Providing Concise Database Covers Instantly by Recursive Tile Sampling

Sandy Moens[1,2], Mario Boley[2,3], and Bart Goethals[1]

[1] University of Antwerp, Belgium, firstname.lastname@uantwerpen.be,
[2] University of Bonn, Germany, firstname.lastname@uni-bonn.de,
[3] Fraunhofer IAIS, Germany, firstname.lastname@iais.fgh.de

Abstract. Known pattern discovery algorithms for finding tilings (covers of 0/1-databases consisting of 1-rectangles) cannot be integrated in instant and interactive KD tools, because they do not satisfy at least one of two key requirements: a) to provide results within a short response time of only a few seconds and b) to return a concise set of patterns with only a few elements that nevertheless covers a large fraction of the input database. In this paper we present a novel randomized algorithm that works well under these requirements. It is based on the recursive application of a simple tile sample procedure that can be implemented efficiently using rejection sampling. While, as we analyse, the theoretical solution distribution can be weak in the worst case, the approach performs very well in practice and outperforms previous sampling as well as deterministic algorithms.

Keywords: Instant Pattern Mining, Sampling Closed Itemsets, Tiling Databases

1 Introduction

Recently, data mining tools for interactive exploration of data have attracted increased research attention [7, 9, 11, 19]. For such an interactive exploration process, a tight coupling between user and system is desired [4, 17], i.e., the user should be allowed to pose and refine queries at any moment in time and the system should respond to these queries instantly [16]. This allows to transfer subjective knowledge and interestingness better as opposed to a batch setting with high computational overhead [4, 17].

When finding collections of patterns that cover large fractions of the input data, existing techniques often fail to deliver the requirement of instant results. The reason is that they iteratively find the best pattern that covers the remaining data [10, 13], which involves an NP-hard problem [10]. Another approach employed in the literature is first enumerating a large collection of patterns and then selecting distinct patterns that optimize the quality [20]. The large bottleneck for such procedures is the enumeration of many patterns.

In this paper, we address the issue of finding patterns that have good descriptive capabilities, i.e., they individually describe a substantial amount of data,

and that can together be used to describe the data. Such collections are helpful in exploratory data mining processes to get a quick overview of the prominent structures in the data [20]. Then, iteratively a user can drill down to specific parts of the data to explore even further. To this end, we propose a randomized procedure to quickly find small collections of patterns consisting of large tiles. Our method is based on a recursive sampling scheme that selects individual cells in a conditional database. The sampling process is based on a heuristic computation reflecting the potential of a cell for being part of a large pattern.

In summary, the contributions of our work are:

- We introduce a sampling method for finding database covers in binary data with near instants results in Section 3. Our sampling method heuristically optimizes the area of individual tiles using a recursive extension process.
- We introduce a new measure for evaluating pattern collections specifically in interactive systems which ensures the total representativeness of a pattern collection while guaranteeing the individual quality of patterns.
- We evaluate our novel sampling method with respect to the proposed measure in a real-time setting, in which algorithms are given only a short time budget of one second to produce results. We compare to state-of-the-art techniques and show that our method outperforms these techniques.

2 Preliminaries

In this paper we consider binary databases, as in itemset mining and formal concept analysis. A formal context is a triple $(O, A, \mathcal{R})$ with a set of **objects** O, a set of **attributes** A and a binary **relation** or database defined between the objects and attributes $\mathcal{R} \subseteq O \times A$. We use o_k and a_l as mappings to individual objects and attributes from the sets. Two Galois operators are defined as $O[X] = \{o \in O : \forall a \in X, (o, a) \in \mathcal{R}\}$ and $A[Y] = \{a \in A : \forall o \in Y, (o, a) \in \mathcal{R}\}$. $O[X]$ is also called the **cover** $cov(X)$. Applying both Galois operators sequentially, yields two closures operators, $\tilde{o}[.] = O[A[.]]$ and $\tilde{a}[.] = A[O[.]]$.

The binary relation is essentially a binary matrix $\{0,1\}^{|O|\times|A|}$ such that a region consisting of only 1's is called a **tile** $\mathcal{T} = (X, Y)$ [10]. A tile can be adopted directly to transactional databases as an itemset X and its corresponding cover Y, such that $\forall a \in X, \forall o \in Y : (o, a) \in \mathcal{R}$. A tile is said to support a **cell** $(k, l) \in \mathcal{R}$ if $o_k \in Y$ and $a_l \in X$. In this work we are interested specifically in formal concepts (also maximal tiles or closed itemsets), such that $X = A[Y]$ and $Y = O[X]$. The interestingness of a tile is defined by its **area** in the data $area(\mathcal{T}) = |X| \cdot |Y|$. Note that the area of a tile is neither monotonic nor anti-monotonic. Hence, typical enumeration strategies [21] can not be used directly.

