Discovering Descriptive Tile Trees
by Mining Optimal Geometric Subtiles

Nikolaj Tatti and Jilles Vreeken

Advanced Database Research and Modeling
Universiteit Antwerpen
{nikolaj.tatti,jilles.vreeken}@ua.ac.be

Abstract. When analysing binary data, the ease at which one can interpret results is very important. Many existing methods, however, discover either models that are difficult to read, or return so many results interpretation becomes impossible. Here, we study a fully automated approach for mining easily interpretable models for binary data. We model data hierarchically with noisy tiles—rectangles with significantly different density than their parent tile. To identify good trees, we employ the Minimum Description Length principle.

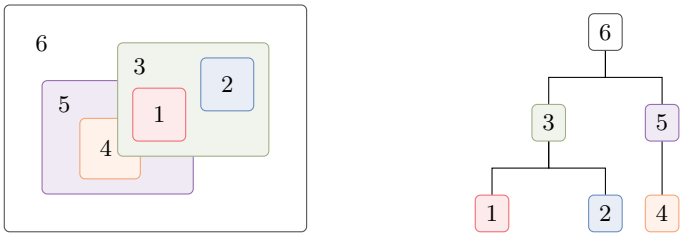
We propose Stijl, a greedy any-time algorithm for mining good tile trees from binary data. Iteratively, it finds the locally *optimal* addition to the current tree, allowing overlap with tiles of the same parent. A major result of this paper is that we find the optimal tile in only $\Theta(NM \min(N, M))$ time. Stijl can either be employed as a top-k miner, or by MDL we can identify the tree that describes the data best.

Experiments show we find succinct models that accurately summarise the data, and, by their hierarchical property are easily interpretable.

1 Introduction

When exploratively analysing a large binary dataset, being able to easily interpret the results is of utmost importance. Many data analysis methods, however, have trouble meeting this requirement. With frequent pattern mining, for example, we typically find overly many and highly redundant results, hindering interpretation [10]. Pattern set mining [2, 5, 21] tackles these problems, and instead provides small and high-quality sets of patterns. However, as these methods generally exploit complex statistical dependencies between patterns, the resulting models are often difficult to fully comprehend.

When analysing 0–1 data, the encompassing question is 'how are the 1s distributed?'. In this paper, we focus on the underlying questions of 'where are the ones?' and 'where are the zeroes?'. To answer these questions in an easily interpretable manner, we propose to model the data hierarchically, by identifying trees of tiles, i.e. sub-matrices that are surprisingly dense or sparse compared to their parent tile. As an example, consider Figure 1, in which we show a toy example of a hierarchical tiling, and the corresponding tile tree. As the figure shows, tiles model parts of the data, and subtiles provide refinements over their



(a) Hierarchical tiling (b) Tile tree

Fig. 1. Toy example of a tiled database, and the corresponding tile tree structure

parents. Next, as an example on real data, consider Figure 2, in which we show the tiling our algorithm discovered on paleontological data. Very easily read, using only 14 tiles, the tiling shows which regions of the data are relatively dense (dark), as well as where relatively few 1s are found (light).

Clearly, we aim to mine descriptions that are succinct, non-redundant, and neither overly complex nor simplistic. We therefore formalise the problem in terms of the Minimum Description Length (MDL) principle [9], by which we can automatically identify the model that best describes the data, without having to set any parameters. For mining good models, we introduce Stijl, a heuristic any-time algorithm that iteratively greedily finds the optimal subtile and adds it to the current tiling. A major result of this paper is that we show that we can find such optimal subtiles in only $\Theta(NM \min(N, M))$, as opposed to $\Theta(N^2M^2)$ when done naively [8].

We are not the first to study the problem of hierarchical tiling. The problem was first introduced by Gionis et al. [8], whom proposed a randomised approach as an alternative to the naive approach. Our FindTile procedure, on the other hand, is deterministic and identifies *optimal* subtiles. Moreover, our MDL formalisation requires no scaling parameters, making the method parameter-free.

These differences aside, both methods assume an order on the rows and columns of the data; as for such data, a subtile can be straightforwardly defined by a 'from' and 'to' selection query. As such, we exploit that the data is ordered, as this allows us to generate more easily understandable and easily visually representable models for the data. Although many datasets naturally exhibit

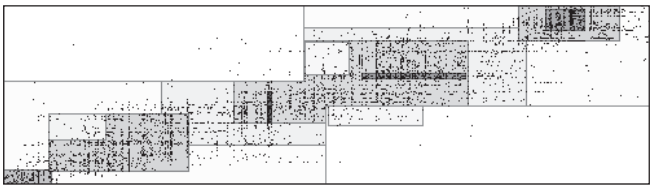


Fig. 2. Tiling of the *Paleo* dataset. See Fig. 4(b) for a cleaned version without 1s

such order, e.g. spatially and/or temporally, not all data does. For unordered data, e.g. through spectral ordering, good orders can be discovered [6, 8, 19].

Experimentation on our method shows we discover easily interpretable models that describe the data very well. STIJL mines trees that summarise the data succinctly, with non-redundant tile trees that consist of relatively few tiles.

The paper is organised as follows. Section 2 discusses preliminaries. Section 3 gives the STIJL algorithm for mining tile trees, and Section 4 details mining optimal subtiles. We discuss related work in Section 5, and experiment in Section 6. We round up with discussion and conclusions. Due to lack of space, we give the proofs for Propositions 1–2 in the Appendix.[1]

2 Encoding Data with Tile Trees

We begin by giving the basic definitions we use throughout the paper, after which we discuss how we can measure the quality of a hierarchical tile set.

Notation A *binary dataset* D is a binary matrix of size N-by-M consisting of N rows, binary vectors of size M. We denote $(i,j)^{\text{th}}$ entry of D by $D(i,j)$. We assume that both rows and columns have an order and, for simplicity, we assume that the indexing corresponds to the orders.

