Tiling Databases

Floris Geerts[1], Bart Goethals[2], and Taneli Mielikäinen[2]

[1] Laboratory for Foundations of Computer Science
School of Informatics, University of Edinburgh
fgeerts@inf.ed.ac.uk
[2] HIIT Basic Research Unit
Department of Computer Science
University of Helsinki, Finland
{goethals,tmielika}@cs.Helsinki.FI

Abstract. In this paper, we consider 0/1 databases and provide an alternative way of extracting knowledge from such databases using tiles. A tile is a region in the database consisting solely of ones. The interestingness of a tile is measured by the number of ones it consists of, i.e., its area. We present an efficient method for extracting all tiles with area at least a given threshold.
A collection of tiles constitutes a tiling. We regard tilings that have a large area and consist of a small number of tiles as appealing summaries of the large database. We analyze the computational complexity of several algorithmic tasks related to finding such tilings. We develop an approximation algorithm for finding tilings which approximates the optimal solution within reasonable factors. We present a preliminary experimental evaluation on real data sets.

1 Introduction

Frequent itemset mining has become a fundamental problem in data mining research and it has been studied extensively. Many efficient algorithms such as Apriori [1], Eclat [18] and FP-growth [7] have been developed to solve this problem.

More recently, a new setting, finding the top-k (closed) most frequent itemsets of a given minimum length, has been proposed [8]. For small k this provides a small and comprehensive representation of the data, but at the same time it raises the question whether frequency is the right interestingness measure. The use of frequency has been criticized in the context of association rules as well and many alternatives have been offered [16]. Unfortunately, finding good and objective interestingness measures for itemsets seems to be a hard problem. Additionally, allowing such measures to prune the huge search space of all itemsets might be even harder. Currently, most techniques rely on the monotonicity property of the frequency of itemsets. Indeed, supersets of infrequent itemsets cannot be frequent, and can therefore be pruned away from the huge space of all itemsets.

In this paper we introduce a new and objective interestingness measure for itemsets. It is based on the concept of a tile and its area. Informally, a tile consists of a block of ones in a 0/1 database as shown in Figure 1.

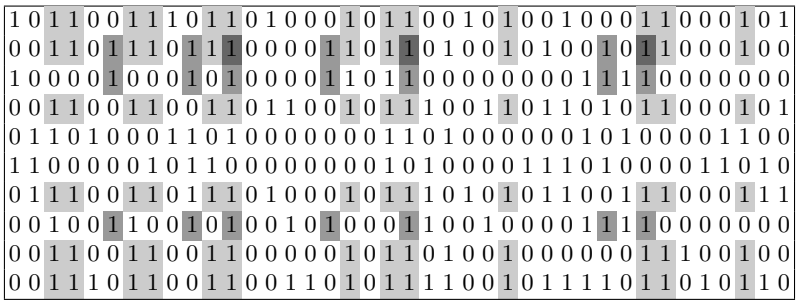

Fig. 1. Example of a 0/1 database a tiling consisting of two overlapping tiles (darkest shaded area correspond to intersection of the two tiles).

We call the number of ones in a tile the area of the tile in the database. A collection of (possibly overlapping) tiles constitutes a tiling (see Figure 1). The area of a tiling in the database is the total number of ones which are part of the tiles in the tiling. Obviously, the larger the tile or tiling, the more knowledge they represent about the database. Moreover, a large tiling consisting of only a small number of tiles can be extremely informative when one is curious about the content of a 0/1 database. Indeed, the tiling implicitely determines an upper bound on the number of different databases that are consistent with a database, and hence, it presents a small characterization of the database at hand.

In this paper we consider the following problems: the maximum k-tiling problem which asks for a tiling consisting of at most k tiles having the largest possible area; the minimum tiling problem which asks for a tiling of which the area equals to the total number of ones in the database and consists of the minimum number of tiles; the large tile mining problem which asks for all tiles in the database each having at least some minimum area; and the top-k tiles problem which asks for the k tiles that have the largest area. Furthermore, we present hardness results on the complexity of the problems listed above; we propose algorithms that solve these problems or give good approximations when optimal solutions are intractable; and we present promising experimental results on several large real world sparse and dense databases.