Geerts et al. [10] developed an algorithm for mining all tiles with at least a given area by adopting Eclat [21]. Using an upper bound on the maximum area of tiles that can still be generated, they prune single attributes during the mining process. Given a set of attributes X and a test attribute a, they count the number of objects $o \in O[X]$ such that $(o, a) \in \mathcal{R}$ and $|A[\{o\}]| \geq \ell$. Denote this count as $count_{\geq \ell}(X, a)$. The upper bound on the area of a tile with $|X| = s$

is given by $s \cdot count_{\geq s}$. The total **upper bound** over all sizes and possible tiles is obtained by taking the maximum:

$$UB_{X \cup \{a\}} = \arg \max_{\ell \in |X|+1,\ldots} (\ell \cdot count_{\geq \ell}(X, a)). \tag{1}$$

A database **tiling** is a collection of possibly overlapping tiles [10]. The problem statement we consider, is finding a tiling covering as much of the data as possible with only few patterns, in short time budgets. Therefore, each pattern should individually have large area. We stress that our setting is new [7, 9, 11] and existing measures do not satisfy these requirements. Given a collection of tiles $\mathcal{F}$, then the quality combines total collection and individual pattern qualities:

$$qual(\mathcal{F}) = \frac{cov(\mathcal{F})}{|\mathcal{R}|} \cdot \frac{1}{|\mathcal{F}|} \sum_{\mathcal{T} \in \mathcal{F}} \frac{area(\mathcal{T})}{|\mathcal{R}|}, \tag{2}$$

with $cov(\mathcal{F})$ the total number of cells covered. If known, for comparison purposes the normalization for area can be replaced by the area of the largest tile in the data $\mathcal{T}_{larg}$. Note that this measure indirectly favors small pattern collections.

3 Biased Sampling of Large Tiles

In this section we introduce a sampling procedure on individual cells from a conditional database. Our method heuristically optimizes the area of tiles.

3.1 Sampling Individual Cells

The upper bound from Equation (1) is good but intensive to compute. We propose to use a less intensive bound to guide the search for large area patterns. Consider a cell $(k, l) \in \mathcal{R}$, corresponding to object o_k and attribute a_l. If $(o_k, a_l) \in \mathcal{R}$, the maximum area of a tile having this cell is given by the **maximum upper bound** $MB(k, l) = |O[\{a_l\}]| \cdot |A[\{o_k\}]|$, where the first part is called the **column marginal** M^A, and the second part the **row marginal** M^O.

Proposition 1. *Given a cell (k, l) in a dataset, $MB(k, l)$ is never less than the true area of a tile containing the cell.*

Proof. Suppose $\mathcal{T} = (X, Y)$ contains cell (k, l) but has area greater than $MB(k, l)$. Then $\mathcal{T}$ either (1) contains object $o_i \notin O[\{a_l\}]$, or (2) contains attribute $a_j \notin A[\{o_k\}]$. Since the cover of $\mathcal{T}$ is an intersection relation over the covers of the attributes, (1) is not possible. Suppose (2) holds, then o_k will never be part of the cover of $\mathcal{T}$, which is in contrast with the original assumption. □

We can use MB as probabilities for sampling individual cells from a binary database by using them as probabilities for sampling a single cell from the data:

$$\Pr(k, l) = \frac{MB(k, l)}{Z} = \frac{|O[\{a_l\}]| \cdot |A[\{o_k\}]|}{Z}. \tag{3}$$

The heuristic behind this sampling scheme is that a cell which is part of tiles with large area, also has a larger sampling probability. Using the Equation 3 we sample individual cells having higher chance of being in a large area tile.

Algorithm 1 RTS($\mathcal{R}, S$)

Require : relation $\mathcal{R}$, current state S
Return : collection of large tiles

1: $(i, j) \sim \Pr(k, l, S)$
2: $S \leftarrow (S^A \cup \{a_j\}, S^O \cup \{o_i\})$
3: $Tiles \leftarrow \{\mathcal{T}_V, \mathcal{T}_H\}$ // Closures of S
4: **if** $A(S^O) \setminus S^A \neq \phi$ and $O(S^A) \setminus S^O \neq \phi$ **then**
5: $\quad Tiles \leftarrow Tiles \cup RTS(\mathcal{R}, S)$
6: **end if**
7: **return** $Tiles$

3.2 Recursive Tile Sampling

Large tiles can be sampled by repeatedly sampling individual cells that consitute a tile. As such, instead of sampling cells independently, we restrict the sampling of new cells, to those that definitely form a tile. We do so by considering only the conditional database constructed by the previous samples.