A geometric *tile* $X = (a,b) \times (c,d)$, where $1 \leq a \leq b \leq N$ and $1 \leq c \leq d \leq M$, identifies a consecutive submatrix of D. In contrast, for combinatorial tiles, the rows and columns are not required to be consecutive. In this paper, we focus on geometric tiles. We say that $X_1 = (a_1,b_1) \times (c_1,d_1)$ is a subtile of $X_2 = (a_2,b_2) \times (c_2,d_2)$ if X_1 is completely covered by X_2, that is, $a_2 \leq a_1$, $b_1 \leq b_2$, $c_2 \leq c_1$, and $d_1 \leq d_2$. We will write $(i,j) \in X$ if $a \leq i \leq b$ and $c \leq j \leq d$.

A *tile tree* $\mathcal{T}$ is a tree of tiles such that each child of a tile $X \in \mathcal{T}$ is a subtile of X. We will denote the children of X by $\mathit{children}(X)$. In our setting, the order of the children matters, so we assume that $\mathit{children}(X)$ is a *list* of tiles and not a set. We also assume that the root tile always covers the whole data. Given a tile tree $\mathcal{T}$, a tile $X \in \mathcal{T}$ and a subtile Y of X, we will write $\mathcal{T} + X \to Y$ to denote a tile tree obtained by adding Y as a last child of X.

Our next step is to define which data entries are covered by which tile. Since we allow child tiles to overlap, the definition is involved—although intuition is simple: the first most-specific tile that can encode a cell, encodes its value, and all other tiles ignore it. More formally, given a tile tree $\mathcal{T}$, consider a post-order, that is, an order where the child tiles appear before their parents and such that if $\mathit{children}(X) = (Y_1, \ldots, Y_L)$, then Y_i is appears before Y_{i+1}. Let $X \in \mathcal{T}$. We define $\mathit{tid}(X;\mathcal{T})$ to be the position of X in the post-order. When $\mathcal{T}$ is clear from the context we will simply write $\mathit{tid}(X)$. An example of the post-order is given in Figure 1(b). Using this order we can define which entries belong to which tile. We define

$$\mathit{area}(X;\mathcal{T}) = \{(i,j) \in X \mid \text{there is no } Y \text{ with } (i,j) \in Y, \mathit{tid}(Y) < \mathit{tid}(X)\},$$

[1] http://adrem.ua.ac.be/stijl/

that is, entries are assigned to the cells first-come first-serve, see Figure 1(a) as an example. Among these entries, we define the number of 1s and 0s as

$$p(X;\mathcal{T},D) = |\{(i,j) \in \mathit{area}(X;\mathcal{T}) \mid D(i,j) = 1\}| \quad \text{and}$$
$$n(X;\mathcal{T},D) = |\{(i,j) \in \mathit{area}(X;\mathcal{T}) \mid D(i,j) = 0\}| \quad .$$

Let us denote by $|\mathcal{T}|$ the number of tiles in a tree $\mathcal{T}$, i.e. $|\mathcal{T}| = |\{X \in \mathcal{T}\}|$, and denote by $\mathcal{T}_0$ the most simple tile tree consisting of only a root tile.

MDL for Tile Trees Our main goal is to find tile trees that summarise the data well; they should be succinct yet highly informative on where the 1s on the data are. We can formalise this intuition through the Minimum Description Length (MDL) principle [9], a practical version of Kolmogorov Complexity [13]. Both embrace the slogan *Induction by Compression*. The MDL principle can be roughly described as follows: Given a dataset D and a set of models $\mathcal{X}$ for D, the best model $X \in \mathcal{X}$ is the one that minimises $L(X) + L(D \mid X)$ in which $L(X)$ is the length, in bits, of the description of the model X, and $L(D \mid X)$ is the length, in bits, of the data as described using X.

This is called two-part MDL, or *crude* MDL. This stands opposed to *refined* MDL, where model and data are encoded together [9]. We use two-part MDL because we are specifically interested in the model: the tile tree $\mathcal{T}^*$ that yields the minimal description length. Further, although refined MDL has stronger theoretical foundations, it cannot be computed except for some special cases [9]. Before we can use MDL to identify good models, we will have to define how to encode a database given a tile tree, as well as how to encode a tile tree.

We encode the values of $\mathit{area}(X;\mathcal{T})$ using prefix codes. The length of an optimal prefix code is given by Shannon entropy, i.e. $-\log P(\cdot)$, where $P(\cdot)$ is the probability of a value [4]. We have the optimal encoded length for all entries $\mathit{area}(X;\mathcal{T})$ of a tile X in a tile tree $\mathcal{T}$ as

$$L(D \mid X,\mathcal{T}) = L(p(X;\mathcal{T},D)\,, n(X;\mathcal{T},D))\,,$$

where $L(p,n) = -p\log\frac{p}{p+n} - n\log\frac{n}{p+n}$ is the scaled entropy.

In order to compare fairly between models, MDL requires the encoding to be lossless. Hence, besides the data, we also have to encode the tile tree itself.

We encode tile trees node per node, in reverse order, and add extra bits between the tiles to indicate the tree structure. We use a bit of value 1 to indicate that the next tile is a child of the current tile, and 0 to indicate that we have processed all child tiles of the current tile. For example, the tree given in Figure 1(b) is encoded, with <tile i> indicating an encoded tile, as

<tile 6>1<tile 5>1<tile 4>001<tile 3>1<tile 2>01<tile 1>000 .

To encode an individual tile, we proceed as follows. Let X be a non-root tile and let $Z = (a,b) \times (c,d)$ be the direct parent tile of X. As we know that X is a subtile of Z, we know the end points for defining the area of X fall within those of Z. As such, to encode the 4 end points of X we need only $4\log(b-a+1) + 4\log(d-c+1)$ bits.

We also know that number of 1s in X are bounded by the area of Z, $(b - a + 1)(d - c + 1)$, and hence we can encode the number of 1s in X in $\log(b - a + 1) + \log(d - c + 1)$ bits. Note that although we can encode the number of 1s more efficiently by using the geometry of X instead of Z, this would introduce a bias to small tiles.

