# Fast estimation of the pattern frequency spectrum

Matthijs van Leeuwen[1] and Antti Ukkonen[2]

[1] Department of Computer Science, KU Leuven, Belgium
[2] Helsinki Institute for Information Technology HIIT, Aalto University, Finland
matthijs.vanleeuwen@cs.kuleuven.be, antti.ukkonen@aalto.fi

**Abstract.** Both exact and approximate counting of the number of frequent patterns for a given frequency threshold are hard problems. Still, having even coarse prior estimates of the number of patterns is useful, as these can be used to appropriately set the threshold and avoid waiting endlessly for an unmanageable number of patterns. Moreover, we argue that the number of patterns for different thresholds is an interesting summary statistic of the data: the *pattern frequency spectrum*.
To enable fast estimation of the number of frequent patterns, we adapt the classical algorithm by Knuth for estimating the size of a search tree. Although the method is known to be theoretically suboptimal, we demonstrate that in practice it not only produces very accurate estimates, but is also very efficient. Moreover, we introduce a small variation that can be used to estimate the number of patterns under constraints for which the Apriori property does not hold. The empirical evaluation shows that this approach obtains good estimates for closed itemsets.
Finally, we show how the method, together with isotonic regression, can be used to quickly and accurately estimate the frequency pattern spectrum: the curve that shows the number of patterns for every possible value of the frequency threshold. Comparing such a spectrum to one that was constructed using a random data model immediately reveals whether the dataset contains any structure of interest.

## 1 Introduction

Pattern mining aims to enable the discovery of patterns from data. As such, it is one of the most-studied problems in exploratory data mining. A *pattern* is a description of some structure that occurs locally in the data. That is, a pattern is an element of a given pattern language $\mathcal{L}$ that describes a subset of a dataset $\mathcal{D}$. The most commonly used formalisation is theory mining, where the goal is to find the theory $Th(\mathcal{L};\mathcal{D};q) = \{X \in \mathcal{L} \mid q(X,\mathcal{D}) = \texttt{true}\}$, with $q$ a selection predicate that returns true iff $X$ satisfies the imposed constraints on $\mathcal{D}$.

The best-known instance of pattern mining is frequent itemset mining [1], which discovers sets of items that frequently occur together in transactional data. Given a minimum support threshold $\sigma$, the theory to be mined consists of all itemsets that occur at least $\sigma$ times in the data. That is, $Th(\mathcal{L};\mathcal{D};q) =$

$\{X \in \mathcal{L} \mid \text{freq}(X, \mathcal{D}) \geq \sigma\}$, where $\text{freq}(X, \mathcal{D})$ denotes the *frequency* of $X$ in $\mathcal{D}$, i.e., the number of transactions in which the pattern occurs. In general, frequent pattern mining techniques have been developed for quite some data types and corresponding pattern types, e.g., for sequences [2], and for graphs [18].

A major problem in frequent pattern mining is that choosing low values for the minimum frequency threshold results in vast amounts of patterns – the infamous *pattern explosion*. One may try to avoid this by choosing the threshold such that the number of patterns is still manageable. Parameter tuning can be a tricky business though, because small changes in $\sigma$ often have a large impact on the number of patterns. For that reason, it would be beneficial to know how many patterns to expect without having to actually mine them.

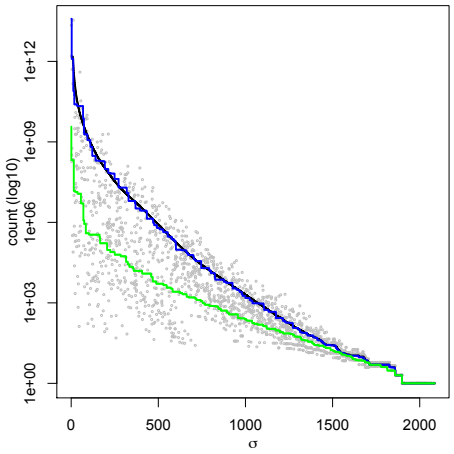


**Fig. 1.** Exact (black line) and estimated (blue line) numbers of patterns for all possible frequency thresholds in *Mammals*. The green line shows the expected number of patterns in random data having the same column marginals, the grey dots represent individual path sample estimates (see Alg. 1).

Unfortunately, both exact and approximate counting of the number of frequent patterns for a given frequency threshold are hard problems [11,6,19]. Nevertheless, having even coarse prior estimates of the number of patterns is useful, as these can be used to appropriately set the threshold $\sigma$ and avoid waiting endlessly for an unmanageable number of patterns. In this paper we introduce methods for the fast estimation of the number of frequent patterns in a dataset, both for individual thresholds and the complete frequency spectrum.

Figure 1 illustrates both the pattern explosion and the accuracy of our method. Note that the counts on the y-axis are in logarithmic scale ($\log_{10}$). It took only 11 seconds to accurately estimate the curve for the complete frequency range, whereas the exact curve took over 12 hours to compute and still does not go lower than a frequency threshold of 10. A quick comparison to the expected curve, which is computed under the assumption that all items are independent, shows that the dataset contains quite some structure: there are many more itemsets than can be explained by this simple model.

**Using frequent patterns for knowledge discovery** In practice, frequent pattern mining is seldom used as final step in the KDD process, because interpretation of a large amount of patterns by a domain expert is impracticable. Nevertheless, frequent pattern mining and hence estimating the number of patterns are important problems.

The most important reason is that frequent patterns are often used as input for some other algorithm, as part of the KDD process. Pattern-based approaches to data mining are popular, in particular for exploratory purposes. Using patterns has clear advantages, the most obvious one being that patterns are interpretable representations and can thus provide explanations.

Many of these techniques can be captured under the umbrella term *pattern set mining* [7], a class of solutions proposed to address the pattern explosion. This is commonly achieved by imposing constraints and/or an optimisation criterion on the complete set of patterns being mined. Pattern set mining methods commonly require a large number of frequent patterns as input. Examples include KRIMP [17] and the iterative data mining framework by De Bie [4].