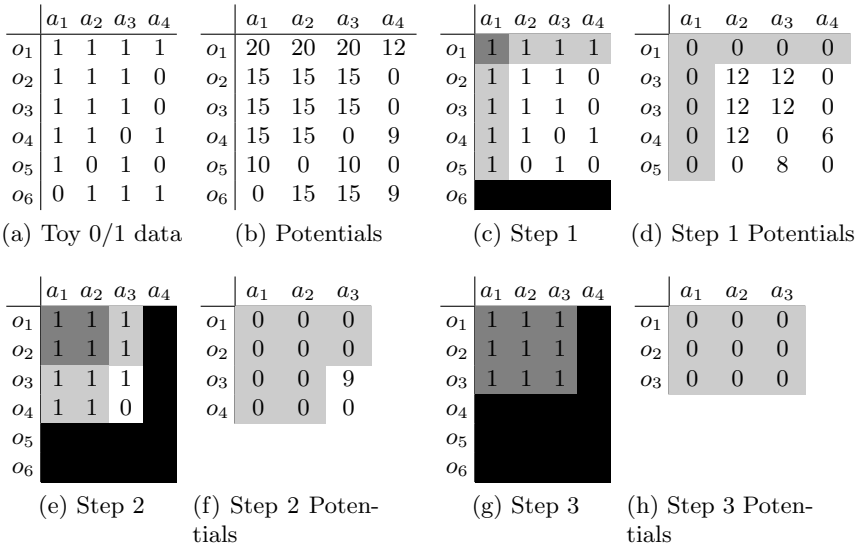
We define a **current state** S by a set of current attributes S^A and a set of current objects S^O (in fact a current state is an intermediate tile). The sampling probabilities from Equation 3 can be updated to incorporate the previous knowledge reflected by the current state. The new probabilities become

$$\Pr(k, l, S) = \frac{|O[\{a_l\}] \cap O[S^A]| \cdot |A[\{o_k\}] \cap A[S^O]|}{Z}, \tag{4}$$

with Z a normalization constant over remaining cells. The intersections construct the conditional database $\mathcal{R}^S$. As we condition on the current state we know that an object in $O[S^A]$ or an attribute in $A[S^O]$ contains only ones. We therefore remove these cells from the conditional database.

Pseudo code for **Recursive Tile Sampling**, RTS, is given in Algorithm 1. The sampling procedure extends a current state in two directions simultaneously (see Figure 1). When either no attributes or no objects remain, the algorithm stops. Therefore, the current state S itself, never represents a rectangular tile. As such, each time when constructing the conditional database we report a pair of tiles defined by the Galois and closure operators on S: the tile containing all remaining attributes is $\mathcal{T}_H = (A[S^O], \tilde{o}[S^O])$ and the tile containing all remaining objects is $\mathcal{T}_V = (\tilde{a}[S^A], O[S^A])$. $\mathcal{T}_H$ is also called a horizontal extension tile and $\tilde{o}[S^O] \supseteq S^O$. Likewise, $\mathcal{T}_V$ is also called a vertical extension tile and $\tilde{a}[S^A] \supseteq S^A$. In the end we return all tiles that are found during the recursive steps.

Example 1. Given the toy dataset from Figure 1a, the initial unnormalized sampling probabilities (or potentials) are given in Figure 1b. First cell (1,1) is sampled and object o_6 is removed because $(o_6, a_1) \notin \mathcal{R}$. Two extension tiles are formed by applying the closure operators: $\mathcal{T}_{H_1} = (\{a_1, a_2, a_3, a_4\}, \{o_1\})$ and $\mathcal{T}_{V_1} = (\{a_1\}, \{o_1, o_2, o_3, o_4, o_5\})$. The recursive step is applied to the database



	a_1	a_2	a_3	a_4
o_1	1	1	1	1
o_2	1	1	1	0
o_3	1	1	1	0
o_4	1	1	0	1
o_5	1	0	1	0
o_6	0	1	1	1

(a) Toy 0/1 data

	a_1	a_2	a_3	a_4
o_1	20	20	20	12
o_2	15	15	15	0
o_3	15	15	15	0
o_4	15	15	0	9
o_5	10	0	10	0
o_6	0	15	15	9

(b) Potentials

	a_1	a_2	a_3	a_4
o_1	1	1	1	1
o_2	1	1	1	0
o_3	1	1	1	0
o_4	1	1	0	1
o_5	1	0	1	0
o_6				

(c) Step 1

	a_1	a_2	a_3	a_4
o_1	0	0	0	0
o_3	0	12	12	0
o_3	0	12	12	0
o_4	0	12	0	6
o_5	0	0	8	0

(d) Step 1 Potentials

	a_1	a_2	a_3	a_4
o_1	1	1	1	
o_2	1	1	1	
o_3	1	1	1	
o_4	1	1	0	
o_5				
o_6				

(e) Step 2

	a_1	a_2	a_3
o_1	0	0	0
o_2	0	0	0
o_3	0	0	9
o_4	0	0	0

(f) Step 2 Potentials

	a_1	a_2	a_3	a_4
o_1	1	1	1	
o_2	1	1	1	
o_3	1	1	1	
o_4				
o_5				
o_6				

(g) Step 3

	a_1	a_2	a_3
o_1	0	0	0
o_2	0	0	0
o_3	0	0	0

(h) Step 3 Potentials

Fig. 1: Running example of RTS: dark gray = S, light gray = cells without sampling probability, black = cells that are not extensions of S.

