_{s+1}}{s+1} + \binom{m_s}{s} + \cdots + \binom{m_r}{r} \\
&\leq \binom{m_k}{k} + \binom{m_{k-1}}{k-1} + \cdots + \binom{m_{s+1}}{s+1} + \sum_{1 \leq i \leq s} \binom{i+p-1}{i} \\
&< \binom{m_k}{k} + \binom{m_{k-1}}{k-1} + \cdots + \binom{m_{s+1}}{s+1} + \sum_{0 \leq i \leq s} \binom{i+p-1}{i} \\
&= \binom{m_k}{k} + \binom{m_{k-1}}{k-1} + \cdots + \binom{m_{s+1}}{s+1} + \binom{s+p}{s}.
\end{aligned}$$

The first inequality follows from the observation that $m_s \leq s+p-1$ implies $m_i \leq i+p-1$ for all $i = s, s-1, \ldots, r$. The last equality follows from a well-known binomial identity. $\square$

Notation We will refer to the upper bound provided by the above theorem as $KK_k^{k+p}(|L|)$ (for Kruskal-Katona). The subscript k, the level at which we are predicting, is important, as the only parameter is the cardinality $|L|$ of L, not L itself. The superscript $k+p$ denotes the level we are predicting. We point out:

Proposition 4 (Tightness). *The upper bound provided by Theorem 3 is* tight: *for any given n, k and p there always exists an L with $|L| = n$ such that $|C_{k+p}(L)| = KK_k^{k+p}(|L|)$.*

Proof. Let us write a finite set of natural numbers as a string of natural numbers by writing its members in decreasing order. We can then compare two such sets by comparing their strings in lexicographic order. The resulting order on the sets is known as the colexicographic order. Now consider the collection L of the first n k-sets of natural numbers in this colexicographic order. It is readily verified that this L has the desired property. $\square$

Analogous tightness properties hold for all upper bounds we will present in this paper, but we will no longer explicitly state this.

Estimating the number of levels The k-canonical representation of $|L|$ also yields an upper bound on the maximal size of a candidate pattern, denoted by maxsize(L). Recall that this size equals the number of iterations the standard Apriori algorithm will perform. Indeed, since $|L| < \binom{m_k+1}{k}$, there cannot be a candidate pattern of size $m_k + 1$ or higher, so:

Proposition 5. *If $\binom{m_k}{k}$ is the first term in the k-canonical representation of $|L|$, then* $\text{maxsize}(L) \le m_k$.

We denote this number m_k by $\mu_k(|L|)$. From the form of KK_k^{k+p} as given by Theorem 3, it is immediate that μ also tells us the last level before which KK becomes zero. Formally:

Proposition 6. $\mu_k(|L|) = k + \min\{p \mid KK_k^{k+p}(|L|) = 0\} - 1$.

Estimating all levels As a result of the above, we can also bound, at any given level k, the *total* number of candidate patterns that can be generated, as follows:

Proposition 7. *The total number of candidate patterns that can be generated from a set L of k-patterns is at most*

$$KK_k^{\text{total}}(L) := \sum_{p=1}^{\mu_k(|L|)} KK_k^{k+p}(|L|).$$

4 Getting the most out of it

The upper bound KK on itself is neat and simple as it takes as parameters only two numbers: the current size k, and the number $|L|$ of current frequent patterns. However, in reality, when we have arrived at a certain level k, we do not merely have the cardinality: we have the actual set L of current k-patterns! For example, if the frequent patterns in the current pass are all disjoint, our current upper bound will still estimate their number to a certain non-zero figure. However, by the pairwise disjointness, it is clear that no further patterns will be possible at all. In sum, because we have richer information than a mere cardinality, we should be able to get a better upper bound. We next show how this is possible, by applying the upper bound recursively on the *structure* of the collection of current frequent patterns.

Some notation L still denotes some set of patterns of size k. Let I be the set of items occurring in L. For an arbitrary item x, define the set L^x as

$$L^x = \{s - \{x\} \mid s \in L \text{ and } x = \min s\}.$$

Also, for any set of patterns H, denote the set $\{h + \{x\} \mid h \in H\}$ simply by $H + x$.