Next, to calculate the encoded size of a tile, we need to take the two bits for encoding the tree structure of X into account. As describe above, one bit is used to indicate that X has no more children and the other to indicate that X is a child of Z. Putting this together, the encoded length of a non-root tile X is

$$L(X \mid \mathcal{T}) = 1 + 1 + 5\log(b - a + 1) + 5\log(d - c + 1) \quad .$$

Let us now assume that X is the root tile. Since we require that a root tile covers the whole data, we need to encode the dimensions of the data set, the number of 1s in X, and following 1 bit to indicate that all tiles have been processed. Unlike for the other tiles in the tree, we have no upper bound for the dimensions of X, and therefore would have to use a so-called Universal Code [9] to encode the dimensions—after which we could subsequently encode the number of 1s in X in $\log N + \log M$ bits. However, as the lengths of these codes are constant over all models for D, and we can safely ignore them when selecting between models. For simplicity, for a root tile X we therefore define $L(X \mid \mathcal{T}) = 0$.

As such, we have for the total encoded size of a database D and a tile tree $\mathcal{T}$

$$L(D, \mathcal{T}) = \sum_{X \in \mathcal{T}} L(X \mid \mathcal{T}) + L(D \mid X, \mathcal{T}) \quad ,$$

by which we now have a formal MDL score for tile trees.

3 Mining Good Tile Trees

Now that we have defined how to encode data with a tile tree, our next step is to find the best tile tree, i.e. the tile tree minimising the total encoded length. That is, we want to solve the following problem.

Problem 1 (Minimal Tile Tree). Given a binary dataset D, find a tile tree $\mathcal{T}$ such that the total encoded size, $L(D, \mathcal{T})$, is minimised.

As simply as it is stated, this minimisation problem is rather difficult to solve. Besides that the search space of all possible tile trees is rather vast, the total encoded size $L(D, \mathcal{T})$ does not exhibit trivial structure that we can exploit for fast search, e.g. (weak) monotonicity. Hence, we resort to heuristics.

For finding an approximate solution to the Minimal Tile Tree problem, we propose the Stijl algorithm.[2] We give the pseudo-code as Algorithm 1. We iteratively find that subtile Y of a tile $X \in \mathcal{T}$ by which the total encoded size is minimised. We therefore refer to Y as the optimal subtile of X. After identifying

[2] named after the art movement De Stijl, to which art our models show resemblance.

Algorithm 1: STIJL$(D, \mathcal{T}, X)$

input : dataset D, current tile tree $\mathcal{T}$, parent tile X
output : updated tile tree $\mathcal{T}$

```
Y ← subtile of X minimising L(D, T + X → Y);
while L(D, T + X → Y) < L(D, T) do
    T ← STIJL(D, T + X → Y, Y);
    Y ← subtile of X minimising L(D, T + X → Y);
return T;
```

the optimal subtile, STIJL adds Y into the tile tree, and continues inductively until no improvement can be made.

Alternative to this approach, we can also approximate the optimal k-tile tree. To do so, we adapt the algorithm to find the subtile Y over all parent tiles $X \in \mathcal{T}$ that minimises the score—as opposed to our standard depth-first strategy. Note that by the observation above, for the k at which the score is minimised, both strategies find the same tree.

By employing a greedy heuristic, we have reduced the problem of finding the optimal tile tree into a problem of finding the optimal subtile.

Problem 2 (Minimal Subtile). Given a binary dataset D, a tile tree $\mathcal{T}$, and a tile $X \in \mathcal{T}$, find a tile Y such that Y is a subtile of X, and $\mathcal{T} + X \to Y$ is minimised.

The main part of this paper details how to find an optimal subtile, a procedure we subsequently use in STIJL.

4 Finding the Optimal Tile

In this section we focus on finding the optimal subtile. Naively, we solve this by simply testing every possible subtile, requiring $\Theta(N^2M^2)$ tests, where N and M are the number of rows and columns in the parent tile, respectively [8].

In this section, we present an algorithm that can find the optimal subtile in $\Theta(N^2M)$. In order to do that, we will break the problem into two subproblems. The first problem is that for two *fixed* integers $c \leq d$, we need to find two integers $a \leq b$ such that the tile $(a, b) \times (c, d)$ is optimal. Once we have solved this, we can proceed to find the optimal tile by finding the optimal (c, d).

We begin by giving an easier formulation of the function we want to optimise. In order to do so, note that adding a subtile Y to X changes only $area(T; \mathcal{X})$, and does not influence (the encoded length of) other tiles in the tree. Hence, we expect to be able to express the difference in total encoded length between $\mathcal{T} + X \to Y$ and $\mathcal{T}$ in simple terms. In fact, we have the following theorem.

Proposition 1. *Let $\mathcal{T}$ be a tile tree. Let $X \in \mathcal{T}$ be a tile and let Y be a subtile of X. Define $\mathcal{T}' = \mathcal{T} + X \to Y$ and have $u = p(Y; \mathcal{T}')$, $v = n(Y; \mathcal{T}')$, $o = p(X; \mathcal{T})$, and $z = n(X; \mathcal{T})$. Then*

$$L(D, \mathcal{T}') - L(D, \mathcal{T}) = L(u, v) + L(o - u, z - v) - L(o, z) + L(Y \mid \mathcal{T}') \quad .$$

In order to find the optimal subtile it is enough to create an algorithm for finding an optimal subtile more dense than its parent tile. To see this, note that we can find the optimal tile by first finding the optimal *dense* tile, and then find the optimal *sparse* tile by applying the same algorithm on the 0–1 inverse of the data. Once we have both the optimal optimal dense and optimal sparse tiles, we can choose the overall optimal subtile by MDL.

Let $X = (s, e) \times (x, y)$ be a tile, and $\mathcal{T}$ a tile tree with $X \in \mathcal{T}$. Assume that we are given indices c and d. Our goal in this section is to find those indices a and b such that $Y = (a, b) \times (c, d)$ is an optimal subtile of X.

Define two vectors, p for positives and n for negatives, each of length $e - s + 1$, that contain the number of 1s and 0s respectively, within the ith row of X, $p_i = |\{(i + s - 1, w) \in \mathit{area}(X) \mid c \leq w \leq d, D(i + s - 1, w) = 1\}|$, and $n_i = |\{(i + s - 1, w) \in \mathit{area}(X) \mid c \leq w \leq d, D(i + s - 1, w) = 0\}|$.