excluding attributes $a_i \notin A[\{o_1\}]$ and objects $o_j \notin O[\{a_1\}]$. In step 2, cell (2,2) is sampled and $\mathcal{T}_{H_2} = (\{a_1, a_2, a_3\}, \{o_1, o_2, o_3\})$ and $\mathcal{T}_{V_2} = (\{a_1, a_2\}, \{o_1, o_2, o_3, o_4\}))$ are reported. In the last step, $(3, 3)$ is sampled and the last extension tiles are found: $\mathcal{T}_{H_3} = \mathcal{T}_{V_3} = (\{a_1, a_2, a_3\}, \{o_1, o_2, o_3, \})$. The process then stops as no recursion can be applied.

3.3 Efficient Sampling of Individual Cells

The distribution $f(k, l, S)$, $f(x)$ for simplicity, defined by Equation 4 is

$$f(x) = \begin{cases} MB(k, l, S)/Z & \text{if } (k, l) \in \mathcal{R}^S, t_k \notin S^O, i_l \notin S^A \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

We show how to sample efficiently from $f(x)$ without completely materializing it. We can use rejection sampling with non-uniform distributions for this purpose. Rejection sampling is a general method for sampling from a probability distribution f by sampling from an auxiliary distribution g that acts as an envelope. It uses the fact that we can sample uniformly from the density of the area under the curve $cg(x)$ [15], $c > 1$. Samples from cg that are generated outside of the density region of $f' = \beta f$ are then rejected, $\beta > 1$. Formally, we have $f'(x) < cg(x)$ and acceptance probability $\alpha \leq f'(x)/cg(x)$, with $\alpha \sim Unif[0, 1]$.

To use rejection sampling we first set f' to the unnormalized version of the true distribution f, using the same condition $(i, j) \in \mathcal{R}$ and $\beta = Z$. As auxiliary

distribution we choose $g(x) = (|O[\{a_l\}] \cap O[S^A]| \cdot |A[\{o_k\}] \cap A[S^O]|)/Z$, without the condition on the presence of (k, l). Setting $c = Z$ we obtain two rules

$$f'(x) = \begin{cases} cg(x) & \text{if } (k,l) \in \mathcal{R} \\ 0 \leq cg(x) & \text{if } (k,l) \notin \mathcal{R}, \end{cases} \tag{6}$$

such that by construction the acceptance probability boils down to accepting if the cell is present, rejecting if not.

What is left is obtaining samples from g. Since this distribution is based on the independence of rows and columns we sample a_l with probability $\Pr_A(l, S) = |O[\{a_l\}] \cap O[S^A]|/Z$ and o_k with probability $\Pr_O(k, S) = |A[\{o_k\}] \cap A[S^O]|/Z$ independently. This results in exact samples from g.

3.4 Incorporating Knowledge

Our sampling method can integrate prior knowledge by assigning weights to cells instead of marginal counts. Suppose a weight function $w : (k, l) \to [0, 1]$, then we have marginals $M^O(o_k) = \sum_{a_i \in A[\{o_k\}]} w(k, i)$ and $M^A(a_l) = \sum_{o_j \in O[\{a_l\}]} w(j, l)$. and potentials $\Pr(k, l) = (\sum_{a_i \in A[\{o_k\}]} w(k, i) \cdot \sum_{o_j \in O[\{a_l\}]} w(j, l))/Z$, with Z a normalization constant to obtain a probability distribution. For the weights we propose the use of multiplicative weights [12] based on the number of times a cell has already been covered by previous knowledge, i.e., $w(k, l) = \gamma^m$, with m the number of times a cell has been covered and γ a discounting factor.

The main bottleneck of this weighting scheme is the computation of individual marginals. When explicitly storing the previous marginal, we can efficiently update them. Suppose for each of the attributes and objects we keep the current marginals in memory. Given that new knowledge is provided in the form of a tile $\mathcal{T}_n = (X, Y)$, then only marginals of attributes and objects supporting cells described by $\mathcal{T}_n$ have to be updated. Using the following scheme over the individual cells of the tile we obtain fast incorporation of knowledge:

$$\begin{gathered} \forall a_l \in X, \forall o_k \in Y \\ \to \\ M^A(a_l) = M^A(a_l) - w_{old}(k, l) + w_{new}(k, l) \\ M^O(o_k) = M^O(o_k) - w_{old}(k, l) + w_{new}(k, l). \end{gathered}$$

This results in updates in $\mathcal{O}(2|X||Y|)$ time rather than an update for the complete database in $\mathcal{O}(2|A||O|)$ time. The factor 2 comes from the fact that we have to update the margins for attributes as well as for objects.