Improved upper bound We now define the following recursive function, for $p > 0$:

$$KK^*_{k+p}(L) := \begin{cases} \binom{|L|}{p+1} & \text{if } k = 1; \\ \min\{KK^{k+p}_k(|L|), \sum_{x \in I} KK^*_{k+p-1}(L^x)\} & \text{if } k > 1. \end{cases}$$

(For the base case, note that $\binom{|L|}{p+1}$, when $k = 1$, is nothing but $KK^{k+p}_k(|L|)$.)

By definition, KK^*_{k+p} is always smaller than KK^{k+p}_k for each $p > 0$. We prove that it is still an upper bound on the number of candidate patterns of size $k + p$:

Theorem 8. *$|C_{k+p}(L)| \leq KK^*_{k+p}(L)$.*

Proof. By induction on k. The base case $k = 1$ is clear. For $k > 1$, it suffices to show that for all $p > 0$

$$C_{k+p}(L) \subseteq \bigcup_{x \in I} C_{k+p-1}(L^x) + x. \tag{1}$$

Indeed, all the L^x are pairwise disjoint, and thus also all the $C_{k+p-1}(L^x)+x$. From the above containment we can therefore conclude

$$\begin{aligned} |C_{k+p}(L)| &\leq |\bigcup_{x \in I} C_{k+p-1}(L^x) + x| \\ &= \sum_{x \in I} |C_{k+p-1}(L^x) + x| \\ &= \sum_{x \in I} |C_{k+p-1}(L^x)| \\ &\leq \sum_{x \in I} KK^*_{k+p-1}(L^x) \end{aligned}$$

where the last inequality is by induction.

To show (1), we need to show that for every $p > 0$ and every $s \in C_{k+p}(L)$, $s - \{x\} \in C_{k+p-1}(L^x)$, where $x = \min s$. This means that every subset of $s - \{x\}$ of size $k - 1$ must be an element of L^x. Let $s - \{x\} - \{y_1, \ldots, y_p\}$ be such a subset. This subset is an element of L^x iff $s - \{y_1, \ldots, y_p\} \in L$ and $x = \min(s - \{y_1, \ldots, y_p\})$. The first condition follows from $s \in C_{k+p}(L)$, and the second condition is trivial. Hence the theorem. $\square$

A natural question is why we must take the minimum in the definition of KK^*. The answer is that the two terms of which we take the minimim are incomparable. The example of an L where all patterns are pairwise disjoint, already mentioned in the beginning of this section, shows that, for example, $KK_k^{k+1}(|L|)$ can be larger than the summation $\sum_{x \in I} KK_k^*(L^x)$. But the converse is also possible: consider $L = \{\{1,2\},\{1,3\}\}$. Then $KK_2^3(L) = 0$, but the summation yields 1.

We can also improve the upper bound $\mu_k(|L|)$ on maxsize(L). Indeed, in analogy with Proposition 6, we define:

$$\mu^*(L) := k + \min\{p \mid KK_{k+p}^*(L) = 0\} - 1.$$

We then have:

Proposition 9. maxsize$(L) \leq \mu^*(L) \leq \mu(L)$.

We finally use Theorem 8 for improving the upper bound KK_k^{total} on the total number of candidate patterns. Indeed, define:

$$KK_{\text{total}}^*(L) = \sum_{p=1}^{\mu^*(L)} KK_{k+p}^*(L)$$

Then we have:

Proposition 10. *The total number of candidate patterns that can be generated from a set L of k-patterns is bounded by $KK_{\text{total}}^*(L)$. Moreover, $KK_{\text{total}}^*(L) \leq KK_k^{\text{total}}(L)$.*

5 Efficient Implementation

To evaluate our upper bounds we implemented an optimized version of the Apriori algorithm using a trie data structure to store all generated patterns, similar to the one described by Brin et al. [4]. This trie structure makes it cheap and straightforward to implement the computation of all upper bounds. Indeed, a top-level subtrie (rooted at some singleton pattern $\{x\}$) represents exactly the set L^x we defined in Section 4. Every top-level subtrie of this subtrie (rooted at some two-element pattern $\{x, y\}$) then represents $(L^x)^y$, and so on. Hence, we can compute the recursive bounds while traversing the trie, after the frequencies of all candidate patterns are counted, and we have to traverse the trie once more to remove all candidate patterns that turned out to be infrequent. This can be done as follows.