Another context in which frequent patterns are used is pattern-based classification: Cheng et al. [9], for example, construct a classifier from a large set of frequent patterns. Finally, frequent patterns can serve as input for interactive exploration, for example, using the MIME tool [10]. In all these cases, frequent patterns have to be mined and a frequency threshold needs to be chosen. Hence, having an estimate for the number of patterns given a certain threshold is useful.

**Approach and contributions** The first main contribution of this paper is the FASTEST algorithm, for Fast Estimation. It is *a fast and accurate method for estimating the number of frequent patterns for a given dataset and frequency threshold*. For this, we adapt the classical algorithm by Knuth [14] for estimating the size of a search tree. We demonstrate that the method is fast and produces very accurate estimates in practice. In particular, we focus on frequent itemsets and rely on the Apriori property, which states that any subset of a frequent itemset must also be frequent. Our method can be easily applied to other types of data and patterns, as long as the Apriori, or monotonicity, property holds.

We also introduce a small variation of the method that can be used to estimate the number of patterns under constraints for which the Apriori property does not hold. The variation empirically adjusts the total number of estimated frequent patterns for the considered constraints.

The second main contribution is *a method for efficiently estimating the total number of frequent patterns for all possible frequency thresholds*: *the pattern frequency spectrum*. The algorithm, dubbed SPECTRA, uses isotonic regression [3] to compute a complete spectrum from point estimates obtained with FASTEST. The resulting curves are extremely accurate and provide useful summary statistics of the data. To demonstrate this, *we investigate spectra constructed for random data*, i.e., assuming that all items in a dataset are independent. Comparing an actual to a randomized spectrum immediately reveals whether the dataset contains any structure of interest (see Fig. 1).

The remainder of this paper is organised as follows. First, we discuss related work in Section 2. Next, in Section 3 we describe the classical algorithm by Knuth upon which we base our method. Sections 4 and 5 introduce our techniques for estimating the number of patterns in a dataset for individual thresholds and threshold ranges respectively. We present the empirical evaluation in Section 6, after which we round up with discussion and conclusions in Sections 7 and 8.

## 2 Related work

We briefly discuss two categories of related work that are relevant to our work: 1) frequent pattern counting, and 2) estimating the size of a search tree.

**Frequent pattern counting** Exactly counting the number of frequent patterns is #P-complete [11], as is counting the number of maximal frequent itemsets [19]. For that reason, methods designed specifically for this task usually compute approximate counts. Boley and Grosskreutz [6] devised an approximate counting algorithm for frequent itemsets that is based on MCMC simulation [12]. Although technically solid, the use of MCMC simulation makes the method rather complex. One of the aims of this paper is to develop a much simpler method that is therefore easier to implement and use. We will compare our algorithm to the MCMC-based method in Section 6. Later, Boley et al. [5] proposed a similar method for sampling and counting closed itemsets.

The alternative approach is to perform exact counting by adapting existing pattern mining techniques for this purpose. Highly optimised frequent itemset mining implementations have been available since the FIMI workshops[3] in 2003 and 2004. One such very efficient implementation is AFOPT [15], which we will use as a baseline in our experiments.

**Estimating search tree sizes** Knuth's original algorithm [14] was proposed for the problem of studying algorithm performance on a given input by estimating the size of the associated search tree. Purdom suggested a modification that incorporates partial backtrack into the algorithm [16]. This will lead to better estimates when the trees have some long and "thin" branches. Kilby et al. [13] addressed the same problem using a slightly different technique, but focused on binary trees. Chen [8] proposed a generalisation of Knuth's method that is based on the idea of stratified sampling. The algorithm we propose for estimating the number of closed patterns has some similarities with this method.

## 3 Preliminaries

This section provides a short recap on the tree size estimation algorithm by Knuth [14] that we will use as foundation for our algorithms.

Most combinatorial problems can be framed in terms of finding an assignment of values to a set of variables, so that the assignment satisfies some constraints and optimizes an objective function. This is also true for pattern mining problems, except that one aims to find all assignments that satisfy the constraints.

*Backtracking algorithms* perform depth-first search over feasible assignments and can be characterised in terms of a *search tree*, where the root contains the "empty" assignment, and nodes correspond to (partial or complete) assignments. The size of the search tree is a practical measure of the hardness of the problem instance, because a simple backtracking algorithm must enumerate all feasible assignments. However, counting the number of nodes is a hard problem, and can

[3] `http://fimi.ua.ac.be/`

usually be solved exactly only by an exhaustive traversal of the search tree. That is, knowing the hardness of a problem instance requires us to solve it!

Knuth proposed an algorithm [14] that computes an *estimate* of the size of the search tree without exhaustive traversal. The intuition of the algorithm is the following: If a search tree is perfectly regular (every internal node has the same outdegree), we can compute its exact size by summing the sizes of every level of the tree. The size of a level is given by the *product of the outdegrees* observed on a path from the root node that ends just above the level.

*Example 1.* Consider a complete binary tree with $h$ levels, including the root at level 1. The size of every level $l$, $2 \leq l \leq h$, is $\prod_{i=1}^{l-1} 2 = 2^{l-1}$, and summing these yields $1 + \sum_{l=2}^{h} 2^{l-1} = \sum_{l=0}^{h-1} 2^l = 2^h - 1$, that is, the number of nodes in a complete binary tree.

Since search trees that arise in practice are rarely (if ever) regular, determining the size by only considering a single path from the root to a leaf is not going to produce the correct size. However, we can consider a number of *random paths*, and compute a *path estimate* for each. In detail, let $(v_1, v_2, \ldots, v_h)$ denote a random path from the root $v_1$ to a leaf $v_h$ in a search tree, and let $d(v)$ denote the outdegree of node $v$. The associated path estimate is given by the sum

$$1 + \sum_{i=2}^{h} \prod_{l=1}^{i-1} d(v_l), \tag{1}$$

where the product $\prod_{l=1}^{i-1} d(v_l)$ is the estimate associated with the $i$th level of the tree. Notice that the path estimate is a sum of such levelwise estimates.