3.5 Worst Case Analysis

Individual Cell Sampling We analyze the worst case performance of our sampling probabilities wrt $area^2$, which is the distribution that is closest to the distribution simulated by our sampling method. We define the worst case

scenario as a dataset where the sampling probabilities do not reflect the $area^2$ of sampled closed tiles. In our framework this happens when $\{0,1\}^{n\times n}$ contains all ones except on the diagonal. Then, the number of closed tiles equals 2^n-2 and the largest tile in the region has area $\lceil n/2\rceil\lfloor n/2\rfloor$. The total sampling potential is $n(n-1)^3$, because each row and column has a count of $(n-1)$ and there are $n(n-1)$ non-zero entries. The true $area^2$ mass induced by all closed tiles equals $\sum_{k=1,\cdots,n-1}\binom{n}{k}(k(n-k-1))^2 \geq n(n-1)^3$ and holds for all $n \geq 2$. Moreover for $n \geq 4$ it holds that the second parts is strictly smaller and we have a constant undersampling of this density region.

Rejection Sampling We first analyze the general sampling complexity and then the worst-case time complexity when many samples are rejected.

Using Section 3.3 we sample independently one column and one row. The materialization of the marginal distributions has $\mathcal{O}(|A|+|O|)$ time and space complexity. (A naïve direct approach for sampling relies on full materialization of the matrix and therefore has time and space complexity $\mathcal{O}(|A||O|)$) For the time complexity we also have to take into account the time for sampling one element. This can be achieved in logarithmic time using a binary seach. The total sampling time with rejection sampling becomes $\mathcal{O}((|A|+\log|A|)+(|O|+\log|O|))$ compared to $\mathcal{O}(|A||O|)+(\log|A|\log|O|)$ for the direct approach.

This does not yet conclude our time analysis as we did not yet take into account the number of times we have to resample due to rejections. We use as basis the worst-case scenario for rejection sampling, which is the setting where the binary representation of the data is an identity matrix I_n of size $n\times n$. The marginal probabilities equal $1/n$ and the probability of sampling a single 1-valued cell equals $1/n^2$. Since the data contains exactly n ones, the total probability of sampling a valid cell with rejection sampling, and thus accepting the sample, equals $n\cdot 1/n^2 = 1/n$. Setting $|\mathcal{R}|$ to n we obtain a total time sampling complexity for rejection sampling equal to $\mathcal{O}((|A|+|O|)+(\log|A|+\log|O|)\cdot|\mathcal{R}|)$. Note that the first term is the time for materializing the distribution and has to be done just once. The last part is the time for sampling the distribution, which has to be repeated in worst case $|\mathcal{R}|$ times.

4 Experiments

We experimented with our sampling method on several real world datasets shown in Table 1 together with their main characteristics. Pumsb, Connect and Acc are publicly available from the FIMI Repository [2]. Adult, Cens-Inc, CovType and Pokhand are made available at the UCI Machine Learning Repository [1].

For the experiments our interests are two-fold:

- How well does RTS work in real instant conditions?
- How does the quality of patterns evolve over time?

The first topic is related to intensive interactive environments: a user is exploring data and is assisted by several pattern mining algorithms. The user

$\mathcal{R}$	$\lvert O\rvert$	$\lvert A\rvert$	RTS		CDPS$_{area*fq^2}$		Asso		Tileminer$_{first}$		Tileminer$_{unif}$	
			quality	$\lvert\mathcal{F}\rvert$	quality	$\lvert\mathcal{F}\rvert$	quality	$\lvert\mathcal{F}\rvert$	quality	$\lvert\mathcal{F}\rvert$	quality	$\lvert\mathcal{F}\rvert$
Adult	48,842	97	**0.64**	25.0	0.40	24.8	0.58	7.0	0.55	25.0	0.61	25.0
Pumsb	49,046	2.113	**0.31**	25.0	0.00	1.10	0.00	0.0	0.00	0.0	0.00	0.0
Connect	67,557	129	**0.47**	25.0	0.09	25.0	0.00	0.0	0.00	25.0	0.00	25.0
Cens-Inc	199,523	519	**0.35**	12.7	0.03	25.0	0.00	0.0	0.04	25.0	0.04	25.0
Acc	340,183	468	**0.18**	10.5	0.00	0.3	0.00	0.0	0.00	0.0	0.00	0.0
CovType	581,012	5.858	0.28	3.0	0.33	24.5	0.00	0.0	**0.45**	14.0	**0.45**	14.0
Pokhand	1,000,000	95	**0.09**	9.2	0.02	5.1	0.00	0.0	0.00	0.0	0.00	0.0

Table 1: Characteristics for different datasets, quality of pattern collections and the number of patterns in the collection obtained in 1 second.