Remember, at that point, we have the current set of frequent patterns of size k stored in the trie. First of all, we have to keep in every node its depth

d (i.e. the size of the frequent pattern it represents), and c, the number of descendants at depth k this node has. For every node at depth smaller than k, we compute the $k-d$-canonical representation of c, which can be used to compute $\mu_{k-d}(c)$ (cf. Proposition 5), $KK^{\ell}_{k-d}(c)$ for any $\ell \leq \mu_{k-d}(c)$ (cf. Theorem 3) and hence also $KK^{\text{total}}_{k-d}(c)$ (cf. Proposition 7). For every node at depth $k-1$, its KK^* and μ^* values are equal to its KK and μ values respectively. Then compute for every $p > 0$, the sum of the $KK^*_{k-d+p-1}$ values of all its children, and let KK^*_{k-d+p} be the smallest of this sum and KK^{k-d+p}_{k-d} until this minimum becomes zero, which also gives us the value of μ^*. Finally, we can compute KK^*_{total} for this node. If this is done for every node, traversed in a depth-first manner, then finally the root node will contain the upper bounds on the number of candidate patterns that can still be generated, and on the maximum size of any such pattern. The soundness and completeness of this method follows directly from the theorems and propositions of the previous sections.

We should also point out that, since the numbers involved can become exponentially large (in the number of items), an implementation should take care to use arbitrary-length integers such as provided by standard mathematical packages. Since the length of an integer is only logarithmic in its value, the lengths of the numbers involved will remain polynomially bounded.

6 Experimental Evaluation

Data sets We have experimented using several synthetic datasets generated by the program provided by the Quest research group at IBM Almaden. For different settings of the parameters of the generator, the resulting figures were very analogous. We also experimented using a real market basket dataset from a Belgian retail store containing 41 373 transactions. The store carries 13 103 products. The results from this experiment were not immediately as good as the results from the synthetic datasets. The reason for this, however, turned out to be the bad ordering of the items, as explained next.

Reordering From the form of L^x, it can be seen that the order of the items can effect the recursive upper bounds. By computing the upper bound only for a subset of all frequent patterns, we win by incorporating the structure of the current collection of frequent patterns, but we also lose some information, because whenever we recursively restrict ourselves to a subtrie L^x, then for

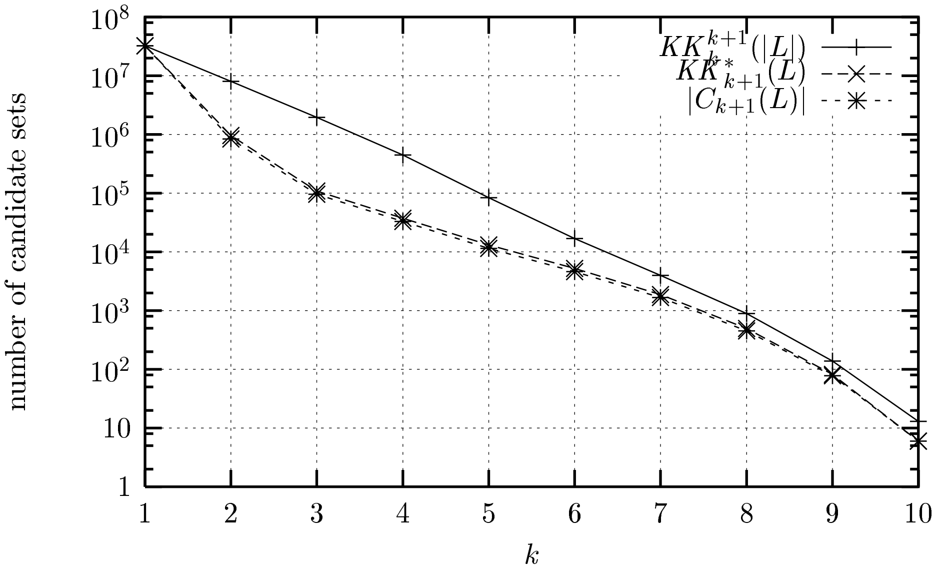

Figure 1: The maximal number of candidate patterns at the next level.

every candidate pattern s with $x = \min s$, we lose the information about exactly one subpattern in L, namely $s - x$, because all patterns in L^x come from patterns in L that contain x. We therefore would like to make it likely that many of these excluded patterns are frequent. A good heuristic, which has already been used for several other optimizations in frequent pattern mining [7, 4], is to force the most frequent items to appear in the most candidate patterns, by reordering the single item patterns in increasing order of frequency.

