Universiteit Leiden
The Netherlands

Opleiding Informatica

Can a human-computer hybrid outperform the LTM tiling algorithm?

Floyd Remmerswaal

Supervisors:
Matthijs van Leeuwen & Lincen Yang

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

19/07/2021

Abstract

Data mining is becoming increasingly important in our world. In a wide range of fields, understanding the constantly growing amount of gathered data is essential. For mining binary data sets specifically, the concept of tiles was introduced as a potentially interesting type of patterns. A tile represents some knowledge about the data set it is a part of. Finding the minimum tiling of a data set, i.e., a tiling that covers the entire data set with the smallest number of tiles used, is a problem that has been proved to be $\mathcal{NP}$-hard. Nonetheless, efforts have been made to design algorithms that find good tilings. The *Large Tile Mining* (LTM) algorithm was introduced to mine large tiles in binary databases. LTM can be used to construct a tiling by iteratively mining the largest tile from a given database. The resulting tiling is often close to the minimum tiling, but improvements can still be made. Previous research has shown that human-computer interaction can improve data mining performance. We try to tackle the minimum tiling problem by letting humans cooperate with LTM to produce a better tiling than can be achieved by only using LTM. To this end, we have constructed a web application that allows users to construct a tiling, while getting suggestions from LTM. The experiment results show that a small percentage of people manage to produce a better tiling than the tiling produced by iteratively using LTM. We also discuss the drawbacks of this approach, mainly the problem of human attention span and the issue that this approach is unlikely to scale well.

Contents

1 Introduction

Gathering, analyzing and interpreting data is becoming increasingly important in our world. Our understanding of a wide variety of topics, including, for example, medical research, has been greatly improved in recent times, thanks in part to new data mining techniques and algorithms. Binary data sets are an interesting target for data mining purposes, as many types of data can be represented as a binary matrix. For example, graphs can be described as a binary adjacency matrix. To properly mine the ever increasing amount of data, having a good interestingness measure is paramount. For binary data sets, the concept of tiles has been introduced, with the area of a tile as an interestingness measure [GGM04]. A tile can be informally defined as a rectangle of '1' fields in a binary data set. A small number of tiles that cover a large part of a data set give a great insight into that data set itself.

In this thesis, we are interested in the *Minimum Tiling* problem, which asks for a tiling consisting of as few tiles as possible. Finding a minimum tiling for a data set is interesting, as that minimum tiling describes the entire data set and highlights relations within the data. However, determining a minimum tiling of a binary data set has been proved to be $\mathcal{NP}$-hard. Efforts have been made to tackle this problem nonetheless, resulting in an algorithm called *Large Tile Mining* (LTM) that finds large tiles in a data set. This algorithm can be used to construct a tiling by iteratively mining the tile that covers the largest number of cells that are not yet covered by another tile. This method results in a fairly good tiling, but improvements are still possible.

In other data science fields, previous research has shown that involving a human data miner in the mining process can improve the results of that process [Hol13]. In this thesis, we aim to use that knowledge to address the minimum tiling problem, by combining a human data miner with the LTM algorithm, to see if, together, they can produce a better tiling than the one constructed by repeatedly mining the largest remaining tile LTM can find. Repeatedly using LTM to find large tiles is a greedy strategy that might lead to suboptimal tilings. A human data miner can look ahead and decide to choose a smaller tile, which can result in a smaller tiling than the one produced by the greedy strategy. Furthermore, if a domain expert is involved, he or she may have prior knowledge that enables them to discard suboptimal suggestions by LTM.

To investigate if this human-computer hybrid approach performs better than LTM alone, a web application has been implemented that allows a user to construct a tiling for a data set, while receiving suggestions from LTM at every step of the process. We are the first to add a human data miner to the tiling process in this way. The experiment we have conducted using this web application shows that it is possible for humans to outperform LTM. Four of the 93 participants managed to create a tiling that consisted of one tile less than the one that LTM produces. Our results, which are discussed in more detail in Section 7, show that adding a human data miner can in fact improve the tiling process. There are, however, some caveats, as only a small number of participants actually managed to beat LTM. Furthermore, the average participant spent about 7 minutes on constructing the tiling, whereas LTM takes only a fraction of a second.