As we will show, the itemset constraints considered here belong to the class of length decreasing support constraints. The algorithms we propose use an adaptation and improvement of the known pruning strategies for length decreasing support constraints [15,17].

The concept of a tile is not entirely new, but the problems we consider have not yet been investigated. The notion of tiles is similar to combinatorial rectangles (communication complexity [10]), co-clusters (clustering [5]), bi-sets (formal concepts [3]), or conjunctive clusters (learning [14]). The difference with our setting is that we look for exact tiles, while most research instantly relaxes the condition of exactness and studies approximations of the tiles.

This paper is organized as follows. We formally state the problems in Section 2 and give some complexity results. An algorithm for each problem is presented in Section 3. Experimental results are reported in Section 4. We conclude the paper in Section 5.

2 Problem Statement

Let $\mathcal{I}$ be a set of n items. We can assume that $\mathcal{I}$ is a finite set $\{1, \ldots, n\}$. An itemset I is a non-empty subset of $\mathcal{I}$. A transaction is a tuple $\langle tid, T \rangle$ where tid is a transaction identifier and T is an itemset. A (transactional) database D is a set of m transactions. The size of a transaction is the number of items in it; the size of a database is the sum of the sizes of the transactions in the database. A transaction database D can also be regarded as a binary matrix of size $m \times n$ where m is the number of transactions in D and n is the number of items in $\mathcal{I}$. The (i, j)th element in the binary matrix corresponding to D is equal to 1 if the ith transaction contains item j, and it is equal to 0 otherwise. Hence, the size of D is equal to the number of ones in this matrix. We will often mix these two interpretations of a transaction database in the remainder of the paper.

Given an itemset I, a transaction $\langle tid, T \rangle$ covers I if $I \subseteq T$. The cover of an itemset I, denoted by $cover(I, D)$, consists of the transaction identifiers of transactions in D which cover I. The support of an itemset I with respect to a database D is the cardinality of $cover(I, D)$. We denote this number by $supp(I, D)$, or simply $supp(I)$ when D is clear from the context.

Definition 1 (tiles, tilings and their area). *Let I be an itemset and D be a database. The tile corresponding to I is defined as*

$$\tau(I, D) = \{(tid, i) \mid tid \in cover(I, D), i \in I\}.$$

When D is clear from the context, we write $\tau(I)$ instead. The area of $\tau(I)$ is equal to its cardinality. Moreover,

$$area(\tau(I), D) = |\tau(I)| = |I| \cdot |cover(I, D)|\,.$$

A tiling $\mathcal{T} = \{\tau(I_1), \ldots, \tau(I_k)\}$ consists of a finite number of tiles; its area is $area(\mathcal{T}, D) = |\tau(I_1) \cup \cdots \cup \tau(I_k)|$.

We call a tile *maximal* if the corresponding itemset is closed. Recall that an itemset is closed if it is not contained in a larger itemset having the same support. All tiles considered in this paper will be maximal.

The area of a tiling $\mathcal{T}$ reflects how many possible databases there exist which are consistent with the tiling. We actually can compute an upper bound on the number of such databases easily based on the area of the tiling. Let m be the number of rows and n the number of single items. Then there are maximum 2^{nm} possible databases. Since a tiling fixes $2^{area(\mathcal{T}, D)}$ entries in a database, there are only $2^{nm - area(\mathcal{T}, D)}$ possible databases left which are consistent with the tiling $\mathcal{T}$. Based on this relationship between the number of databases consistent with

$\mathcal{T}$ and the area of D which the tiling $\mathcal{T}$ covers, the most interesting tilings are the ones with the maximum area. Indeed, these tilings reduce the number of databases consistent with the tiling the most.

The main goal of this paper is to find an algorithm which computes the maximum k-tiling.

*Problem 1 (*Maximum k-Tiling). Given a database D and a positive integer k, find a tiling $\mathcal{T}$ consisting of at most k tiles with maximum area $area(\mathcal{T}, D)$.

Related to Problem 1 is the problem of finding the minimum number of tiles covering the database. This number is a measure for the complexity of the database. When the complexity of the database is small, then the maximum k-tiling for small values of k can be regarded as an excellent representation of the database.