This allows us to define $\mathit{cnt}(a, b; p) = \sum_{i=a}^{b} p_i$ (and similarly for n). Let $u = \mathit{cnt}(a, b; p)$ and $v = \mathit{cnt}(a, b; n)$. It follows that $p(Y; \mathcal{T} + X \to Y) = u$ and $n(Y; \mathcal{T} + X \to Y) = v$, where $Y = (a, b) \times (c, d)$; those are the entries of X now to be encoded by Y. Let us define $\mathit{cost}(a, b; p, n, o, z) = L(u, v) + L(o - u, z - v)$. We will write $\mathit{cost}(a, b)$, when p, n, o, z are known from the context. Proposition 1 states that minimising $L(D, \mathcal{T}')$ is equivalent to minimising $\mathit{cost}(a, b)$.

Further, let us define $\mathit{fr}(a, b; p, n) = \mathit{cnt}(a, b; p) / (\mathit{cnt}(a, b; p) + \mathit{cnt}(a, b; n))$ to be the *frequency* of 1s within Y. Proposition 1 then allows us to formulate the optimisation problem as follows.

Problem 3 (Minimal Border Points). Let p and n be two integer vectors of the same length, m. Let o and z be two integers such that $\mathit{cnt}(1, m; p) \leq o$ and $\mathit{cnt}(1, m; n) \leq z$. Find $1 \leq a \leq b \leq n$ such that $\mathit{fr}(a, b; p, n) > o/(o + z)$ and that $\mathit{cost}(a, b)$ is minimised.

The rest of the section is devoted to solving this optimisation problem. Naively we could test every pair (a, b), which however requires quadratic time. Our approach is to ignore a large portion of suboptimal pairs, such that our search becomes linear.

To this end, let p and n be two vectors, and let $1 \leq b \leq |p|$ be an integer. We say that $a \leq b$ is a *head border* of b if there are no integers i and j such that $1 \leq i < a \leq j \leq b$ and $\mathit{fr}(i, a - 1) \geq \mathit{fr}(a, b)$. Similarly, we say that $b \geq a$ is a *tail border* of a if there are no indices $a \leq i \leq b < j \leq |p|$ such that $\mathit{fr}(i, b) \leq \mathit{fr}(b + 1, j)$. We denote the list of all head borders by $\mathit{bh}(b, p, n)$ and the list of all tail borders by $\mathit{bt}(a, p, n)$.

Given a head border a of b, we say that a is a *head candidate* if there are no indices $1 \leq i < a \leq j \leq b$ such that $\mathit{fr}(i, a - 1) \geq \mathit{fr}(j, b)$. Similarly, we say that $b \in \mathit{bt}(a)$ is a *tail candidate* of a if there are no indices $a \leq i \leq b < j \leq |p|$ such that $\mathit{fr}(a, i) \leq \mathit{fr}(b + 1, j)$. We denote the list of all head candidates by $\mathit{ch}(b, p, n)$ and the list of all tail candidates by $\mathit{ct}(a, p, n)$.

To avoid clutter, we do not write p and n wherever clear from context.

As an example, consider Figure 3(a). Since $\mathit{fr}(a_2, a_3 - 1) > \mathit{fr}(a_3, a_4 - 1)$, it follows that $a_3 \notin \mathit{bh}(a_4 - 1)$. Note that $a_4 \in \mathit{bh}(a_6 - 1)$ but $a_4 \notin \mathit{ch}(a_6 - 1)$ since $\mathit{fr}(a_2, a_4 - 1) > \mathit{fr}(a_5, a_6 - 1)$.

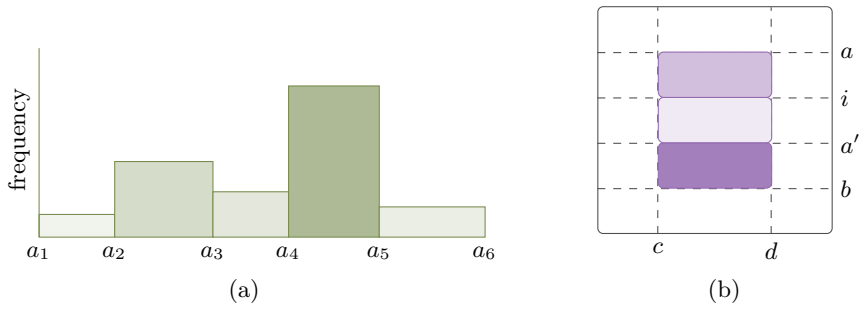


Fig. 3. Example of how FindTile considers head candidates. (a) a_3 is no head border for a_4-1, as $fr(a_2, a_3-1) > fr(a_3, a_4-1)$. Although a head border for a_6-1, a_4 is not a head candidate for a_6-1, as $fr(a_3, a_4-1) > fr(a_5, a_6-1)$. (b) Proposition 2 states that we can ignore i as head candidate for a tile to j, as by $fr(u', i-1) > fr(i, u-1)$ we know $fr(u', j-1) > fr(i, j-1)$.

We are now ready to state the main result of this section: in order to find the optimal tile we need to only study head and tail candidates.

Proposition 2. *Let p and n be two vectors and let o and z be two integers. Let $i \leq j$ be two indices such that $fr(i,j) > o/(o+z)$. Then there are $a \leq b$ such that $cost(a,b) \leq cost(i,j)$, $a \in ch(b)$ and $b \in ct(a)$.*

Proposition 2 is illustrated in Figure 3(b). Since $fr(a, i-1) \geq fr(i, a')$, we know that $i \notin ch(b)$. Proposition 2 implies that we can safely ignore (i,b) and consider instead (a',b) or (a,b).

Proposition 2 states that we need to only study candidates, a subset of borders. Fortunately, there exists an efficient algorithm to construct a border list $bh(b+1)$ given the existing list $bh(b)$ [3]. The approach relies on several lemmata.

Let $(a_1, \ldots, a_K) = bh(b)$. We claim that $bh(b+1) \subseteq (a_1, \ldots, a_K, b+1)$.

Lemma 1. *If $a \leq b$ and $a \notin bh(b)$, then $a \notin bh(b+1)$.*

Proof. By definition there are i and j such that $1 \leq i < a \leq j \leq b$ such that $fr(i, a-1) \geq fr(a,j)$. These indices are valid for $b+1$, hence $a \notin bh(b+1)$. □

Hence, in order to construct $bh(b+1)$, we only need to delete entries from $(a_1, \ldots, a_K, b+1)$. Let us define a *head frequency* $hfr(b) = \max_{i \leq b} fr(i,b)$ and a *tail frequency* $tfr(a) = \max_{a \geq i} fr(a,i)$. The following two lemmata say that the last entry in $bh(b+1)$ has to be the smallest index j such that $fr(j,b) = hfr(b)$, and that the borders of $b+1$ smaller than j are all included in $bh(b)$.