1.1 Thesis overview

This bachelor thesis is supervised by Dr. Matthijs van Leeuwen & Lincen Yang (MSc) of the Leiden Institute of Advanced Computer Science. This chapter contains the introduction and is followed by Section 2 which discusses related works. These related works concern different tiling methods, general human computation work and data mining ideas. In Section 3, we introduce the definitions used throughout this thesis. Then, we explain the problem we are trying to solve, the LTM algorithm, and the pitfall of using this algorithm for this problem in Section 4. Section 5 states the requirements that an application should satisfy before it can be used to research our research question. The application that was constructed to satisfy the previously stated requirements is shown and discussed in Section 6. In Section 7 we present the results of our research and discuss them. Finally, we present the conclusion of this thesis and propose further research in Section 8.

2 Related Work

Floris Geerts, Bart Goethals, and Taneli Mielikäinen wrote the paper [GGM04] that was the primary inspiration for this research. In [GGM04], Geerts et al. prove that tiling is $\mathcal{NP}$-hard. In addition, they introduce the Large Tile Mining (LTM) algorithm. This algorithm uses a greedy branch-and-bound strategy to solve the problem of finding large tiles in 0/1 databases. This algorithm is used to solve the *Minimum Tiling* problem. They also implemented a few pruning techniques, which have not been used in this thesis. The most important parts of the paper for this thesis are discussed in more detail in Section 4.2. In this thesis, we try to solve the minimum tiling problem with fewer tiles than can be achieved by iteratively using LTM, by adding a human data miner to the process. Combining human computation and tiling algorithms has not been researched before. Research on other data mining techniques indicates that a human-computer combination can result in better performance than algorithms can on their own [Hol13]. Some other approaches to tiling have been undertaken, which we will discuss in this section.

One of these approaches to tiling is in making a distinction between geometric and combinatorial tiles [GMS04]. As implied by their names, combinatorial tiles are in a way combinations of geometric tiles. A tile is called *geometric* when its rows and columns are contiguous. If this is not the case, and the tile rows and columns denoting the tile are arbitrary subsets of the rows and columns of the data set, then these tiles are called *combinatorial*. Another significant difference in their approach is that their tiles are *hierarchical* and *probabilistic*. In their model, basic tiles (also called noisy tiles) include the probability of finding a '1' inside the cells of the tile. This notion is extended by a hierarchy of such basic tiles, where each hierarchical tile consists of a basic tile and a set of exception tiles. Exception tiles are sub-tiles of that basic tile with a different probability of finding a '1' in their cells.

These ideas are built upon by Tatti et al. [TV12] using Tile Trees. Tile Trees are hierarchical tiles with a *list* of tiles as children instead of a set. This difference is introduced because the order of the tiles matters in this case. Moreover, the root tile always completely covers the data set. An algorithm called *Stijl* is introduced that finds, using a greedy approximation, good tile trees. The tile trees that are generated by this algorithm are claimed to be easy to interpret and analyze by hand.

[GMS04] and [TV12] are both mainly concerned with geometric tiles instead of combinatorial tiles. In our research, we are interested in combinatorial tiles. Geometric tiles are a subset of combinatorial tiles, therefore interesting patterns may be discovered by combinatorial tiles that cannot be found using geometric tiles. However, [GMS04] prove that using spectral ordering methods one can find orderings on which good combinatorial tiles become geometric tiles.

De Bie et al. considered interestingness measures for databases, including binary databases like the ones that we research. In [DB11], they argue that *subjective* interestingness measures have some practical benefits over objective interestingness measures, as the subjective ones take into account prior knowledge of the data miner. The interestingness measure introduced in [GGM04], i.e., the area of a tile, is an objective one: only taking into account the database and (statistical) relations between the data.
So far, we have discussed different approaches to defining tiles, but an interesting aspect of our research is that it combines this technical side with the field of Human Computation. Human computation is often defined as: "a paradigm for utilizing human processing power to solve problems that computers cannot yet solve" [QB11]. In our case, the problem is solving the minimum tiling problem with fewer tiles than LTM. In [QB11], Quinn et al. introduce a general framework to relate Human Computation systems to. In their framework, our research would have the following values:

- Motivation: Enjoyment
- Quality control: Automatic Check
- Aggregation: *not applicable*
- Human skill: Visual recognition
- Process order: Computer-Worker-Requester is the best fit
- Task request cardinality: One-to-one

[QB11] states that any Human Computation system cannot be a Data Mining system and vice versa. This would mean that our research cannot be both, however their stance on this matter is largely dictated by their definitions. They argue that Data Mining does not encompass gathering data, while Human Computation necessarily includes just that. In our research, however, the users do not gather data and therefore the authors might not consider it Human Computation. This is a difference of opinion and definition, rather than a fundamental truth.

[Hol13] investigated the combination of Human-Computer Interaction and Knowledge Discovery (of which tiling is a subset). Summarized, their argument is that involving a human in a knowledge discovery project can lead to the discovery of "new, previously unknown insights into data". This is slightly different from our research, as our objective is determining if we can get a better (more efficient) tiling of the data set. This more efficient tiling does not necessarily lead to new insights, although it might help in doing so. Holzinger et al. conclude that combining these two fields might enable users to find and recognize novel data patterns, with the goal of understanding these patterns.

3 Definitions

In this section, we will introduce definitions for key concepts used throughout this thesis.
A **0/1 database** is a matrix consisting of n rows and m columns that only contains the values 0 and 1. Such a database can be seen on the left in Figure 1. Throughout this thesis, we will visualize these databases with a table in which the cells are coloured according to the value that should be in the cell. Black and white cells correspond to '0' and '1' fields, respectively. The result can be seen in the right of Figure 1.

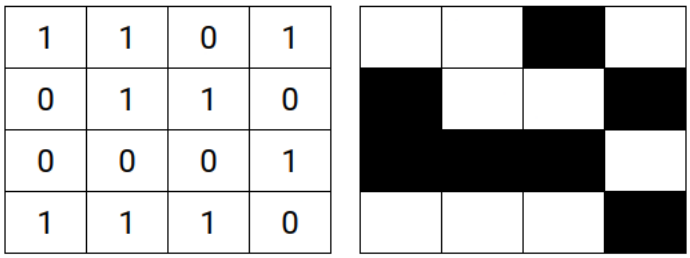


Figure 1: Two visualisations of a 0/1 database.

Such a 0/1 database D consists of a number of rows, also called *transactions*. The names of the columns of this database are called the *items*, denoted by $\mathcal{I}$. A '1' field in the matrix means that the item that the column represents is in that transaction, and a '0' field means that that transaction does not contain that item. A transaction is said to *cover* an item set $I \subset \mathcal{I}$ if it contains all items from that set. The cover of an item set I, written as $cover(I, D)$, is then the set of transactions that cover I in the database D. The size of a database D is equal to the sum of the sizes of all transactions in it. The size of a transaction is just the cardinality of that set. Informally, the size of the database is also equal to the number of '1' fields it contains.
At the core of this thesis is the concept of a **tile**. Informally, a tile is a rectangle of '1' entries in a 0/1 database. Such a tile may not be immediately obvious, as the order of the rows and columns is arbitrary and thus any row or column can be swapped with any other. Formally, we define a tile as follows:
$\tau(I, D) = \{(tid, i) \mid tid \in cover(I, D), i \in I\}$
It constitutes the sets of all combinations (tid, i), where i is an item (or column), and tid is a transaction (or row). In the example in Figure 1, one tile could be the intersection of the first two columns and the first and last row, as visualised in Figure 2.

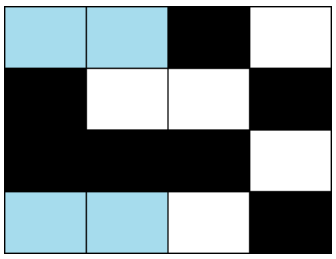


Figure 2: Example of a tile.

The number of cells in a tile is the **area** of that tile. Bigger tiles represent more information about the database of which they are part. The example tile has an area of 4.
A **tiling** of a database is a finite set of tiles that cover all '1' fields in the database. Such a tiling may contain overlapping tiles.