*Problem 2 (*Minimum Tiling). Given a database D, find a tiling of D with area equal to the size of the database and consisting of the least possible number of tiles.

In order to solve Problem 1 and Problem 2, the main algorithmic task is to find all maximal tiles with area at least a given minimum area threshold.

*Problem 3 (*Large Tile Mining). Given a database D and a minimum area threshold σ, find all tiles in D with an area at least σ.

Since the collection of large tiles can be very large, it is only natural to look at a few largest tiles only:

*Problem 4 (*Top-k Tiles). Given a database D and a positive integer k, find k tiles with largest areas.

To analyze the computational complexity of finding tiles and tilings, let us first consider the simple subproblem of finding the largest tile in a given database D:

*Problem 5 (*Maximum Tile). Given a database D, find the tile with the largest area in D.

A solution for this problem is important for the approximations of good tilings.

Theorem 1. Maximum k-tiling, Minimum Tiling, Large Tile Mining, Top-k Tiles *and* Maximum Tile *are* NP*-hard.*

Proof. The Maximum Tile problem can be seen also by viewing the database as an adjacency matrix of a bipartite graph $G = (V_1, V_2, E)$ and looking for the largest complete bipartite subgraph of G, i.e., the largest subset E' of edges such that $E' = V_1' \times V_2'$ for some $V_1' \subseteq V_1$ and $V_2' \subseteq V_2$. This problem is known as the Maximum Edge Biclique problem which is shown to be NP-hard [13]. Based on this correspondence, also the Minimum Tiling problem is NP-hard [12].

The other complexity results now follow directly since Maximum Tile is a special case of the Top-k Tiles problem, with $k = 1$. It is also a special case of the Large Tile Mining problem when σ is assigned to be the maximum area of a tile in D. Finally, the Maximum Tile problem is a special case of the Maximum k-Tiling problem with $k = 1$. □

3 Algorithms

In this section, we first present the LTM algorithm for finding large tiles. LTM uses a branch and bound search strategy combined with several pruning techniques. Based on the LTM algorithm, we give an algorithm, k-LTM which finds the top-k tiles. Finally, we use k-LTM to find an approximation of a maximum k-tiling of a database. The approximation algorithm k-Tiling is a greedy algorithm and ensures an approximation within a constant factor from the optimal solution.

3.1 Large Tile Mining

Given a database D and a minimum area threshold σ, we want to find all maximal tiles which have an area at least σ.

In our algorithm, we will refine pruning techniques used in the context of length-decreasing support constraints on itemsets [15,17]. Let $f(x)$ be a monotone decreasing function, i.e., $f(x) \geq f(x+1) \geq 1$; f is called a length-decreasing support constraint and an itemset I is called frequent w.r.t. f if $supp(I, D) \geq f(|I|)$. An itemset I which correspond to a tile $\tau(I)$ of area at least σ can be seen as an itemset frequent w.r.t the length- decreasing support constraint $f(|I|) = \sigma/|I|$. Indeed, $area(I, D) = |I| \cdot supp(I, D) \geq \sigma$ iff $supp(I, D) \geq \sigma/|I|$.

First, our algorithms will follow a similar depth-first search strategy and counting mechanism as used in the Eclat algorithm and its variants [18,19]. The same search strategy was later also successfully used in the FP-growth algorithm [7], and is based on a divide and conquer mechanism.

Denote the set of all tiles in D with area at least σ, corresponding to itemsets with the same prefix $I \subseteq \mathcal{I}$ by $\mathcal{T}[I](D, \sigma)$.

The main idea of the search strategy is that all large tiles containing item $i \in \mathcal{I}$, but not containing any item smaller than i, can be found in the so called *i-conditional database* [7], denoted by D^i. That is, D^i consists of those transactions from D that contain i, and from which all items before i, and i itself are removed. In general, for an itemset I, we can create the I-conditional database, D^I, consisting of all transactions that contain I, but from which all items before the last item in I and that item itself have been removed. Whenever we compute the area of a tile in D^I, we simply need to add $|I|$ to the width of the tile and multiply this with the support of the corresponding itemset. Then, after adding I to the itemset, we found exactly all large tiles containing I, but not any item before the last item in I which is not in I, in the original database, D.