After reordering the items in the real life dataset, using this heuristic, the results became very analogous with the results using the synthetic datasets, so we refrain from giving multiple analogous figures. The figures we will show are those from the real life dataset.

Results Figure 1 shows, after each level k, the computed upper bound KK and improved upper bound KK^* for the number of candidate patterns at the next level, as well as the actual number it turned out to be.

Figure 2 shows the computed upper bounds μ and μ^* on the maximal size of a candidate pattern. In this experiment, this maximum turned out to be 11.

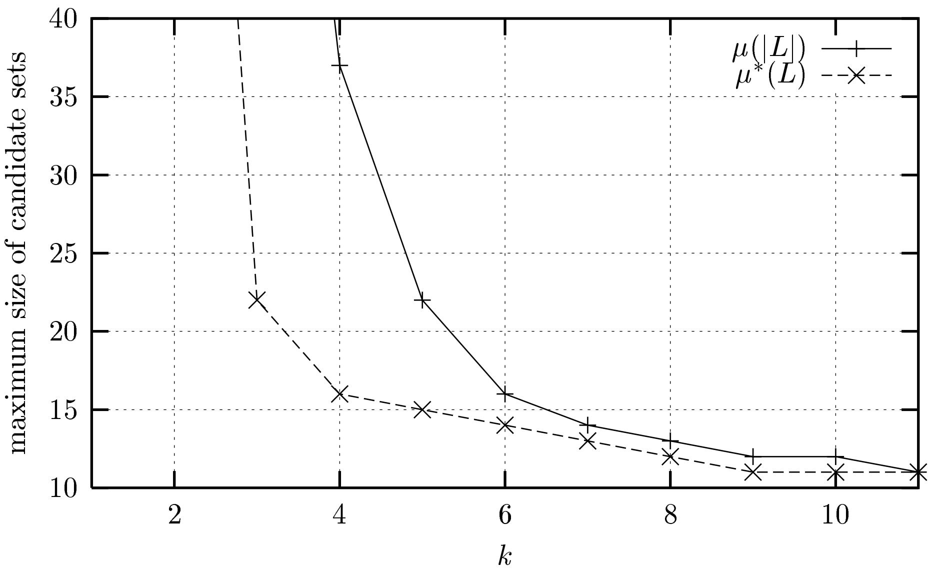

Figure 2: The size of the largest possible candidate pattern.

Figure 3 shows the upper bounds on the total number of candidate patterns that could still be generated, compared to the number of candidate patterns, $|C_{\text{total}}|$, that effectively was still generated when we combined all further passes into a single one.

Discussion The results are pleasantly surprising. First, note that the improvement of KK^* over KK, and of μ^* over μ, anticipated by our theoretical discussion, is indeed dramatic. Second, comparing the computed upper bounds with the actual numbers, we observe the high accuracy of the estimations given by KK^*. Indeed, the estimations of KK^*_{k+1} match almost exactly the actual number of candidate patterns that has been generated at level $k+1$. The upper bounds on the total number of candidate patterns are still very large when estimated in the first two passes, which is not surprising because at these initial stages, there is not much information yet. From the fourth pass on, however, when the frequent patterns of size 4 are known, the estimations become almost exact. Since the totals become manageable from there on, an optimizer using our estimates would decide to combine all iterations 5–11 in a single one, and would have generated at most 142 237 more candidate patterns.