Lemma 2. *Let j be the smallest index s.t. $fr(j,b) = hfr(b)$. Then $j = \max bh(b)$. Let j be the largest index s.t. $fr(a,j) = tfr(a)$. Then $j = \min bt(a)$.*

Proof. First note that $j \in bh(b)$. Let i be an index $j < i \leq b$. We have $fr(i,b) \leq fr(j,b)$ which implies that $fr(j, i-1) \geq fr(i,b)$. This implies that $i \notin bh(b)$. The case for $bt(a)$ is similar. □

Algorithm 2: SCAN(p, n, o, z)

```
   input  : integer vectors p and n, number of 1s (0s) in the parent tile, o (z)
   output : an interval t solving Problem 3
 1 best ← ∞; t ← (0, 0); B ← C ← ∅;
 2 foreach b = 1, ..., |p| do
 3     push b to the front of B;
 4     push b to the front of C;
 5     while |B| > 1 and fr(B_1, b; p, n) ≤ fr(B_2, B_1 − 1; p, n) do
 6         if B_1 = C_1 then remove C_1;
 7         remove B_1;
 8     while |C| > 1 and fr(C_2, C_1 − 1; p, n) ≥ tfr(b + 1) do
 9         c ← cost(C_1, b);
10         if c < best then t ← (C_1, b); best ← c;
11         remove C_1;
12     c ← cost(C_1, b);
13     if c < best then t ← (C_1, b); best ← c;
14 return t;
```

Lemma 3. *Let $a \in bh(b)$. Let k be the smallest index such that $fr(k, b+1) = hfr(b+1)$. If $a < k$, then $a \in bh(b+1)$.*

Proof. Assume that $a \notin bh(b+1)$, that is, there are i and j such that $1 \leq i < a \leq j \leq b+1$ such that $fr(i, a-1) \geq fr(a, j)$. We must have $j = b+1$. Otherwise $a \notin bh(b)$. Note that $fr(a, b+1) < fr(k, b+1)$, which implies $fr(a, k-1) < fr(a, b+1)$. Since $k - 1 \leq b$, we have $a \notin bh(b)$, which is a contradiction. □

These lemmata give us a simple approach. Start from $(a_1, \ldots, a_K, b+1)$ and delete entries until you find index k such that $fr(k, b+1)$ is maximal. We will see later in Proposition 3 that we can easily check the maximality.

Unfortunately, as demonstrated in [3] there can be $\Theta(b^{2/3})$ entries in $bh(b)$. Hence, checking every pair will not quite yield a linear algorithm. In order to achieve linearity, we use two additional bounds. Consider Figure 3(a). We have $bh(a_5 - 1) = (a_1, a_2, a_4)$. First, since $tfr(a_5) = fr(a_5, a_6 - 1) > fr(a_1, a_2 - 1)$, we have $a_5 - 1 \notin ct(a_1)$. Consequently, we do not need to check the pair $(a_1, a_5 - 1)$. Secondly, we know that for any $k \geq a_5$ we have $fr(a_5, k) \leq tfr(a_5) \leq fr(a_3, a_4 - 1)$. Hence, $a_4 \notin ch(k)$ and we can ignore a_4 after we have checked $(a_4, a_5 - 1)$. We can now put these ideas together in a single algorithm, given as Algorithm 2, and which we will refer to as the SCAN algorithm.

Proposition 2 stated that it is enough to consider intervals where then end points are each other candidates. The next proposition shows that SCAN actually tests all such pairs. Consequently, we are guaranteed to find the optimal solution.

Proposition 3. *Let p and n be count vectors and let o and z be two integers.* SCAN(p, n, o, z) *tests every pair (a, b) where $a \in ch(b)$ and $b \in ct(a)$.*

To show this, we first need the following lemma.

Lemma 4. *Let* $(a_1, \ldots, a_L) = bh(b)$. *Then* $fr(a_{k-1}, a_k - 1) < fr(a_k, a_{k+1} - 1)$.

Proof. Assume that $fr(a_{k-1}, a_k - 1) \geq fr(a_k, a_{k+1} - 1)$. Then $a_k \notin bh(b)$. □

By which we can proceed with the proof for Proposition 3.

Proof. Let us first prove that B at bth step is equal to $bh(b)$. We prove this using induction. The case $b = 1$ is trivial and assume that the result hold for $b - 1$. Let $(a_1, \ldots, a_L) = bh(b - 1)$. Lemma 1 implies that $bh(b) \subseteq (a_1, \ldots, a_L, b)$.

Assume that $fr(b, b) > fr(a_L, i - 1) = hfr(b - 1)$. By definition, $b \in bh(b)$. Lemma 2 implies that b is the smallest index k for which $fr(k, b) = hfr(b)$. Lemma 3 now states that $bh(b) = (a_1, \ldots, a_L, b)$ which is exactly what we get since the while loop on Line 5 is not executed.

Assume that $fr(b, b) \leq fr(a_L, i - 1)$. Then $b \notin bh(b)$ and indeed it is deleted in the first run of the while loop (Line 5). Let $a_k \in bh(b)$ be the first entry in B after the while loop has finished. Let $a_l \in bh(b)$ be the smallest index for which $fr(a_l, b) = hfr(b)$. We claim that $k = l$. If $l > k$, then $fr(a_l, b) \leq fr(a_{l-1}, a_l - 1)$ which implies that $fr(a_l, b) \leq fr(a_{l-1}, b)$ which is a contradiction. Assume that $l < k$. By definition of k, we have $fr(a_{k-1}, a_k - 1) < fr(a_k, b)$. Lemma 4 implies that $fr(a_l, a_k - 1) \leq fr(a_{k-1}, a_k - 1)$. Hence, $fr(a_l, b) < fr(a_k, b)$, which is a contradiction. Consequently, $k = l$. Lemma 3 now states that $bh(b) = (a_1, \ldots, a_k)$ which is exactly what we have.