To compute a single path estimate, the algorithm starts from the root, selects one child at random at every level until it reaches a leaf, and then applies Eq. 1. The final estimate is defined as *the average of the path estimates* from a number of random paths. Depending on their number, this process only considers a very small part of the search tree, but can in practice obtain an accurate and *unbiased* estimate of the tree size [14].

## 4 Estimating the number of patterns

Let a database $\mathcal{D}$ be a bag of transactions over a set of items $\mathcal{I}$, where a transaction $t$ is a subset of $\mathcal{I}$, i.e., $t \subseteq \mathcal{I}$. Furthermore, a pattern $X$ is an itemset, $X \subseteq \mathcal{I}$, and pattern language $\mathcal{L}$ is the set of all such possible patterns, $\mathcal{L} = 2^{\mathcal{I}}$. An itemset $X$ occurs in a transaction $t$ iff $X \subseteq t$, and its frequency is defined as the number of transactions in $\mathcal{D}$ which it occurs, i.e., $\text{freq}(X, \mathcal{D}) = |\{t \subseteq D \mid X \subseteq t\}|$. A pattern $X$ is said to be frequent iff its frequency exceeds the minimum frequency threshold $\sigma$. The frequent itemset mining problem is to find all frequent patterns, i.e., all $X \in \mathcal{L}$ for which $\text{freq}(X, \mathcal{D}) \geq \sigma$. Frequent itemsets can be mined efficiently due to monotonicity of the frequency constraint with respect to set inclusion, which is also known as the Apriori property [1]. This property states that for any frequent itemset $X$, all itemsets $Y \subseteq X$ must also be frequent.

### 4.1 Frequent patterns and the FastEst algorithm

Next we describe a modification to Knuth's algorithm [14] and use it to estimate the number of frequent itemsets[4]. The core question we must address is how to turn the task of counting frequent itemsets to that of estimating the size of a tree. By constructing a tree where every node corresponds to a frequent itemset, we can use Knuth's method. The following discussion focuses on frequent itemsets, but the same approach can be applied also to other patterns that are constructed "piece-by-piece" from elements of some language.

For any itemset $X$, let $X \cup u$, where $u$ is some item not in $X$, denote an *expansion* of $X$. Consider a tree $T$ rooted at $\emptyset$, where each node is a frequent itemset. The children of a node $X$ are all of its frequent expansions. More formally,

$$\text{children}(X) = \{X \cup u \mid u \in \{\mathcal{I} \setminus X\} \wedge \ \text{freq}(X \cup u) \geq \sigma\}.$$

That is, the root $\emptyset$ has all singleton items having a high enough frequency as children, these have all frequent pairs as children, and so on. The leaves of the tree correspond to *maximal* frequent itemsets, i.e., those that cannot be expanded without violating the frequency constraint. Note that there cannot be any itemset that is frequent but *not* in $T$, because of the monotonicity of the frequency constraint: if itemset $X$ is not frequent, no $Y \supset X$ can be frequent either.

Now we could use Knuth's algorithm "as is" to estimate the size of $T$. However, observe that $T$ contains multiple copies of the frequent itemsets. The tree $T$ is in fact an "unfolded" *itemset lattice*. Indeed, every frequent itemset $X$ of size $|X|$ is contained in $T$ exactly $|X|!$ times. This is because every node of $T$ is connected to the root $\emptyset$ by a path where the items in $X$ are added one by one in some particular order, and this happens in as many ways as there are permutations of $|X|$ items.

To obtain a proper estimate of the number of frequent itemsets, we need to correct for this property of $T$. As noted earlier, Eq. 1 is in fact a sum over all levels of the tree. In the tree $T$, the level $i$ (with root $\emptyset$ on level 0) contains $i!$ copies of the same itemset. We must thus replace Equation 1 with

$$1 + \sum_{i=1}^{l} \frac{1}{i!} \prod_{j=0}^{i-1} d(v_j), \tag{2}$$

where $\frac{1}{i!}$ corrects the sizes of the levels so that each itemset is counted only once.

Pseudocode of the full FastEst algorithm is shown in Algorithm 1. In short, this is Knuth's algorithm applied on the frequent pattern lattice combined with the modified path estimate equation. In practice we do not materialise the tree $T$, but only sample paths through it using the SamplePath subroutine. On every step it finds the set $E$ of extensions to the current pattern $P$ that are still frequent (lines 2 and 8), and the size of $E$ gives the outdegree of $P$ (line 5).

[4] Or any other type of pattern for which the Apriori / monotonicity property w.r.t. pattern inclusion holds.

**Algorithm 1** The FastEst algorithm

1: Sample a number of paths using the SamplePath subroutine.
2: Use Equation 2 to compute the path-specific estimates.
3: **Return** the average of these as the final estimate.

1: SamplePath:
2: $P \leftarrow \emptyset$, $i \leftarrow 0$
3: $E \leftarrow \{x \in \mathcal{I} \mid \text{freq}(P \cup x) \geq \sigma\}$
4: **while** $|E| > 0$ **do**
5: $\quad i \leftarrow i + 1$, $d_i \leftarrow |E|$
6: $\quad e \leftarrow$ random element of $E$
7: $\quad P \leftarrow P \cup e$, $E \leftarrow E \setminus e$
8: $\quad E \leftarrow \{x \in E \mid \text{freq}(P \cup x) \geq \sigma\}$
9: **return** $(d_1, \ldots, d_i)$

The algorithm proceeds until it hits a maximal frequent itemset, after which it returns the sequence of encountered outdegrees.

The main bottleneck when running FastEst are the support computations in SamplePath. These can be made very efficient by using a vertical representation of the database where we have a list of transaction identifiers for each item. As SamplePath proceeds deeper into the tree, we simply maintain a list of transaction identifiers for the current node $P$, and intersect this with the lists for other items when computing the support for each extension (line 8).