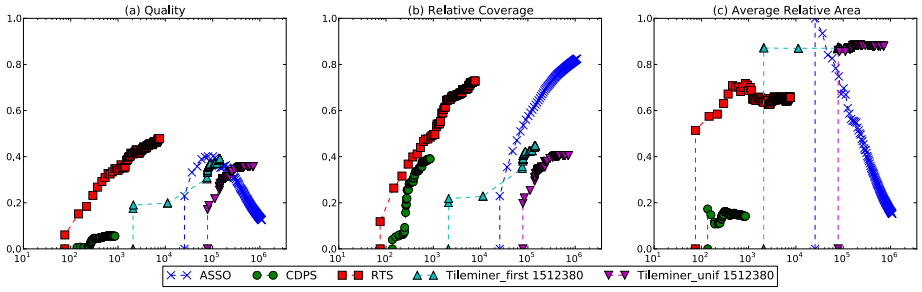


Fig. 2: Quality of 100 patterns over time in millisecond (log scale) for Cens-Inc.

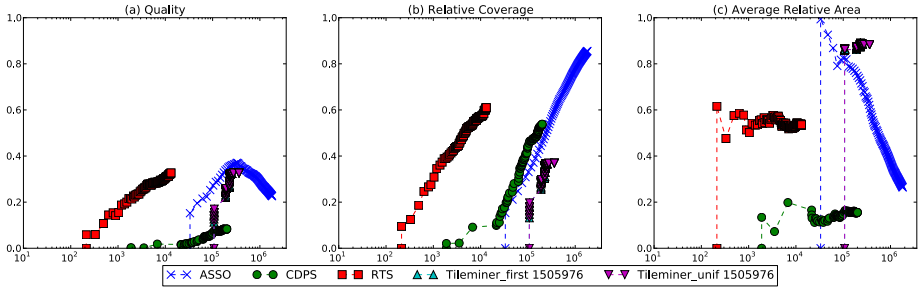


Fig. 3: Quality of 100 patterns over time in millisecond (log scale) for Acc.

hereby expects instant condense results. The second topic is related also to an interactive environment with more relaxed conditions. For instance, instead of having instant results, a user is willing to wait up to 1 minute. Due to space limitations, we do not show our evaluation for the incorporation of knowledge.

For the experiments we compared our method to three other techniques for finding tiles. As deterministic baseline we used a boolean matrix factorization algorithm called Asso [13], which greedily optimizes the coverage of k basis vectors. Asso is implemented in C++. We used a Tileminer implementation that finds all tiles given a minimum area, which can be seen as a baseline for finding large area tiles. Tileminer is also written in C++. We use Controlled

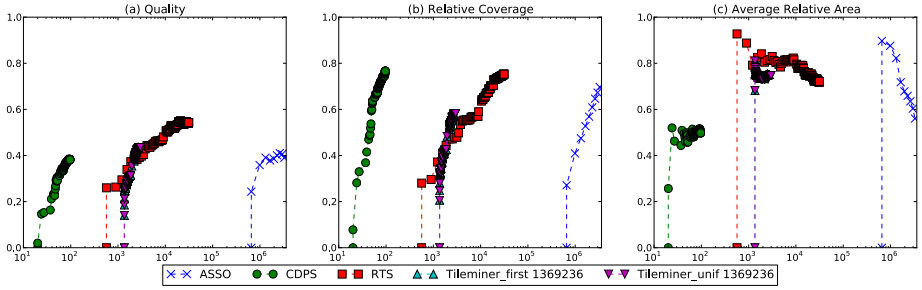


Fig. 4: Quality of 100 patterns over time in millisecond (log scale) for CovType.

Direct Pattern Sampling [8] (CDPS), written in Java, as alternative sampling method and as baseline for a fast anytime sampling technique. CDPS provides i.i.d. samples in rapid succession from a user customizable distribution over the complete pattern space. At last, RTS[4] is implemented in Java. In our experiments we reported a single pattern for each recursive run, which is the largest area pattern produced. The tests themselves are executed on our local server running Ubuntu 12.04. The machine contains 32 Intel Xeon CPUs and 32 GB of RAM.

4.1 Instant Pattern Quality

We evaluated the representativeness of pattern collections in combination with the individual quality of patterns, while simulating an interactive environment: we set a hard constraint on the number of patterns generated and on the time budget. The first constraint makes sure that the user is not overwhelmed by the patterns he receives for investigation. The second constraint allows explorability while retaining the attention span of the user [16]. In fact, we envision a setting where a user clicks a mining button and wants to obtain good results instantly. Moreover the mining can only start when the user tells the system to start, such that knowledge from a previous mining round can be taken into account.