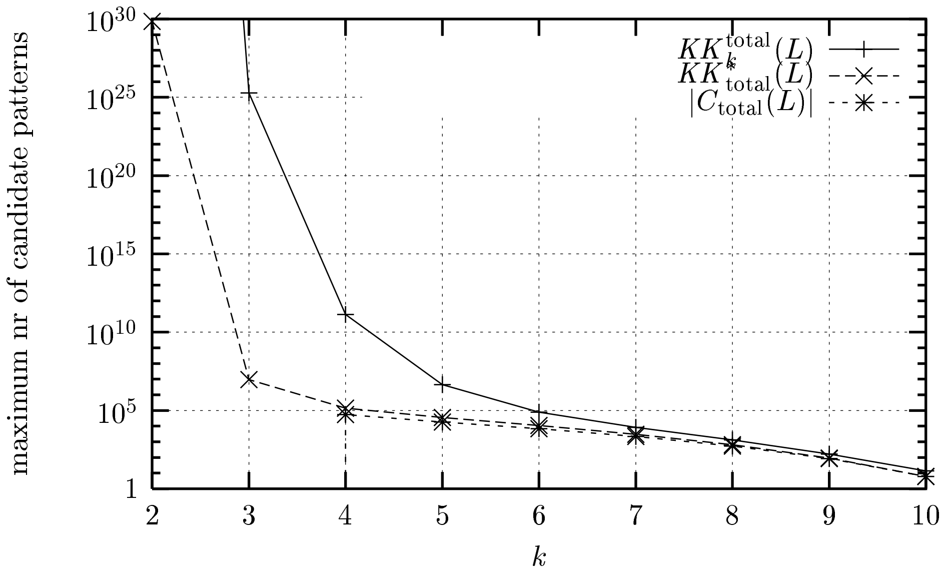

Figure 3: The maximal number of candidate patterns yet to be generated.

7 Conclusion

Motivated by several heuristics to reduce the number of database scans in the context of frequent pattern mining, we provide a hard and tight combinatorial upper bound on the number of candidate patterns and on the size of the largest possible candidate pattern, given a set of frequent patterns. Our findings are not restricted to a single algorithm, but can be applied to any frequent pattern mining algorithm which is based on the levelwise generation of candidate patterns. Using the standard Apriori algorithm, on which most frequent pattern mining algorithms are based, our experiments showed that these upper bounds can be used to considerably reduce the number of database scans without taking the risk of getting a combinatorial explosion of the number of candidate patterns. In future work, we would like to investigate how other levelwise frequent pattern mining algorithms could benefit from implementing our upper bounds.

Acknowledgment

The second author would like to thank Hannu Toivonen for some inspiring discussions on the subject.

References

[1] R. Agrawal, T. Imielinski, and A.N. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, volume 22:2 of *SIGMOD Record*, pages 207–216. ACM Press, 1993.

[2] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A.I. Verkamo. Fast discovery of association rules. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 307–328. MIT Press, 1996.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. IBM Research Report RJ9839, IBM Alamaden Research Center, San Jose, California, June 1994.

[4] S. Brin, R. Motwani, J.D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, volume 26:2 of *SIGMOD Record*, pages 255–264. ACM Press, 1997.

[5] P. Frankl. A new short proof for the Kruskal–Katona theorem. *Discrete Mathematics*, 48:327–329, 1984.

[6] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, volume 29:2 of *SIGMOD Record*, pages 1–12. ACM Press, 2000.

[7] R.J. Bayardo Jr. Efficiently mining long patterns from databases. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, volume 27:2 of *SIGMOD Record*, pages 85–93. ACM Press, 1998.

[8] G.O.H. Katona. A theorem of finite sets. In *Theory Of Graphs*, pages 187–207. Akadémia Kiadó, 1968.

[9] J.B. Kruskal. The number of simplices in a complex. In *Mathematical Optimization Techniques*, pages 251–278. Univ. of California Press, 1963.

[10] H. Mannila, H. Toivonen, and A. I. Verkamo. Efficient algorithms for discovering association rules. In *Proceedings AAAI Workshop on Knowledge Discovery in Databases*, pages 181–192. AAAI Press, 1994.

[11] H. Toivonen. Sampling large databases for association rules. In *Proceedings 22nd International Conference on Very Large Data Bases*, pages 134–145. Morgan Kaufmann, 1996.