**Remark:** Algorithms for pattern mining often perform search by considering a depth-first tree of the patterns. Applying Knuth's method directly on this tree is a bad idea, however. This is because the DFS tree is by construction imbalanced, and therefore the random paths have very different lengths. Most paths will underestimate the tree size, while few paths blow up the estimate. This can, and will, to some extent also happen with the tree $T$ we defined, but since an itemset can be reached along several paths, we expect $T$ to be less imbalanced. Also, the DFS tree has a different structure depending on the order in which the items are considered; $T$ is not dependent on any such ordering.

### 4.2 A non-monotonic constraint: closed patterns

We conclude the section by discussing a simple approach that extends our estimation algorithm for non-monotonic constraints. The example that we focus on are *closed* patterns, i.e., patterns that are frequent and cannot be extended without decreasing the support. More formally, a pattern $X$ is closed iff $\text{freq}(X) > \text{freq}(X \cup u)$ for every possible $u$.

As described in [14], Knuth's algorithm can be modified to count only those nodes of the tree that satisfy a given property. The first idea is thus to use this approach for closed patterns, as closedness is simply a property of the pattern associated with a node. However, since closed patterns can be rare, this approach leads to poor estimates. The high variance of individual path estimates

implies that a very large number of samples are needed to produce reasonable estimates. Instead, we propose a method somewhat related to [8]. This estimates the fractions of closed patterns on each level of the tree $T$, and corrects the final estimate with these.

In more detail, when sampling a path through $T$, we can collect statistics on the number of patterns we observe on every level. We maintain two vectors of counters that have as many elements as there are levels in $T$. The first, denoted $q_p$, counts the number of all patterns found. The second, denoted $q_c$, keeps track of the number of closed patterns we observe. Then, we estimate the fraction of closed patterns on level $l$ by computing $q_c(l)/q_p(l)$. Given the output of Algorithm 1, we can also compute independent estimates for the sizes of every level of $T$. (Recall that Eq. 2 is just the sum of these.) By multiplying these with $q_c(l)/q_p(l)$, we obtain estimates of the numbers of closed patterns on every level. The path-estimate is simply the sum of these, and the final estimate is again the average over a number of random paths.

## 5 The pattern frequency spectrum

The pattern frequency spectrum shows the number of frequent patterns in data $\mathcal{D}$ as a function of $\sigma$. More formally, we define the spectrum as

$$f(\sigma, \mathcal{D}) = |\{X \in \mathcal{L} \mid \text{freq}(X, \mathcal{D}) \geq \sigma\}|.$$

Below we write $f(\sigma)$ for short, unless $\mathcal{D}$ is not clear from the context.

### 5.1 The Spectra algorithm

A simple method to estimate $f(\sigma)$ is to run the FastEst algorithm for a number of fixed values of $\sigma$, and construct $f(\sigma)$ by interpolating from these point estimates. The problem with this approach is that determining the values of $\sigma$ to use is not easy. By using a too coarse grid, we will miss some of the structure present in the frequency spectrum. On the other hand, using many values of $\sigma$ may be very slow. Instead, we will use an approach where we obtain a number of estimates for *random values* of $\sigma$ using the algorithm from the previous section, and then fit a nonlinear regression line through these.

In more detail, we propose the following algorithm called Spectra:

1. Compute the set of points $S = \{(\sigma_1, g(\sigma_1)), \ldots, (\sigma_N, g(\sigma_N))\}$, where every $\sigma_i$ is drawn uniformly at random from some predefined interval, and $g(\sigma_i)$ is a single path estimate given by SamplePath (Alg. 1).
2. We fit a nonlinear, nonincreasing regression line through $S$ and use this as our estimate of the pattern frequency spectrum.

This method has the advantage that we can simultaneously estimate $f(\sigma)$ for all values of $\sigma$, rather than interpolate from point estimates at predefined locations.

We know $f(\sigma)$ must be monotonically nonincreasing as $\sigma$ increases. A suitable method for step 2 is thus *isotonic regression* [3]. The task is to find a strictly

nonincreasing function that minimises the squared error to the input points in $S$. More formally, we find an estimate $\hat{f}$ by solving

$$\min \sum_{(\sigma,g(\sigma))\in S} \left(\hat{f}(\sigma) - g(\sigma)\right)^2,$$
$$\text{s.t.} \quad \hat{f}(\sigma_i) \geq \hat{f}(\sigma_j) \;\; \forall \sigma_i \leq \sigma_j.$$

In this paper we use the `isoreg` function of GNU R that represents $\hat{f}$ by a *piecewise constant* function (a step function). However, any other algorithm for finding monotonically non-increasing functions subject to squared error can be applied just as well. We point out that solving the regression problem is in general orders of magnitude faster than obtaining the set $S$, and is thus not a critical component from a complexity point of view.

### 5.2 Frequency curves in random data

We now study what frequency spectra look like in data that has no real structure. They can be used as a kind of 'null hypothesis' to compare real curves to: is there any structure in the dataset or not? For this we consider two types of random data: 1) constant background, i.e., each value occurs with fixed probability, 2) variable column marginals, i.e., each value in each column occurs with a given probability.

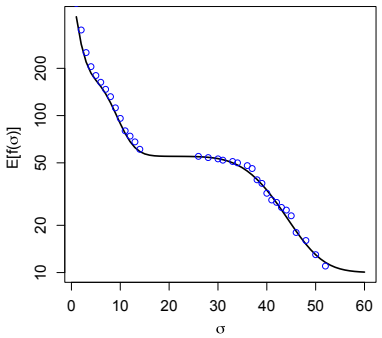


**Fig. 2.** Expected frequency curve in uniformly random data; $n = 1000$, $m = 10$, and $p = 0.2$. The points show observed frequencies in one instance of random data.

**Uniformly random data with constant background** We derive an analytic expression for the expected frequency spectrum for data that is uniformly random. Turns out that even under this simple model $f(\sigma)$ has non-trivial structure.