The large tile mining algorithm LTM as shown in Figure 2, recursively generates for every item $i \in \mathcal{I}$ the set $\mathcal{T}[\{i\}](D^i, \sigma)$. (Note that $\mathcal{T}[\{\}](D, \sigma) = \bigcup_{i \in \mathcal{I}} \mathcal{T}[\{i\}](D^i, \sigma)$ contains all large tiles.)

In order to compute the area of a tile, we need to know the support of the itemset representing that tile. This value is computed exactly as in the Eclat algorithm. That is, the algorithm stores the database in its vertical layout

Input: D, σ, I (initially called with $I = \{\}$)
Output: $\mathcal{T}[I](D, \sigma)$

1: $\mathcal{T}[I] := \{\}$;
2: $Prune(D, \sigma, I)$.
3: **for all** i occurring in D **do**
4: **if** $|cover(\{i\})|(|I|+1) \geq \sigma$ **then**
5: Add $\tau(I \cup \{i\})$ to $\mathcal{T}[I]$;
6: **end if**
7: $D^i := \{\}$;
8: **for all** j occurring in D such that $j > i$ **do**
9: $C := cover(\{i\}) \cap cover(\{j\})$;
10: Add (j, C) to D^i;
11: **end for**
12: Compute $\mathcal{T}[I \cup \{i\}](D^i, \sigma)$ recursively;
13: Add $\mathcal{T}[I \cup \{i\}]$ to $\mathcal{T}[I]$;
14: **end for**

Fig. 2. The LTM algorithm.

Input: D, σ, I

1: **repeat**
2: **for all** i occurring in D **do**
3: **if** $UB_{I \cup \{i\}} < \sigma$ **then**
4: Remove i from D.
5: **end if**
6: **for all** $tid \in cover(\{i\})$ **do**
7: **if** $size(tid) < ML_{I \cup \{i\}}$ **then**
8: Remove tid from $cover(\{i\})$.
9: **end if**
10: **end for**
11: **end for**
12: **until** nothing changed

Fig. 3. The Prune procedure.

which means that each item is stored together with its cover instead of listing explicitly all transactions. In this way, the support of an itemset I can be easily computed by simply intersecting the covers of any two subsets $J, K \subseteq I$, such that $J \cup K = I$. This counting mechanism is perfectly suited for our purposes since it immediately gives us the list of transaction identifiers that, apart from the itemset itself, constitutes a tile.

We now describe the LTM algorithm in more detail. First, the algorithm is initialized (line 1). Then, on line 2, the main pruning mechanism is executed, as will be explained later. This mechanism will remove certain items from the candidate set that can no longer occur in a large tile and remove transactions that can no longer contribute to the area of a large tile. On line 3, the main loop of the algorithm starts by considering each item separately. On lines 4–6, each large tile is added in the output set. After that, on lines 7–11, for every item i, the i-projected database $\mathcal{D}^i$ is created. This is done by combining every item j with i, such that $j > i$ and computing its cover by intersecting the covers of both items (line 10). On line 12, the algorithm is called recursively to find all large tiles in the new database $\mathcal{D}^i$. However, in every such conditional database, each item must be treated as the tile represented by the itemset $I \cup \{i\}$.

For reasons of presentation, we did not add the details of restricting the search space to closed itemsets such that only maximal tiles are considered. Nevertheless, the algorithm can be easily extended with the techniques used in the CHARM algorithm [19] such that only closed itemsets are generated.

Unfortunately, if the area of a tile does not meet the minimum area threshold, we cannot simply prune away all tiles in its branch, because its supersets considered in that branch might still represent tiles with large areas. Neverthe-

less, it is possible to compute an upper bound on the area of any tile that can still be generated in the current branch. Based on this upper bound, the *Prune* procedure, as shown in Figure 3, is able to prune some items from the search space and at the same time it reduces the size of the database.