4 Tiling

4.1 The minimum tiling problem

The **minimum tiling problem** is defined by [GGM04] as follows:
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.
[GGM04] proved that this problem is $\mathcal{NP}$-hard, but also that a solution to the minimum tiling problem can be approximated within a factor of $\mathcal{O}(\log nm)$. They prove these statements by first first proving that **Maximum Tile**, the problem of finding the largest tile in a database, is $\mathcal{NP}$-hard. Geerts et al. do this by converting the Maximum Tile instance to an instance of the **Maximum Edge Biclique** problem, which is known to be $\mathcal{NP}$-hard. Because of the correspondence between *Minimum Tiling* and *Maximum Tile*, *Minimum Tiling* is proved to be $\mathcal{NP}$-hard as well.

4.2 The LTM algorithm

LTM is the Large Tile Mining algorithm proposed by [GGM04], which solves the following problem: Given a database D, and a minimum threshold σ, find all maximal tiles that have an area of at least σ. It does so in a recursive manner. Each iteration considers a different possible item set, starting from the item set only including the first column. The pseudo-code for the algorithm can be seen in Algorithm 1. The $Prune(D, \sigma, I)$ function, used in line 2 of the algorithm, does not change the output of the algorithm but only speeds up its execution. This procedure was not implemented in our research, and as such it is not shown here.

Algorithm 1: LTM [GGM04]

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

```
1  T[I] := {};
2  Prune(D, σ, I);
3  forall i occurring in D do
4  |  if |cover({i})|(|I| + 1) ≥ σ then
5  |  |  Add τ(I ∪ {i}) to T[I];
6  |  end
7  |  D^i := {};
8  |  forall j occurring in D such that j > i do
9  |  |  C := cover({i}) ∩ cover({j});
10 |  |  Add(j, C) to D^i;
11 |  end
12 |  Compute T[I ∪ {i}](D^i, σ) recursively;
13 |  Add T[I ∪ {i}] to T[I];
14 end
```

The algorithm has three inputs: the database D, the threshold value σ and the currently considered item set I. D is initially the entire database, and I is initially the empty set. σ is the minimum size threshold, a tile is only mined when its area is at least σ. We can think of the database as consisting of an ordered list of columns (or items), numbered 0 through n. Lines 3-14 of the algorithm step through all items (columns) in the current database (which initially is the entire database). First, a check is done if the I and the currently considered item i can form a tile (lines 4 and 5). If so, it is added to the tiling. A new database is then constructed (lines 7-11), called the *i-conditional database*, denoted by D^i. This database contains all items j for $j > i$ (where j and i refer to the numbers of the columns), and all transactions such that every transaction is in the intersection of the covers of both i and j. The idea behind this strategy is that all large tiles containing item $i \in I$, but not containing any item smaller than i can be found in this i-conditional database. We mine the tiles of this i-conditional database recursively, with $D = D^i$ and $I = I \cup \{i\}$. This is all done in a loop through the items in D. At the last step of every iteration, the tiles found in the recursive calls are added to the tiling.

4.3 Suboptimality of LTM

We use LTM to solve the *Minimum Tiling* problem by iteratively mining the tile T with the largest added value $val(T)$, defined to be the number of cells that T contains that are not yet part of the tiling. The tiling that is constructed in this way may be suboptimal. In this section, we demonstrate that constructing a data set where this happens is fairly trivial. We will show a small example that can easily be expanded.

LTM works well for finding large tiles, but constructing a good tiling is more than just finding the largest tiles in a data set. In Figure 3, a data set is represented as a table, where, as previously discussed, black and white cells should be seen as '0's and '1's in the data set respectively.

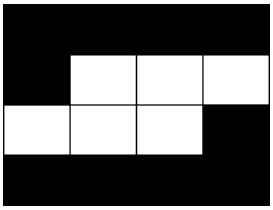


Figure 3: Situation where using LTM is sub-optimal for Minimum Tiling.

Here, picking the largest tile that LTM can find would result in needing two additional (1 by 1 cell) tiles to tile the data set. This is can be seen in Figure 4.