Let $\mathcal{D}$ denote a random binary dataset, with $n$ rows and $m$ attributes, where every item appears with probability $p$ in every row. The expected value of $f(\sigma, \mathcal{D})$ can be written as

$$\mathrm{E}_{\mathcal{D}}[f(\sigma,\mathcal{D})] = \sum_{l=1}^{m} \binom{m}{l} \sum_{k=\sigma}^{n} \mathrm{Binomial}(k; n, p^l). \tag{3}$$

The above equation follows from taking the expected value of $\sum_{X\subseteq\mathcal{I}} \mathrm{I}\{\mathrm{freq}(X) \geq \sigma\}$, and observing that the probability of the indicator function is given by the tail of a Binomial distribution with parameters $n$ and $p^l$.

Figure 2 shows an example of the expected frequency spectrum for uniformly random data with parameters $n = 1000$, $m = 100$, and $p = 0.2$. Perhaps somewhat surprisingly, the plot shows a clear "staircase" pattern, despite there not

being any structure in the data. The points in Fig. 2 are exact frequencies computed from a single instance of random data that matches the parameters. We can see that the expected curve closely matches the observed points.

**Uniformly random data with variable column marginals** Next, we consider the case when the data are generated by a model where the items are still all independent, but every item $i$ has its own occurrence probability $p_i$. In this case it is no longer straightforward to derive a closed form expression for $\mathrm{E}[f(\sigma)]$. However, we can in fact use the SPECTRA algorithm to estimate the expected frequency spectrum under this model as well. We point out that in this case, SPECTRA becomes a heuristic that seems to give useful results, but it does not necessarily converge to the correct expected pattern spectrum.

The algorithm is exactly the same as for real data, but we replace the support computation in SAMPLEPATH with *expected supports* defined for itemset $X$ as

$$\mathrm{E}_{\mathcal{D}}[\mathrm{freq}(X)] = \sum_{\mathcal{D}} \mathrm{freq}(X \mid \mathcal{D}) \Pr[\mathcal{D}] = n \prod_{i \in X} p_i,$$

where the expectation is taken over all possible datasets. Under the considered model this becomes the expected value of a Binomial distribution. For a given $\sigma$, the algorithm again starts from the empty itemset, and proceeds to sample a path by determining whether the expected support of an extension is larger than $\sigma$. As described in Section 5.1, we run SAMPLEPATH for a number of randomly selected values of $\sigma$, and fit an isotonic regression line through the points obtained.

## 6 Experiments

In this section we empirically evaluate the FASTEST and SPECTRA algorithms.

**Datasets:** For the experiments, we selected 14 moderately-sized datasets for which exact frequent itemset counting is still possible for reasonable frequency thresholds. The datasets were taken from the FIMI[5] (*Accidents*, *Kosarak*, and *Pumsbstar*) and LUCS-KDD[6] dataset repositories. Table 2 lists the dataset dimensions: the number of transactions and the total number of items.

**Evaluation criteria:** Primary evaluation criteria are 1) accuracy of the estimated counts, and 2) runtime. For purposes of presentation, all counts are given in logarithmic scale, i.e., $\log_{10}$. An additional reason is that we are primarily interested in correctly estimating the order of magnitude; in practice the difference between mining and processing 1 million or 1.1 million itemsets is negligible.

**Implementation:** FASTEST was implemented in C++ and is publicly available[7]. The implementation of the MCMC-based method was kindly provided by the authors of [6]. For exact counting, also used by the MCMC-based method, we use the original AFOPT implementation [15] taken from the FIMI repository. All experiments were executed single-threaded on a regular desktop computer (Intel i5-2500 @ 3.3GHz, 8GB RAM).

[5] `http://fimi.ua.ac.be/data/`
[6] `http://cgi.csc.liv.ac.uk/~frans/KDD/Software/`
[7] `http://patternsthatmatter.org/implementations/`

**Table 1.** Itemset counts for a fixed frequency threshold $\sigma$: comparing exact counting, FastEst (FE), and the MCMC-based method. Estimated counts are given with their 95% confidence intervals. The last column shows the fraction of frequency computations required by FastEst when compared to exact counting.

| | | Count [conf. interval] ($\log_{10}$) | | | Time (sec) | | | Frac. |
|---|---|---|---|---|---|---|---|---|
| Dataset | $\sigma$ | Exact | FE | MCMC | Exact | FE | MCMC | FE |
| *Adult* | 1 | 7.8 | 7.7 [7.5,7.9] | 7.8 [7.8,7.8] | 3 | 18 | 27 | 0.0245 |
| *Anneal* | 1 | 6.6 | 6.6 [6.4,6.8] | 6.8 [6.7,6.8] | 1 | 1 | 5 | 0.2291 |
| *Hepatitis* | 1 | 7.8 | 7.8 [7.7,7.9] | 7.9 [7.8,7.9] | 2 | 1 | 2 | 0.0146 |
| *Letrecog* | 1 | 8.8 | 8.7 [8.6,9.0] | 8.8 [8.8,8.9] | 23 | 12 | 51 | 0.0029 |
| *Mushroom* | 1 | 9.7 | 9.7 [9.6,9.9] | 10.0 [9.9,10.0] | 91 | 8 | 67 | 0.0005 |
| *Pendigits* | 1 | 8.7 | 8.7 [8.6,8.8] | 8.7 [8.6,8.7] | 21 | 8 | 39 | 0.0030 |
| *Waveform* | 1 | 10.1 | 10.1 [9.9,10.3] | 10.2 [10.1,10.3] | 331 | 5 | 53 | 0.0002 |
| *Accidents* | 4000 | 9.5 | 9.2 [8.5,9.7] | - | 496 | 475 | - | 0.0006 |
| *Chess* | 300 | 9.8 | 9.8 [9.5,10.1] | 9.8 [9.7,9.9] | 344 | 7 | 28 | 0.0002 |
| *Connect* | 7500 | 10.6 | 10.5 [10.2,10.8] | 10.7 [10.5,10.7] | 753 | 250 | 117 | 0.0000 |
| *Ionosphere* | 10 | 10.7 | 10.6 [10.5,10.8] | 11.5 [11.4,11.5] | 1231 | 1 | 146 | 0.0000 |
| *Kosarak* | 800 | 7.6 | 7.5 [6.7,7.9] | - | 15 | 206 | - | 0.1938 |
| *Mammals* | 10 | 12.2 | 12.3 [11.8,12.7] | 12.1 [12.0,12.2] | 43841 | 3 | 45 | 0.0000 |
| *Pumsbstar* | 7500 | 10.7 | 10.7 [10.6,10.8] | 8.5 [8.4,8.5] | 1042 | 320 | 554 | 0.0000 |

### 6.1 Estimating the number of frequent itemsets

We first evaluate how well the FastEst algorithm performs when estimating the number of frequent itemsets for a fixed threshold $\sigma$. We compare our method to both exact counting and the existing MCMC-based approach for approximate counting. Table 1 shows the results obtained on all 14 datasets. The results are split into two groups, according to the used frequency threshold: $\sigma = 1$ was used for the upper seven datasets, higher thresholds were used for the lower seven datasets to make sure that exact counting finished in reasonable time. Note that lower thresholds are no problem for FastEst, as we will see later.