The *Prune* procedure consist of a repeated application of the node pruning methods used in LPMiner [15] and BAMBOO [17]. More specifically, for each item i in the database D (line 2) we compute an upper bound $UB_{I\cup\{i\}}$ of the largest possible tile containing $I\cup\{i\}$. To obtain this upper bound for a given i, we count how many transactions in the current I-conditional database, containing i, have size at least ℓ, for all occurring ℓ. Denote this number by $supp_{\geq\ell}(i, D^I)$. Then, the upper bound on the size of the largest possible area of a tile containing $I \cup \{i\}$ is given by $UB_{I\cup\{i\}} = \max\{(|I| + \ell) \cdot supp_{\geq\ell}(i, D^I) \mid \ell \in \{1, 2, \ldots\}\}$. Obviously, to compute this number, we need the size of each transaction in the current conditional database. This size is perfectly derivable from the vertical representation of the conditional database. In practice, however, it is more efficient to store a separate array in which the size of each transaction is stored for each conditional database. Also, note that the size for a given transaction in the current conditional database can be much smaller than the size of that transaction in the original database.

If the upper bound for a given item i in the database is smaller than the minimum area threshold (line 3), then this means that i will never be part of an itemset corresponding to a tile of area larger than σ, and therefore, it can be removed from D (line 4).

This also implicitly means that the size of all transactions is decreased, which may affect the upper bounds of the other items, and hence, they can be recomputed. This process can be repeated until no more items can be removed.

Even if an item cannot be completely removed from the database, it is sometimes possible to remove it from several transactions. More specifically, consider the following number $ML_{I\cup\{i\}} = \min\left\{\ell \mid \ell \cdot supp_{\geq\ell}(i, D^I) \geq \sigma\right\}$. This number gives the minimum size of a transaction containing i, that can still generate a tile with area at least σ. Hence, from each transaction containing i that is shorter, i can be removed (lines 7–8). In the Prune procedure in Figure 3, we find the size of a transaction *tid* using *size*(*tid*). Again, this removal can have an effect on the upper bound of the other items, such that their upper bounds can be recomputed and maybe some of them can still be removed.

3.2 Top-k Tiles

In order to find the top-k largest tiles, we adapt the LTM algorithm as follows. Initially the minimum area threshold is zero. Then, after the algorithm has generated the first k large tiles, it increases the minimum area threshold to the size of the smallest of these k tiles. From here on, the minimum area threshold can be increased every time a large tile is generated w.r.t. the current threshold. All generated tiles that do not have an area larger than the increased minimum area threshold can of course be removed.

3.3 Finding the Tiling

Even if finding tilings close to the best ones with reasonable guarantees is not feasible in theory, we would like to find good tilings anyway.

For the Minimum Tiling problem it is clear what the best tiling is: it is a complete tiling of the database with the smallest number of tiles. The best k-tiling can be similarly considered to be the k-tiling with the largest area.

However, in data mining this might not be enough: due to the exploratory nature of data mining the data mining tool should support interactive use. For example, determining the value k in advance might be an unreasonable requirement for the data analyst. Instead, the tool should make it as convenient as possible to explore different values of k. This is not the case if the suggested k-tiling and $k+1$-tiling differ a lot from each other. In the best case (for the data analyst) the suggested k-tiling and $k+1$-tiling would differ only by one tile.

Such k-tilings for all values of k can be determined simply by fixing an ordering for the tiles and considering the k first tiles in the ordering as the best k-tiling [11]. Clearly, this kind of ordering does not provide k-tilings with maximum area for all values of k but some approximation ratio might be guaranteed for all values of k simultaneously if the ordering of the tiles would be determined in a good way. For example, if the ordering of tiles is constructed greedily by adding the tile that covers the largest area of the uncovered parts of the database, we get decent upper bounds for the approximation ratios of the Minimum Tiling and the Maximum k-Tiling problems.

Theorem 2. *The* Minimum Tiling *problem can be approximated within the factor* $\mathcal{O}(\log nm)$ *and* Maximum k-Tiling *can be approximated within the factor* $e/(e-1)$ *for all values of* k *simultaneously, given an oracle that finds for any database* D *and tiling* $\mathcal{T}$ *the tile* $\tau(I)$ *such that* $\mathit{area}(\mathcal{T} \cup \{\tau(I)\}, D) = \max_{I' \subseteq \mathcal{I}} \mathit{area}(\mathcal{T} \cup \{\tau(I')\}, D)$.