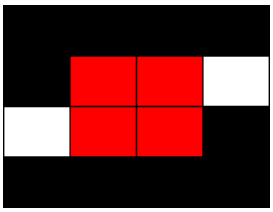


Figure 4: We would need two additional tiles to complete the tiling.

If we opt to pick a smaller tile first, it is possible to tile the remaining cells with just a single tile. This results in a smaller tiling: only two tiles. Of course, the order of these two smaller tiles is irrelevant.

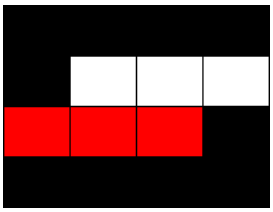


Figure 5: A better first tile.

This example can be expanded when we consider why this happens. Essentially, there are two rows (or columns, the orientation does not matter) with a large intersection of 1s. In addition, these rows both contain one (or perhaps more) '1' that the other row does not have. LTM will then always first mine the large overlap, resulting in needing two more tiles for the cells that are only contained in one of the rows.

An argument could be made that the tile found by LTM contains more information, and perhaps the cells not in the large tile should be treated as outliers. Such considerations, however, are not within the scope of this thesis. The real problem is not necessarily in LTM itself, as LTM is good at its purpose: finding large tiles in a database. The problem arises because we use LTM to iteratively mine the tiles with the most added value in the data set. Sometimes not picking the 'best' tile that LTM can find can result in smaller tilings down the line. This is the core of this research, finding out if humans can select better tiles than LTMs best suggestion.

5 Requirements and design

To conduct the experiment, we need an application that allows us to investigate our research question. This application will have to satisfy some requirements before it can be used. Obviously, to test whether humans can perform better than LTM on its own, we need the application to allow the user to make a tiling for a data set. Moreover, LTM needs to be able to make suggestions. To get a decent number of participants, it must be easy to invite people to join the research.

We discern two classes of requirements, *Must have* and *Should have* requirements. If any of the *Must have* requirements is not satisfied, the built application cannot be used for our purpose. The *Should have* requirements are also important, but the research may be done without them, albeit with some serious drawbacks (e.g. far less potential participants). The following requirements fall in the *Must have* category, as without these our research question cannot be investigated:

- The user must be able to view the data set
- The user must be able to switch rows and columns of the data set
- The user must be able to create a tile and add it to the tiling
- The user must be presented with suggestions for tiles
- LTM must be implemented to suggest tiles to the user
- The application must gather information about the tiling and the user

Then there are some requirements that fall into the *Should have* category. These are important for the success of the study, but without these it might still be possible to do some research (albeit with greater difficulty). These requirements are:

- The application must be accessible on any common desktop operating system
- Inviting participants to use the application should be easy
- The user interface must be intuitive enough for the average person to use

One might argue that mobile devices could potentially greatly increase the number of participants, but supporting mobile devices would introduce its own drawbacks. The user interface would have to be made touch screen friendly, and doing so would cost (a lot) more time. Moreover, a small phone screen has some inherent problems with displaying the data set, and any serious data mining project is very unlikely to be done on a mobile device. Because of these considerations, supporting mobile devices is not a requirement for this research.

Out of these previously mentioned requirements arise some constraints for the layout of the application. As there is a finite amount of screen space, trade offs have to be made about what is and is not implemented in the user interface.The two most vital elements are a visualisation of data set itself and the suggestions by LTM. Because of their importance, a logical position would be at the top of the screen, where they catch the attention of the user immediately.

So far, we have discussed the requirements for the (user-facing) front-end of the application. There are also requirements for the back-end, but there are fewer of them and they are more general. We identify the following (*Must have*) requirements for the back-end:

- The information about the participant and their tiling must be stored in a database
- The database must be remotely accessible

An application that satisfies the requirements mentioned in this section can be used for our research.

6 Implementation

To satisfy the previously stated requirements, a web application was created. A desktop application would give rise to difficulties with distribution to participants. For example, most people would be hesitant to install software from an unknown source (and rightfully so). Another option would be to have participants do the tiling at the university, but that would make getting a large sample size difficult and time consuming. A web application would not have either of those drawbacks and is therefore the preferred method. One might be concerned about the performance of a JavaScript implementation of LTM, compared to the speed of a native desktop application written in (for example) C++. In practice, there was no performance issue at all and mining all tiles (with size > 1) of our constructed data set of 12 by 12 is virtually instantaneous, and that is even without any pruning implemented.