The estimated counts are evaluated and compared using the following procedure. For FastEst, we first obtain a large population of 10000 path samples. We then compute the expected estimate and 95% confidence interval for 1000 path samples by subsampling the large pool 100 times. For MCMC we use a similar approach: the procedure is executed 100 times, and expected estimates and confidence intervals are based on taking a single sample from that pool. The presented runtimes for FastEst and MCMC match this procedure, that is, they are based on computing 1000 and 1 sample(s) respectively.

Looking at the upper half of the table first, we observe that the estimates by our method are spot on in expectation. There is clearly some variance between the estimates when using 1000 samples, but the 95% confidence intervals indicate that they are always in the same order of magnitude. The expected estimates obtained by MCMC are sometimes slightly off, but the variance is smaller. Except for *Adult* and *Anneal*, FastEst is always the fastest of the three methods.

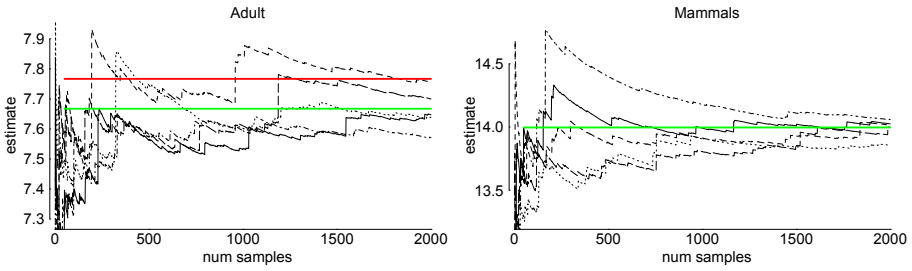


**Fig. 3.** Estimation stabilisation: the running estimate is updated after each new sample. Five independent runs of 2000 samples each, for *Adult* (left, $\sigma = 1$) and *Mammals* (right, $\sigma = 2$). Also indicated are the expected estimates (green) and the actual counts (red, only for *Adult*).

The results for the larger datasets in the lower half of the table show similar behaviour. The results for MCMC are missing for *Accidents* and *Kosarak* due to the implementation running out of memory. In general, the estimates by both FE and MCMC are accurate, although those for *Ionosphere* and *Pumbstar* by MCMC are clearly off. In terms of runtime, FastEst is the fastest in five out of seven cases. The gain in runtime is particularly large for *Chess*, *Ionosphere*, and *Mammals*, with attained speed-ups of 25-1230x.

Finally, the rightmost column of Table 1 shows the ratio of the number of the frequency computations required by FE, relative to the number needed by exact counting with Apriori [1]. Computing the frequency of an itemset is the expensive part of the algorithm, and requiring fewer such computations results in lower runtimes. The generally very low ratios confirm that FastEst needs very few frequency computations to obtain accurate estimates.

An important question we have not addressed so far is how many samples are needed to obtain accurate estimates. In the previous, we have shown that 1000 samples are generally enough, but it could be possible that fewer would be sufficient as well. To investigate this, consider the stabilisation plots in Figure 3. Although some small jumps in the estimates remain, they stabilise rather quickly. Taking into account the scales of the y-axes, we conclude that 1000 samples is more than enough to get at the correct order of magnitude. We observed similar behaviour for other datasets, and Table 1 also confirms this finding.

As a side note: computing the 10000 samples required for the *Mammals* plot, with $\sigma = 2$, took only 33 seconds. This demonstrates that FastEst can easily deal with lower frequency thresholds. Unfortunately, we were not able to produce the exact count for this threshold; we killed the process after it ran for two days.

**Closed frequent itemsets** We now present results obtained with our adaptation of FastEst for estimating the number of closed frequent itemsets, here denoted by C-FE. The results, all for $\sigma = 0.05|\mathcal{D}|$, are shown in the 'Count' and 'Time' columns in Table 2. We do not include confidence intervals for reasons of space, but observed similar intervals as previously. The exact results for *Chess* are missing because the AFOPT miner crashed.

**Table 2.** Dataset properties, closed itemset count estimation, and spectrum errors. $|\mathcal{D}|$ and $|\mathcal{I}|$ represent the number of transactions and items in a dataset, respectively. The next four columns contain the results for exact and FastEst (C-FE) closed itemset counting, with $\sigma = 0.05|\mathcal{D}|$. The rightmost columns contain curve errors obtained with Spectra and 1000 and 10000 samples (lower error is better).