Proof. These problems can be interpreted as instances of the Minimum Set Cover problem and the Maximum k-Coverage problem, respectively: the set that is to be covered corresponds to the ones in the database and the sets that can be used in the cover are the maximal tiles in D. The only problem in the straightforward reduction to set cover is that the number of maximal tiles can be exponential in the size of the database. Fortunately, the greedy algorithm for set cover that gives the desired approximation bounds for both variants of the problem depends only on the ability of finding the tile $\tau(I, D)$ that maximizes the area $\mathit{area}(\mathcal{T} \cup \{\tau(I)\}, D)$ for a given collection $\mathcal{T}$ of tiles [6].

If we have such an algorithm, then the approximation bounds follow from the bounds on the Minimum Set Cover problem and the Maximum k-Coverage problem [2]. □

The oracle that gives the tile covering the most extra area outside a given tiling, can be implemented reasonably efficiently by adapting the LTM algorithm. Whenever we compute the area of a tile (line 4) in the LTM algorithm in Figure 2, we subtract the part of the area that was already covered earlier.

Additionally, we can also improve the Prune procedure by computing a second upper bound which takes the already covered area into account. Indeed, the current upper bound needs to take the size of the original transactions into account and we can not simply remove what is already covered from them, since the remaining part might not even be a tile anymore. We can, however, store a second array containing only the uncovered size of each transaction. Using this array, the uncovered area of any candidate tile in the current conditional database, which contains item i, is at most the sum of of these sizes of the transactions that contain i. Finally, we take the minimum of these two bounds: $UB^*_{I\cup\{i\}} = \min\{UB_{I\cup\{i\}}, \sum_{tid\in cover(\{i\})} size^*(tid)\}$, which can replace the old upper bound (line 4) in the Prune procedure in Figure 3. Note that $size^*$ now stores the sizes of the uncovered parts of the transactions, containing item i.

Thus, in practice we can solve the problems Minimum Tiling and Maximum k-Tiling with very good approximation bounds.

4 Experimental evaluation

We implemented our algorithms in C++ and experimented on a 2GHz Pentium-4 PC with 1GB of memory, running Linux.

We present the evaluation of the algorithms on two substantially different datasets. The mushroom data set is a dense dataset, containing characteristics of various species of mushrooms and was originally obtained from the UCI repository of machine learning databases [4]. It consists of 119 single items, 8 124 transactions, resulting in 186 852 ones. The BMS-Webview-1 dataset is a sparse dataset, containing several months of click-stream data from an e-commerce website, and is made publicly available by Blue Martini Software [9]. It consists of 497 items, 59 602 transactions, resulting in 149 639 ones.

In the first series of experiments we ran the LTM algorithm for various minimum area thresholds, which is shown in Figure 4. As can be seen, the execution times show the feasibility of our approach, taking into account the extremely low minimum area thresholds. Indeed, a minimum area of 250 for the BMS-Webview-1 dataset, corresponds to $250/149\,639 = 0.3\%$ of the ones in the database. Similarly, for the mushroom dataset, a minimum area of 250 corresponds to $250/186\,852 = 0.1\%$ of the ones in the database.

The number of tiles found and the number of generated candidate tiles by the LTM algorithm for various minimum area thresholds is shown in Figure 5. Here, it shows that the dense mushroom dataset indeed contains a huge number of large tiles, while the sparse BMS-Webview-1 dataset contains a modest number of large tiles, but a lot smaller ones. For the same reason, the number of candidate tiles is much closer to the actual number of large tiles in the mushroom dataset as compared to the BMS-Webview-dataset. Note, since the mushroom database contains only 8 124 transactions, the minimum area threshold of 9 000 forces that no discovered tile consists of only a single item. Similarly, a minimum area threshold which is larger than the size of all transactions, guarantees that no single transaction can be a large tile.