Now that we have proved that B at bth step is equal to $bh(b)$. Let us consider the list C. Let $a \in B \setminus C$. This means that a was deleted during some previous round, say $k < i$, and that there is j such that $fr(j, a - 1) \geq tfr(k + 1) \geq fr(k + 1, b)$. Hence a is not a head candidate of b. Consequently, all head candidates of b are included in C at bth step.

Not all entries of C are tested during the bth step. Assume that we have completed bth step and C_k is not tested ($k > 1$). Since C is a subset of the border list, Lemma 4 implies that $fr(C_k, C_{k-1} - 1) \leq fr(C_2, C_1 - 1) < tfr(b + 1)$. There is j such that $tfr(b + 1) = fr(b + 1, j)$. This implies that b is not a tail candidate for C_k, which completes the proof. □

Let us finish this section by demonstrating the linear execution time $\Theta(|p|)$ of SCAN. To this end, note that we have three while-loops in the algorithm: two inner and one outer. During each iteration of the first inner loop we delete an entry from B, a unique number between 1 and $|p|$. Consequently, the *total* number of times we execute the first inner loop is $|p|$, at maximum. Similarly, for the second inner loop. The outer loop is executed $|p|$ times. Next, note that there are two non-trivial subroutines in the algorithm. First, on Lines 6 and 9, we need to compute frequencies. This can be done in constant time by, e.g. precomputing $c_j = \sum_{i=1}^{j} p_i$, and then using the identity $cnt(i, j; p) = c_j - c_{i-1}$. Secondly, on Line 9, we need to compute $tfr(b)$. We can precompute this in linear time by using the algorithm given in [3], which involves computing tail borders (equivalent to computing B in SCAN) and applying Lemma 2. This shows that the total execution time for the algorithm is $\Theta(|p|)$.

Algorithm 3: FINDTILE$(X, \mathcal{T}, D)$

input : parent tile $X = (s, e) \times (x, y)$, current tile tree $\mathcal{T}$, dataset D
output : B, a tile optimizing $\mathcal{T} + X \to B$, see Problem 2

$o \leftarrow p(X; \mathcal{T})$; $z \leftarrow n(X; \mathcal{T})$; $B \leftarrow X$;
foreach c and d such that $x \leq c \leq d \leq y$ **do**
    update p and n;
    $(a, b) \leftarrow$ SCAN(p, n, o, z);
    $Y \leftarrow (a + s - 1, b + s - 1) \times (c, d)$;
    if $L(\mathcal{T} + X \to Y, D) < L(\mathcal{T} + X \to B, D)$ **then** $B \leftarrow Y$;
    $(a, b) \leftarrow$ SCAN(n, p, z, o);
    $Y \leftarrow (a + s - 1, b + s - 1) \times (c, d)$;
    if $L(\mathcal{T} + X \to Y, D) < L(\mathcal{T} + X \to B, D)$ **then** $B \leftarrow Y$;
return B;

Now that we have a linear algorithm for discovering an optimal tile given a fixed set of columns, we need an algorithm for discovering the columns themselves. We employ a simple quadratic enumeration given in Algorithm 3. Note that SCAN assumes that the optimal tile is more dense than the background tile, we have to call SCAN twice, once normally to find the optimal dense subtile, and once with ones and zeroes reversed to find the optimal sparse subtile.

The computational complexity of FINDTILE is $\Theta(N^2 M)$ where N is the number of columns and M is the number of rows in the parent tile. However, if M is smaller than N, we can transpose the parent tile and obtain a $\Theta(NM \min(N, M))$ execution time.

5 Related Work

Frequent itemset mining [1] is perhaps the most well-known example of pattern mining. Here, however, we are not just interested in the itemsets, but also explicitly want to know which rows they cover. Moreover, we are not interested in finding all tiles, but aim to find tilings that describe the data well.

Mining sets of patterns that describe a dataset is an actively studied topic [2, 20, 21]. Related in that it employs MDL, is the KRIMP algorithm [21], which proposed the use of MDL to identify pattern sets. Geerts et al. discuss mining large tiles of only 1s [7]. Different from these approaches, our models are hierarchical, and do allow for noise within tiles.

Kontonasios and De Bie discuss ranking a candidate collection of tiles, employing a maximum entropy model of the data to measure the interestingness of a tile [5, 12]. Boolean matrix factorisation [15] can be regarded as a tile mining. The goal is to find a set of Boolean factors such that the Boolean product thereof (essentially tiles of only 1s) approximates the dataset with little error. Similarly, bi-clustering can be regarded as a form of tiling, as it partitions the rows and columns of a dataset into rectangles [18]. Compared to these approaches, a ma-

jor difference is that we focus on easily inspectable hierarchical models, allowing nested refinements within tiles.

Most closely related to STIJL is the approach by Gionis et al. [8], who proposed mining hierarchies of tiles, and gave a randomised heuristic for finding good subtiles. We improve over this approach by formally defining the problem in terms of MDL, employing a richer modelling language in the sense that it allows tiles with the same parent to overlap, introducing a deterministic iterative any-time algorithm, that given a tile tree efficiently finds the optimal subtile. Since the approach by Gionis et al. [8] does not use MDL as a stopping criterion, it is not possible to compare both methods directly. In principle, it is possible to adopt their search strategy to our score. A fair comparison between the two search strategies, however, is not trivial since the randomised search depends on a parameter, namely the number of restarts. This parameter acts as a trade-off between the expected performance and execution time. Choosing this parameter is difficult since there are no known bounds for the expected performance.

Our approach for discovering optimal subtiles is greatly inspired by the work of Calders et al. [3] in which the goal was to compute the head frequency, $hfr(i)$, given a stream of binary vectors.

As STIJL is an iterative any-time algorithm, and hence iterative data mining approaches are related. The key idea of these approaches is to iteratively find the result providing the most novel information about the data with respect to what we already know [5, 11, 14]. Here, we focus on hierarchical tiles, and efficiently find the locally optimal addition.

6 Experiments

In this section we empirically evaluate our approach. We implemented our algorithms in C++, and provide the source code, along with the synthetic data generator.[3] All experiments were executed single-threaded on Linux machines with Intel Xeon X5650 processors (2.66GHz) and 12 GB of memory.