| Dataset | | | Count ($\log_{10}$) | | Time (s) | | Spectra curve error | |
|---|---|---|---|---|---|---|---|---|
| Name | $\lvert\mathcal{D}\rvert$ | $\lvert\mathcal{I}\rvert$ | Exact | C-FE | Exact | C-FE | 1000 samples | 10000 samples |
| *Accidents* | 340183 | 468 | 7.81 | 7.49 | 27665 | 1168 | 0.68 [0.42,1.10] | 0.34 [0.24,0.48] |
| *Adult* | 48842 | 15 | 4.41 | 4.35 | 0 | 71 | 0.23 [0.16,0.35] | 0.12 [0.10,0.16] |
| *Anneal* | 898 | 71 | 3.55 | 3.91 | 0 | 2 | 0.24 [0.15,0.36] | 0.14 [0.10,0.18] |
| *Chess* | 3196 | 37 | - | 8.64 | - | 12 | 0.57 [0.38,0.81] | 0.30 [0.23,0.39] |
| *Connect* | 67557 | 43 | 7.45 | 10.09 | 3324 | 1301 | 0.54 [0.37,0.79] | 0.29 [0.22,0.42] |
| *Hepatitis* | 155 | 20 | 5.18 | 5.17 | 2 | 1 | 0.28 [0.20,0.40] | 0.14 [0.11,0.18] |
| *Ionosphere* | 351 | 35 | 7.45 | 8.42 | 40974 | 4 | 0.74 [0.50,1.13] | 0.40 [0.30,0.53] |
| *Kosarak* | 990002 | 41270 | 1.52 | 1.52 | 0 | 51 | 0.11 [0.07,0.15] | 0.05 [0.04,0.07] |
| *Letrecog* | 20000 | 17 | 4.48 | 4.54 | 1 | 14 | 0.35 [0.22,0.58] | 0.15 [0.11,0.22] |
| *Mammals* | 2183 | 121 | 7.61 | 7.71 | 5722 | 3 | 0.47 [0.34,0.65] | 0.23 [0.18,0.28] |
| *Mushroom* | 8124 | 23 | 4.11 | 4.53 | 0 | 17 | 0.39 [0.28,0.55] | 0.20 [0.15,0.26] |
| *Pendigits* | 10992 | 17 | 3.76 | 3.74 | 0 | 8 | 0.25 [0.18,0.37] | 0.13 [0.11,0.15] |
| *Pumsbstar* | 49046 | 2088 | 6.97 | 10.63 | 1609 | 805 | 0.44 [0.30,0.64] | 0.22 [0.18,0.27] |
| *Waveform* | 5000 | 22 | 5.59 | 5.53 | 3 | 5 | 0.42 [0.28,0.64] | 0.22 [0.16,0.27] |

The estimates are pretty accurate for most datasets, but there are some exceptions. Of these, *Connect* is the most obvious: the estimate is three orders of magnitude off. Investigating this in more detail, it turns out that this is due to the extreme ratio between the number of frequent and closed itemsets: with only 1000 samples, the estimated correction coefficient cannot be reliable if it is much lower than 0.1. If we increase the number of samples for *Connect* to 2000, for example, the estimate becomes 9.2 – already one order of magnitude better.

Closed frequent itemset mining is a much harder problem than frequent itemset mining, as is also reflected by the runtimes. The exact miner is faster in eight cases, but those are the relatively easy datasets. The exact counting runtimes explode for the more difficult datasets, whereas C-FE is relatively fast.

### 6.2 Estimating the pattern frequency spectrum

We now turn our focus to the Spectra algorithm. That is, we estimate the number of frequent itemsets for the complete frequency ranges. For each dataset, we obtain a curve using both 1000 and 10000 samples, and compare the resulting curves to the exact curves as far as they are available, i.e., for the frequency range $[\sigma, |\mathcal{D}|]$ (for values of $\sigma$ mentioned in Table 1).

We initially computed the mean error over the complete curve, but since the spectra are quite accurate these were hardly informative. We therefore turned to another measure, i.e., the curve errors presented in the rightmost columns of Table 2. These numbers are the 95% quantiles of the errors: 95% of the curve has an error that is at most as large as indicated. For all datasets, 1000 samples are

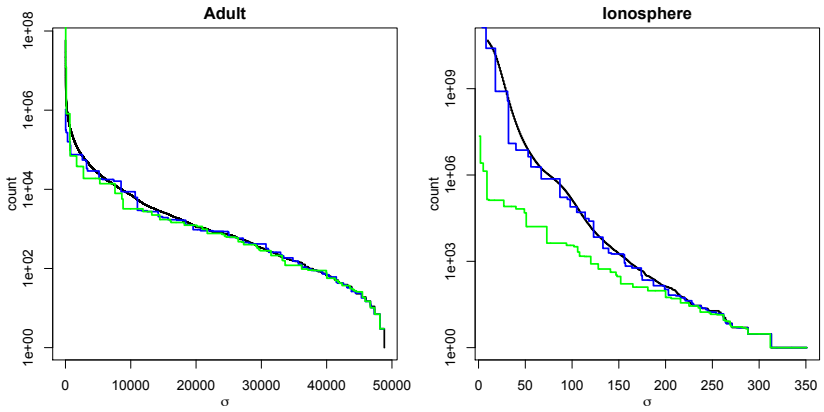


**Fig. 4.** Frequency curves obtained with SPECTRA for *Adult* and *Ionosphere*: estimated on actual data (blue) and expected in random data (green). For comparison, the exact curve (black) is also drawn.

enough to estimate at least 95% of the *full spectrum* with an error of less than one order of magnitude, and often much less. With 10000 samples, 95% of the curve is often only 0.1 - 0.2 away from the actual counts. Of course, the estimates are inevitably less accurate than when using the same number of samples for a single $\sigma$, but nonetheless SPECTRA succeeds in obtaining accurate estimates for the *whole frequency range* with a limited number of samples.

The runtimes of SPECTRA are not listed, but they are even shorter than when estimating the number of itemsets for a single, low threshold. The reason is that samples are drawn for arbitrary values of $\sigma$, and runtimes are shorter for higher values. When 1000 samples are used, computing a curve takes only seconds for the smaller datasets, and a few minutes for the larger ones. The longest runtime was measured for *Accidents*: 401 seconds.

Figure 4 shows example frequency spectra obtained for *Adult* and *Ionosphere*. The estimated curves (in blue) clearly match the actual curves (in black) very well, although there is a slight tendency towards underestimation for the lower frequency ranges. It is very easy to read out ballpark pattern counts and use these to tune the frequency threshold. For *Ionosphere*, for example, choosing $\sigma = 80$ would result in a modest $10^6$ frequent itemsets, whereas anything lower will quickly get you vast amounts of patterns.