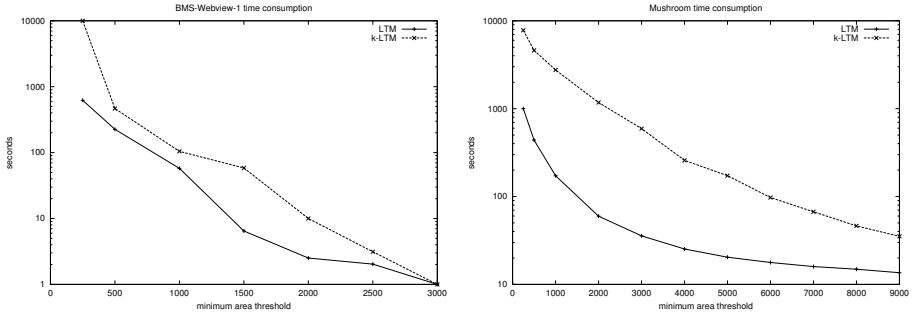

Fig. 4. Large tile mining on the BMS-Webview-1 (left) and Mushroom (right) data set. The plots show the execution times of the LTM and k-LTM algorithms for various minimum area thresholds.

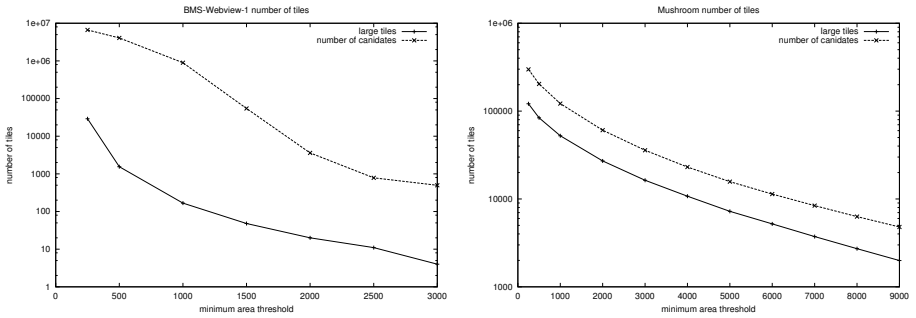

Fig. 5. Large tile mining on the BMS-Webview-1 (left) and Mushroom (right) data set. The plots show the number of tiles found and the number of tiles checked by LTM algorithm for various minimum area thresholds.

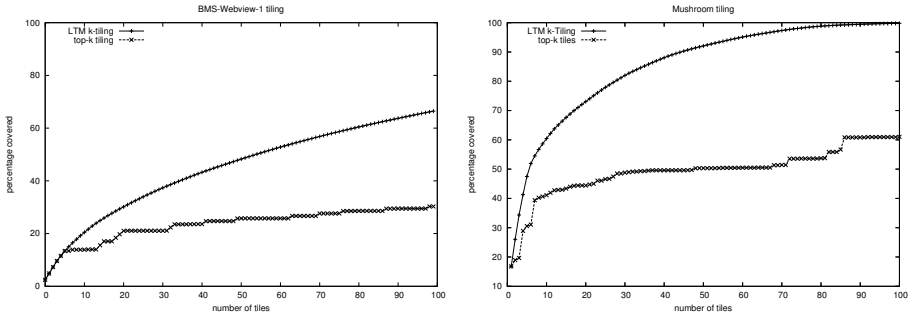

Fig. 6. Comparison of the covered area of tilings for the BMS-Webview-1 dataset (left) and Mushroom dataset (right) obtained by running the k-LTM algorithm and the k-Tiling algorithm on these data sets for different values of k.

In the second series of experiments we ran the k-LTM algorithm for various minimum area thresholds, which is shown in Figure 4. As can be seen, the execution times are larger than the execution times of the standard LTM algorithm, but it still shows the feasibility of our approach.

Finally, to show the effectiveness of the k-Tiling algorithm, we tested the algorithm for a varying number k of tiles. To compare, we plot the percentage of the number of ones covered by the k-Tiling algorithm, together with the number of ones covered by taking the top-k tiles generated by the k-LTM algorithm. The results, shown in Figure 6, are striking for the mushroom dataset. Indeed, already 60% of the ones in the database are covered by only 10 tiles! To obtain a tiling that covers 90% of the database, only 45 tiles are needed. For the BMS-Webview-dataset, slightly more tiles are needed to obtain a similar tiling, which is of course expected, given the sparsity of this database.

5 Conclusions and Future Work

We introduced the concepts of tiles and tilings in the context of transaction or 0/1 databases and argued that the area of a tile and tilings is a good interestingness measure. Indeed, a tiling implicitely determines an upper bound on the number of different databases that are consistent with a database, and hence, it presents a small characterization of the database at hand.