We use the shorthand notation $L\%$ to denote the compressed size of D with the tile tree $\mathcal{T}$ as discovered by STIJL relative to the most simple tree $\mathcal{T}_0$, $\frac{L(D,\mathcal{T})}{L(D,\mathcal{T}_0)}\%$, wherever D and $\mathcal{T}$ are clear from context.

We do not compare to the naive strategy of finding optimal subtiles as $\Theta(N^2M^2)$ execution time is impractical even for very small datasets.

Datasets We evaluate our measure on one synthetic, and four publicly available real world datasets. The 240-by-240 synthetic dataset *Composition* was generated to the likeness of the famous Mondrian painting 'Composition II in Red, Blue, and Yellow', where we use different frequencies of 1s for each of the colours. *Abstracts* contains the abstracts of papers accepted at ICDM up to 2007, for which we take the words with a frequency of at least 0.02 after stemming and removing stop words [5]. The *DNA* amplification data contains DNA copy number amplifications. Such copies are known to activate oncogenes and are

[3] `http://adrem.ua.ac.be/stijl/`

Table 1. Results of STIJL on five datasets. Shown are, per dataset, number of rows and columns, overall density, and for resp. without and with overlap, the relative compression $L\%$ (lower is better), number of discovered tiles, and wall-clock runtime.

				Disjoint			**Overlap**		
Dataset	N	M	%1s	$L\%$	$\lvert\mathcal{T}\rvert$	*time*	$L\%$	$\lvert\mathcal{T}\rvert$	*time*
Composition	240	240	23.2	81.72	8	57s	81.58	7	1m23s
Abstracts	859	541	6.6	89.59	14	16m03s	89.54	14	27m54s
DNA Amp.	4 590	391	1.5	61.91	466	334m	61.61	446	625m
Mammals	2 183	121	20.5	54.69	55	1m37s	54.62	50	3m06s
Paleo	501	139	5.1	80.23	14	39s	79.07	13	1m22s

the hallmarks of nearly all advanced tumours [17]. The *Mammals* presence data consists of presence records of European mammals[4] within geographical areas of 50×50 kilometers [16]. Finally, *Paleo* contains information on fossil records[5] found at specific palaeontological sites in Europe [6].

We give the basic properties of these datasets in Table 1. To obtain good orders for the real world datasets, we applied SVD, that is, we ordered items and transactions based on first left and right eigenvectors.

Synthetic Data As a sanity check, we first investigate whether STIJL can reconstruct the model for the *Composition* data. We ran experiments for both the disjoint and the overlapping tile settings. With the latter setup we perfectly capture the underlying model in only 7 tiles. We show the data and discovered model as Figure 4(a); each of the rectangles in the painting are represented correctly by a tile, including the crossing vertical and horizontal bars. When we require disjoint tiles, the fit of the model is equally good, however the model requires one additional tile to model the crossing bars.

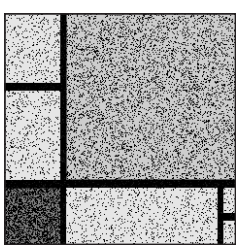
Quantitative Analysis Next, we consider quantitative results on the real datasets. We run STIJL both for disjoint and overlapping tiles. Table 1 gives the relative compressed size $L\%$, the number of tiles in the returned tile trees, and the wall-clock time it took to find these models.

These results show STIJL finds trees that summarise the data well. The relative compressed sizes tell the data is described succinctly, while the tile trees remain small enough to be considered by hand; even for *DNA*, by the hierarchical nature of the model, the user can quickly read and understand the model.

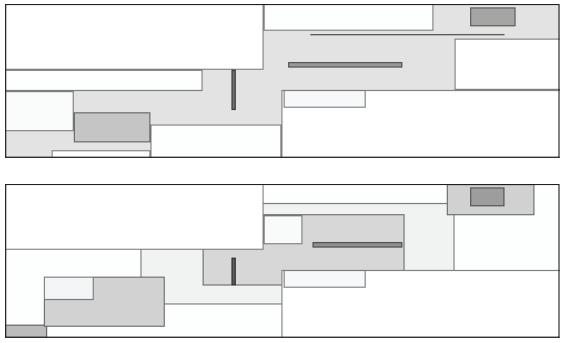
For *Mammals* we see that whereas the baseline model $\mathcal{T}_0$ requires 193 315 bits, the STIJL model with overlapping tiles only requires 105 589 bits. Note that, as the total compressed size essentially is the log-likelihood of the model and the data, a gain of a single bit corresponds to a twice as likely model.

In all experiments, allowing overlap results in better models. Not only do they give more succinct data descriptions, the discovered tile trees are also sim-

[4] Available for research purposes: `http://www.european-mammals.org`
[5] NOW public release 030717 available from [6].



(a) Composition



(b) Paleo (transposed)

Fig. 4. Results of STIJL on (a) *Composition* and (b) *Paleo*, with (top) the disjoint hierarchical tiling, and (bottom) the tiling allowing overlap within the same parent tile. For *Paleo* we do not show individual 1s. Darker tiles correspond to higher frequency

pler, requiring fewer tiles to do capture the structure of the data. By allowing overlap, the search space is expanded, and hence more computation is required: on average, in our experiments, twice as much.

On these datasets, the current STIJL implementation requires from seconds up to a few hours of runtime. By its iterative any-time nature, users, however, can already start to explore models while in the background further refinements are calculated.

Qualitative Analysis Next, we investigate the discovered models in more detail. To this end, we first use the *Paleo* data as by its modest size it is easily visually representable. In Figure 4(b) we show the result of STIJL on this data, with the top figure the result of allowing only disjoint tiles, and in the bottom figure when allowing overlap. Darker toned tiles correspond to more dense areas of the data. For clarity, we here do not show the individual 1s (as we did in Fig. 2, which corresponds to the bottom plot of Fig. 4(b)).

The first thing we note, is that the two results are quite alike. The model with overlap, however, is a bit simpler and 'cleaner': the relatively dense areas are of the data are easier to spot for this model, than for the disjoint one. Second, it uses the hierarchical property as intended: in the top right corner, for instance, we see a dense, dark-grey tile within a lighter tinted square, within a very sparse tile. While for reasons of space we can only show these examples, these are observations that hold in general—by which it may come at no surprise that by allowing overlap we obtain better MDL scores.