The web app was built using Angular, a TypeScript front-end framework. Using such a framework allows for quick and easy deployment, especially when used in combination with the (NoSQL) database FireStore. Both Angular and FireStore are created by Google. When participants open the web application, they first encounter a landing page with a short survey that asks them to provide their education level and in which age range they fall. They are also asked if they have a background in Computer Science. After this short survey, they are instructed on what is expected from them, and they are shown a small tutorial. Finally, they are tasked with the tiling of a data set.

6.1 The user interface

The interface of the web app consists of four parts: (in order from left to right and top to bottom)

- The data set
- The suggestions by LTM
- The history of tiles added by the user
- A textual description of the task at hand

For a visualization of the web app, see Figure 6.

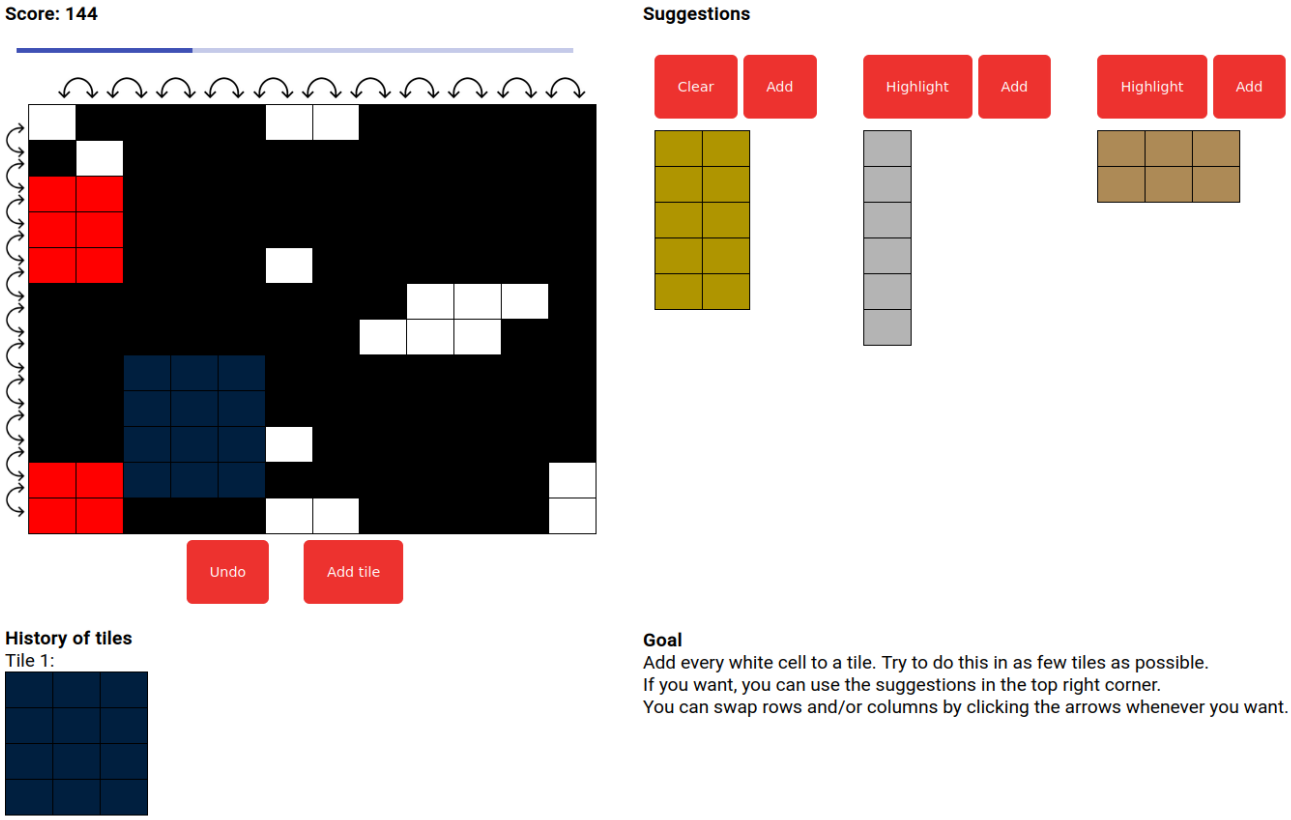


Figure 6: An overview of tiling page of the web application.