We presented some computational challenges to computing different types of tilings. A theoretical analysis of these challenges shows that they are all NP-hard. In practice, however, computing these tilings might be still feasible.

Therefore, based on a powerful iterative upper bounding mechanism, we developed an elegant branch and bound algorithm that discovers all large tiles with respect to a minimum area threshold. This algorithm prunes away a lot of tiles from the huge search space of all possible tiles. Based on this algorithm, we derive two other algorithms using small adaptations for the problem of finding the k largest tiles and finding the largest tiling consisting of k tiles only.

Our experiments verify the efficacy and efficiency of the algorithms. From a knowledge discovery point of view, we show that using only a marginal number of tiles, we are already able to present a very good picture of the databases at hand.

This work is preliminary in the sense that the the knowledge extraction aspect is barely investigated. The main goal of this paper was to introduce the problem and show the feasibility of our approach. In future work, we will investigate the quality of tilings as knowledge representations. A more thorough experimental evaluation and comparison with existing approaches (frequent itemsets, top-k frequent itemsets) is postponed to a follow-up paper as well. Finally, we want to improve upon the upper bound of databases consistent with a given tiling, since that would result in more accurate interestingness measures.

Acknowledgments. We wish to thank Blue Martini Software for contributing the KDD Cup 2000 data [9].

References

1. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A.I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, chapter 12, pages 307–328. AAAI/MIT Press, 1996.
2. G. Ausiello, P. Crescenzi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer-Verlag, 1999.
3. J. Besson, C. Robardet, and J.-F. Boulicaut. Constraint-based mining of formal concepts in transactional data. *Proceedings of PAKDD'04*, pages 615–624, 2004.
4. C.L. Blake and C.J. Merz. UCI repository of machine learning databases. http://www.ics.uci.edu/ mlearn/MLRepository.html, 1998.
5. I.S. Dhillon, S. Mallela, and D.S. Modha. Information-theoretic co-clustering. *Proceedings of KDD'03*, pages 89–98, 2003.
6. U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the Association for Computing Machinery*, 45(4):634 – 652, 1998.
7. J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery*, 8(1):53–87, 2004.
8. J. Han, J. Wang, Y. Lu, and P. Tzvetkov. Mining top-k frequent closed patterns without minimum support. In *Proceedings of ICDM'02*, pages 211–218, 2002.
9. R. Kohavi, C. Brodley, B. Frasca, L. Mason, and Z. Zheng. KDD-Cup 2000 organizers' report: Peeling the onion. *SIGKDD Explorations*, 2(2):86–98, 2000. `http://www.ecn.purdue.edu/KDDCUP`.
10. E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge, 1996.
11. T. Mielikäinen and H. Mannila. The pattern ordering problem. In *Proceedings of PKDD'03*, volume 2838 of *Lecture Notes in Artificial Intelligence*, pages 327–338. Springer-Verlag, 2003.
12. J. Orlin. Containment in graph theory: covering graphs with cliques. *Indigationes Mathematicae*, 39:211–128, 1977.
13. R. Peeters. The maximum edge biclique is NP-complete. *Discrete Applied Mathematics*, 131:651–654, 2003.
14. D. Ron, N. Mishra, and R. Swaminathan. On conjunctive clustering. *Proceedings of COLT'03*, pages 448–462, 2003.
15. M. Seno and G. Karypis. LPMiner: An algorithm for finding frequent itemsets using length-decreasing support constraint. *Proceedings of ICDM'01*, pages 505–512, 2001.
16. P.-N. Tan, V. Kumar, and J. Srivastava. Selecting the right interestingness measure for association patterns. In *Proceedings of KDD'02*, pages 32–41, 2002.
17. J. Wang and G. Karypis. BAMBOO: Accelerating closed itemset mining by deeply pushing the length-decreasing support constraint. *Proceedings of SIAM DM'04*, 2004.
18. M.J. Zaki. Scalable algorithms for association mining. *IEEE TKDE*, 12(3):372–390, 2000.
19. M.J. Zaki and C.-J. Hsiao. CHARM: An efficient algorithms for closed itemset mining. In R. Grossman, J. Han, V. Kumar, H. Mannila, and R. Motwani, editors, *Proceedings of SIAM DM'02*, 2002.