We used a hard time budget of 1 second for all datasets and reported the top 25 results obtained. Throughout the experiments we did not take into account the loading times because in interactive environments the data is already in memory. For Asso we did not allow to find fault-tolerant covers and set the penalty to 100. Since Asso is greedy, the first 25 results are the top 25. For Tileminer we made a sweep over the parameter space, varying the area threshold from 5% to 95% of $area(\mathcal{T}_{larg})$ (area of largest tile) with a 5% step increase. Moreover, we used two settings: Tileminer$_{first}$, uses the first k patterns obtained and Tileminer$_{unif}$, selects k patterns uniformly at random from all tiles found in one second. For CDPS we used several area distribution settings [8]. However, in the results we show only CDPS$_{area*fq^2}$ which is the setting that performed best overall. We selected the first 25 non duplicates. For the samplers we made 10 runs and

[4] Implementation can be found at http://adrem.ua.ac.be/rts

averaged over the results. Each collection was scored wrt Equation (2). For comparison, we normalized the $area$s over $area(\mathcal{T}_{larg})$ rather than $|\mathcal{R}|$.

The qualities of pattern collections are shown in Table 1. It shows per dataset and for each of the algorithms the quality and the number of patterns obtained when running for at most 1 second and selecting 25 top patterns. The experiments show that RTS performs good on our quality measure and even outperforms other algorithms on 6 out of 7 datasets. Only for the sparse dataset CovType it is not able to produce enough patterns to beat the other methods. Moreover, it is the only method that produces at least one pattern within one second. Aside from quality this also is very important in our experiment because even while retrieving just one pattern the knowledge of the user is influenced! The user can then incorporate this knowledge when exploring the data further. It is clear that RTS outperforms all other methods on this experiments.

4.2 Pattern Quality over Time

We evaluate the quality using larger time budgets and show the evolving quality over time. In this scenario all algorithms are given at most one hour to produce 100 patterns. For Tileminer we used the same parameter selection procedure as in Section 4.1. We selected the setting giving the highest overall total coverage for the first 100 patterns and generated a random selection of 100 patterns for that complete collection of patterns. Evolving scores for quality, relative coverage and average relative area are shown in Figures 2, 3 and 4 for a selection of datasets. The graphs show elapsed time in milliseconds in log scale on the x-axis and the respective qualities, with $area$s normalized by $area(\mathcal{T}_{larg})$ rather than $|\mathcal{R}|$.

The results show similar behaviour except again on CovType, a sparse dataset with a low number of high area patterns. Generally, the experiments show that Asso is good at providing large covers while maximizing new information. However, the mining process often takes long and due to optimization of uncovered parts, the area of tiles reduces drastically after the first pattern. Tileminer is good at providing multiple large area patterns but lacks the ability to cover the data with the patterns found. This is due to the enumeration strategy. CDPS is not able to outperform other methods due to i.i.d sampling of $area * fq^2$, which does not optimize enough the area. It is however one of the fastest methods. RTS is the only method that is quick and robust in terms of quality. For all datasets it is at least an order of magnitude faster than Asso with only slightly lower total coverage. It is possible to obtain better coverage by also incorporating previous samples (Section 3.4). This, however, negatively influences the score.

5 Related Work

We describe some related research to this paper. We cite two types of research material, work based on finding database covers and work based on sampling.

Geerts et al. [10] adopted Eclat to mine non fault-tolerant (FT) large area tiles, i.e., tiles with no false positives (0 values) in the result. They also describe

how to produce tilings, by recursively finding the largest tile in the data excluding previous tiles. Vreeken et al. [18] optimize the Minimum Description Length in pattern collections. As such, they maximize the number of times patterns are used when encoding the data, rather than maximizing the total coverage.

In the FT setting multiple algorithms exist that try to find database covers. These methods allow false positives in their patterns. Miettinen et al. [13] use boolean matrix factorization in Asso, to find products of matrices that reconstruct the complete data with a low number of errors. Asso can be adopted to mine non FT patterns in binary data. Xiang et al. [20] use the concept of hyperrectangles, which is similar to hierarchical tiles. They find hyperrectangles by combining frequent itemsets with a given budget of false positives.

Sampling the output space is relatively new in pattern set mining. Existing algorithms enumerate a large fraction of the pattern space and filter a selection of patterns a posteriori. Not many techniques exist that sample the output space directly. Al Hasan and Zaki [3] coined the term output space sampling on graphs. They used a simple random walk on the partially ordered graph (POG), by allowing only extensions of the current POG. Moens and Goethals [14] used the same technique to sample maximal itemsets. They use an objective functions to prune the search space and to propose new transitions for the random walk.