**Comparing against random frequency spectra** The plots in Figure 4 also show expected spectra obtained with the procedure described in Section 5.2. That is, we consider the model where the items are independent but variable column marginals are given. The number of transactions and item probabilities of the random data are equal to those of the real dataset, and high-confidence random frequency spectra are obtained by obtaining 10 samples for each possible value of $\sigma$; this procedure is extremely fast and can be done in one or two seconds for any dataset considered in this paper.

Comparing the actual to the random spectra, we observe that the itemset frequencies of *Adult* can be mostly explained by the marginal probabilities of the individual items. Still, there is some structure present in the lower part of the frequency range, where more itemsets are found than expected. For *Ionosphere*, the picture is rather different: this dataset has clearly much more structure than can be explained from the individual item probabilities.

## 7 Discussion

The results demonstrate that our methods for estimating the number of frequent patterns perform very well. Still, there are many possibilities for future research. One obvious direction is to investigate the adaptation for closed itemsets in more detail. One task would be to make it reliable for any dataset, but more interesting is to investigate the approach for other non-monotonic constraints.

The estimated frequency spectra can be reliably used to choose a minimum support threshold for frequent itemset mining; the order of magnitude is always right. Furthermore, these spectra are potentially useful as 'fingerprints' of the data. For example, we witnessed rather different shapes and curves, and we can imagine that it might be possible to cluster datasets based on their frequency spectra. Also, more advanced models could be used for the generation of expected curves, to see whether the actual curves match those.

We only considered frequency spectra for complete datasets, but one could also consider subsets of the pattern language. For example, by only considering those patterns that are supersets of a given 'query' pattern. This would give query-based frequency spectra, which could inform us about the local structure for a given query. The sampling procedure would remain almost unchanged: given a query, each path is sampled starting from that query instead of the empty set.

Note that the FastEst algorithm provides estimates for each individual depth in the search tree, i.e., for each itemset size. This implies that our method could be used to estimate, e.g., the number of n-grams in a document dataset.

Finally, our approach can be easily parallelised to make it run efficiently on very large datasets. When sampling a path, we must compute the frequency of a pattern several times. We can easily adapt the algorithm to make it an Apriori-style algorithm, which has the advantage of being database friendly. Then, the database scan can be efficiently implemented on, for example, relational databases or distributed platforms for large-scale data processing.

## 8 Conclusions

We introduced two methods for approximate counting of the number of frequent patterns in a given dataset. Based on Knuth's classical algorithm for estimating the size of a search tree, the FastEst algorithm estimates the number of frequent itemsets for a given frequency threshold. The Spectra algorithm combines FastEst with isotonic regression to estimate the complete pattern frequency spectrum for a given dataset.

The experiments showed that both methods are very accurate and efficient, when compared to exact counting and an existing MCMC-based estimator. The adaptation for closed itemsets gives good estimates in most cases. Finally, we showed how pattern frequency spectra provide interesting summary statistics that can be compared to expected curves generated for random data models.

**Acknowledgements.** Matthijs van Leeuwen is supported by a Post-doctoral Fellowship of the Research Foundation Flanders (FWO).

## References

1. R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of VLDB'94*, pages 487–499, 1994.
2. R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of ICDE'95*, pages 3–14, 1995.
3. R.E. Barlow and H.D. Brunk. The isotonic regression problem and its dual. *Journal of the American Statistical Association*, 67(337):140–147.
4. T. De Bie. Maximum entropy models and subjective interestingness: an application to tiles in binary databases. *Data Min. Knowl. Discov.*, 23(3):407–446, 2011.
5. M. Boley, T. Gärtner, and H. Grosskreutz. Formal concept sampling for counting and threshold-free local pattern mining. In *Proc. of SDM'10*, pages 177–188, 2010.
6. M. Boley and H. Grosskreutz. A randomized approach for approximating the number of frequent sets. In *Proceedings of ICDM'08*, pages 43–52, 2008.
7. B. Bringmann, S. Nijssen, N. Tatti, J. Vreeken, and A. Zimmermann. Mining sets of patterns: Next generation pattern mining. In *Tutorial at ICDM'11*, 2011.
8. P.C. Chen. Heuristic sampling: A method for predicting the performance of tree searching programs. *SIAM Journal on Computing*, 21(2):295–315, 1992.
9. H. Cheng, X. Yan, J. Han, and C.-W. Hsu. Discriminative frequent pattern analysis for effective classification. In *Proceedings of the ICDE*, pages 716–725, 2007.
10. B. Goethals, S. Moens, and J. Vreeken. MIME: a framework for interactive visual pattern mining. In *Proceedings of KDD'11*, pages 757–760, 2011.
11. D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, and R.S. Sharm. Discovering all most specific sentences. *ACM Trans. Database Syst.*, 28(2):140–174, 2003.
12. M. Jerrum and A. Sinclair. The markov chain monte carlo method: an approach to approximate counting and integration. *Approximation algorithms for NP-hard problems*, pages 482–520, 1996.
13. P. Kilby, J.K. Slaney, S. Thiébaux, and T. Walsh. Estimating search tree size. In *Proceedings of AAAI'06*, pages 1014–1019, 2006.
14. D.E. Knuth. Estimating the efficiency of backtrack programs. *Mathematics of computation*, 29(129):122–136, 1975.
15. G. Liu, H. Lu, J.X. Yu, W. Wang, and X. Xiao. AFOPT: An efficient implementation of pattern growth approach. In *Proc. of FIMI at ICDM'03*, 2003.
16. P.W. Purdom. Tree size by partial backtracking. *SIAM Journal on Computing*, 7(4):481–491, 1978.
17. J. Vreeken, M. van Leeuwen, and A. Siebes. Krimp: mining itemsets that compress. *Data Min. Knowl. Discov.*, 23(1):169–214, 2011.
18. X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *Proceedings of ICDM'02*, pages 721–724, 2002.
19. G. Yang. The complexity of mining maximal frequent itemsets and maximal frequent patterns. In *Proceedings of KDD'04*, pages 344–353, 2004.