Next, we inspect the results on *Abstracts*. This sparse dataset has no natural order by itself, and when we apply SVD to order it, we find most of the 1s are located in the top-left corner of the data. When we apply STIJL, we see it correctly reconstructs this structure. Due to lack of space, however, we do not give the visual representation. Instead, we investigate the most dense tile, which covers the top-left corner. We find that it includes frequent words that are

often used in conjunction in data mining abstracts, including *propose*, *efficient*, *method*, *mine*, and *algorithm*. Note that, by design, STIJL gives a high level view of the data; that is, it tells you where the ones are, not necessarily their associations. Extending it to recognise structure within tiles is future work.

7 Discussion

The experiments show STIJL discovers succinct tile trees that summarise the data well. Importantly, the discovered tile trees consist of only few tiles, and are even easier inspected by the hierarchical property of our models.

The complexity of STIJL is much lower than that of the naive locally optimal approach; as with $\Theta(NM \min(N, M))$ its complexity is only squared in the smallest dimension of the data. However, for datasets with both many rows and columns, runtimes may be non-trivial. STIJL, however, does allow ample opportunity for optimisation. FINDTILE, for instance, can be trivially run in parallel per parent tile, as well as over a and b.

As there is no such thing as a free lunch, we have to note that MDL is no magic wand. In the end, constructing an encoding involves choices—choices one can make in a principled manner (fewer bits is better), but choices nevertheless. Here, our choices were bounded by ensuring optimality of FINDTILE. As such, we currently ignore globally optimal encoding solutions, such as achievable by maximum entropy modelling [5]. Although we could so obtain globally optimality of the encoding, the effects of adding a tile become highly unpredictable, which would break the locally optimal search of FINDTILE.

We assume the rows and columns of the data to be ordered. That is, in the terminology of [8], we are interested in geometric tiles. Although [6, 8] showed good geometric tilings can be found on spectrally ordered data, it would make for engaging research to investigate whether we can find good orderings on the fly, that is while we are tiling, ordering the data such that we optimise our score.

8 Conclusion

We discussed finding good hierarchical tile-based models for binary data. We formalised the problem in terms of MDL, and introduced the STIJL algorithm for greedily approximating the score on binary data with ordered rows and columns. For unordered data, spectral techniques can be used to find good orders [8]. We gave the FINDTILE procedure for which we proved it finds the locally optimal tile in $\Theta(NM \min(N, M))$.

Experiments showed STIJL discovers high-quality tile trees, providing succinct description of binary data. Importantly, by their hierarchical shape and small size, these models are easily interpreted and analysed by hand.

Future work includes optimising the encoded cost by mining tiles and orders at the same time, as opposed to using ordering techniques oblivious to the target.

Acknowledgements

Nikolaj Tatti and Jilles Vreeken are supported by Post-Doctoral Fellowships of the Research Foundation – Flanders (FWO).

References

1. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB*, pages 487–499, 1994.
2. B. Bringmann and A. Zimmermann. The chosen few: On identifying valuable patterns. In *ICDM*, pages 63–72, 2007.
3. T. Calders, N. Dexters, and B. Goethals. Mining frequent itemsets in a stream. In *ICDM*, pages 83–92. IEEE, 2007.
4. T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience New York, 2006.
5. T. De Bie. Maximum entropy models and subjective interestingness: an application to tiles in binary databases. *Data Min. Knowl. Disc.*, 23(3):407–446, 2011.
6. M. Fortelius, A. Gionis, J. Jernvall, and H. Mannila. Spectral ordering and biochronology of european fossil mammals. *Paleobiology*, 32(2):206–214, 2006.
7. F. Geerts, B. Goethals, and T. Mielikäinen. Tiling databases. In *DS*, pages 278–289, 2004.
8. A. Gionis, H. Mannila, and J. K. Seppänen. Geometric and combinatorial tiles in 0-1 data. In *PKDD*, pages 173–184. Springer, 2004.
9. P. Grünwald. *The Minimum Description Length Principle*. MIT Press, 2007.
10. J. Han, H. Cheng, D. Xin, and X. Yan. Frequent pattern mining: Current status and future directions. *Data Min. Knowl. Disc.*, 15, 2007.
11. S. Hanhijärvi, M. Ojala, N. Vuokko, K. Puolamäki, N. Tatti, and H. Mannila. Tell me something I don't know: randomization strategies for iterative data mining. In *KDD*, pages 379–388. ACM, 2009.
12. K.-N. Kontonasios and T. De Bie. An information-theoretic approach to finding noisy tiles in binary databases. In *SDM*, pages 153–164. SIAM, 2010.
13. M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications*. Springer, 1993.
14. M. Mampaey, N. Tatti, and J. Vreeken. Tell me what I need to know: Succinctly summarizing data with itemsets. In *KDD*, pages 573–581. ACM, 2011.
15. P. Miettinen, T. Mielikäinen, A. Gionis, G. Das, and H. Mannila. The discrete basis problem. *IEEE TKDE*, 20(10):1348–1362, 2008.
16. A. Mitchell-Jones, G. Amori, W. Bogdanowicz, B. Krystufek, P. H. Reijnders, F. Spitzenberger, M. Stubbe, J. Thissen, V. Vohralik, and J. Zima. *The Atlas of European Mammals*. Academic Press, 1999.
17. S. Myllykangas, J. Himberg, T. Böhling, B. Nagy, J. Hollmén, and S. Knuutila. DNA copy number amplification profiling of human neoplasms. *Oncogene*, 25(55):7324–7332, 2006.
18. R. G. Pensa, C. Robardet, and J.-F. Boulicaut. A bi-clustering framework for categorical data. In *PKDD*, pages 643–650. Springer, 2005.
19. N. Tatti. Are your items in order? In *SDM*, pages 414–425. SIAM, 2011.
20. N. Tatti and H. Heikinheimo. Decomposable families of itemsets. In *ECML PKDD*, pages 472–487. Springer, 2008.
21. J. Vreeken, M. van Leeuwen, and A. Siebes. KRIMP: Mining itemsets that compress. *Data Min. Knowl. Disc.*, 23(1):169–214, 2011.