Boley et al. [6] introduced two-step sampling procedures for the discovery of patterns following a target distribution. In step one a single data object is materialized and in step two a pattern is sampled from the object. Boley et al. [8] extended this into a general framework, where a user specifies a full distribution in terms of frequency factors over specific parts of the data. This framework can not optimize enough the area of patterns to find good covers. Boley [5] uses Metropolis-Hastings to uniformly sample closed itemsets, however he does not incorporate the area of patterns for biasing the results towards large area tiles.

6 Conclusion and Future Work

Interactive KD tools demanding short response times and concise pattern collections are becoming increasingly popular. Existing techniques for finding large covers in 0/1 databases fail in at least one of the two requirements and can therefore not be integrated properly in such frameworks. We presented a novel technique for sampling large database covers given a real-time situation of interactive data mining with very short time budgets. We showed how our method can be implemented efficiently using rejection sampling. Moreover, we showed that our technique outperforms existing techniques for finding large database covers given very short time. For larger time budgets we showed that our method obtains comparable results to greedy optimizations, yet, much faster.

Interesting future work on this topic relates to tiles that are enumerated in one recursive step. In the current implementation we select only the largest tile in the collection. Another technique is to maintain an evolving list of top-k tiles using for instance reservoir sampling. Other reseach directions for this technique are the FT setting and probabilistic databases.

References

1. Uci machine learning repository. http://archive.ics.uci.edu/ml/.
2. Frequent itemset mining dataset repository. http://fimi.ua.ac.be/data, 2004.
3. M. Al Hasan and M. J. Zaki. Output space sampling for graph patterns. *Proc. VLDB Endow.*, pages 730–741, 2009.
4. A. Blumenstock, J. Hipp, S. Kempe, C. Lanquillon, and R. Wirth. Interactivity closes the gap. In *Proc. of the KDD Workshop on Data Min. for Business Applications, Philadelphia, USA*, 2006.
5. M. Boley. *The Efficient Discovery of Interesting Closed Pattern Collections*. PhD thesis, 2011.
6. M. Boley, C. Lucchese, D. Paurat, and T. Gärtner. Direct local pattern sampling by efficient two–step random procedures. In *Proc. ACM SIGKDD*, 2011.
7. M. Boley, M. Mampaey, B. Kang, P. Tokmakov, and S. Wrobel. One click mining: Interactive local pattern discovery through implicit preference and performance learning. In *IDEA'13 Workshop in Proc ACM SIGKDD*, pages 27–35. ACM, 2013.
8. M. Boley, S. Moens, and T. Gärtner. Linear space direct pattern sampling using coupling from the past. In *Proc. ACM SIGKDD*, pages 69–77. ACM, 2012.
9. V. Dzyuba and M. Leeuwen. Interactive discovery of interesting subgroup sets. pages 150–161, 2013.
10. F. Geerts, B. Goethals, and T. Mielikäinen. Tiling databases. In *Discovery Science*, pages 278–289. Springer Berlin Heidelberg, 2004.
11. B. Goethals, S. Moens, and J. Vreeken. Mime: a framework for interactive visual pattern mining. In *Proc. ACM SIGKDD*, pages 757–760. ACM, 2011.
12. N. Lavrač, B. Kavšek, P. Flach, and L. Todorovski. Subgroup discovery with cn2-sd. *J. Mach. Learn. Res.*, pages 153–188, 2004.
13. P. Miettinen, T. Mielikäinen, A. Gionis, G. Das, and H. Mannila. The discrete basis problem. *IEEE Trans. on Knowl. and Data Eng.*, pages 1348–1362, 2008.
14. S. Moens and B. Goethals. Randomly sampling maximal itemsets. In *IDEA'13 Workshop in Proc ACM SIGKDD*, 2013.
15. R. M. Neal. Slice sampling. *Ann. Statist.*, pages 705–767, 2003.
16. R. T. Ng, L. V. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained association rules. In *ACM SIGMOD Record*, pages 13–24. ACM, 1998.
17. M. van Leeuwen. Interactive data exploration using pattern mining. In *Int. Knowl. Discov. and Data Min.: State-of-the-Art and Fut. Chal. in Biomed. Inf.*, LNCS. Springer, To Appear.
18. J. Vreeken, M. Leeuwen, and A. Siebes. Krimp: Mining itemsets that compress. *Data Min. Knowl. Discov.*, pages 169–214, 2011.
19. R. Škrabal, M. Šimůnek, S. Vojíř, A. Hazucha, T. Marek, D. Chudán, and T. Kliegr. Association rule mining following the web search paradigm. In *Proc. ECML-PKDD 2012*, ECML PKDD'12, pages 808–811. Springer-Verlag, 2012.
20. Y. Xiang, R. Jin, D. Fuhry, and F. F. Dragan. Summarizing transactional databases with overlapped hyperrectangles. *Data Min. Knowl. Discov.*, pages 215–251, 2011.
21. M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. Parallel algorithms for discovery of association rules. *Data Min. Knowl. Discov.*, pages 343–373, 1997.