The data set is presented as an HTML table, where every cell represents a data point in the database. This simple HTML table is extended by the JavaScript library *js-table-cell-selector* by DarkRiDDeR [Dar21]. This library makes it easy to handle selecting cells from a table. When a user selects cells from the table, it is first checked if all selected cells are '1' cells, and if not the selection is rejected. If the selection passes the check, the user may add the tile to the tiling by clicking the *Add tile* button underneath the table. For removing tiles from the tiling, an *Undo* button was implemented.

Next to the columns and rows are arrows, pointing at two rows or columns. Clicking these arrows swaps the rows or columns pointed at. Tiles that are added to the tiling get a color assigned to them from a predetermined pool of colors. The colors were selected from the palette at [noa17]. In the table, the cells in the tile are given the color of that tile. If a cell that is already in a tile is added to another tile, it retains the color of the original tile.

A score and a progress bar are shown above the tiling table, to encourage the user to perform well. The progress bar indicates what percentage of cells are already tiled, and the score tries to represent a measure of how good the tiling is, where a higher score is better. The score starts at zero, and for every added tile T it is increased by $val(T)^2$, where $val(T)$, as defined earlier, is the number of cells that T contains that are not yet part of the tiling. This value is squared to encourage the user

to find large tiles, as the reward for a single large tile would be greater than for the same cells, divided over multiple tiles.

The suggestions by LTM are recalculated at every step of the tiling process. Initially, all possible tiles are mined (tiles with area ≥ 1). Mining these tiles finishes in a fraction of a second on any reasonable hardware. This list of tiles is updated at every step to contain only tiles that include at least one cell that has not been tiled yet. To determine what tiles will be suggested, we compute for every remaining possible tile their added value $val(T)$ and then sort the tiles based on that value from high to low. The three tiles that score the highest on this metric are then suggested to the user. Ties are determined by the implementation of the JavaScript sort function. These suggestions are displayed from left to right, so the leftmost suggestion is deemed the best. They are colored in gold, silver and bronze to suggest a hierarchy to the user. The user has the possibility of visualising a suggestion by clicking the *Highlight* button above the suggestion. This turns the suggested tile red in the table, as can be seen in Figure 6. The suggestions also have an *Add* button, to immediately add the suggestion to the tiling.

The history of the tiles is straightforward: it contains a visual representation of all tiles that have been added. This list is sorted by 'most recently added' and only serves as an aid to the user. The textual description is a reminder to the participant, as during initial tests of the application, some participants had trouble remembering what exactly their task was.

6.2 Gathered information

A number of data points are recorded for every participant, and sent to the database. This database is a NoSQL database called Firestore, by Google. This database is easy to use in combination with TypeScript, as it is possible to insert JavaScript objects (JSON) into the database. When a participant is done tiling the data set (i.e. their tiling completely covers the data set), all data is automatically sent to the Firestore database. The following information is stored:

- The highest form of education enjoyed by that person
- Whether they have a background in Computer Science or not
- What age bucket they are in
- Their start time
- Their end time
- The score they obtained
- The number of tiles they used
- A string representation of their tiling (JSON string)

Here, the first three points are given by the participants themselves at the start of the experiment. After this step, the participant is shown a small tutorial on what tiles are and what is expected from them. Details of the exact goal of their efforts are not shared with the participants, they are only told that they need to tile the data set in as few tiles as possible, with the possibility of using the suggestions. The start time is a timestamp at the moment that the tiling page loads. The end time is then of course also a timestamp, recorded at the moment the user adds a tile that completes the tiling. The highest form of education is split into the following categories:

- High School or equivalent
- MBO (vocational education) or equivalent
- Bachelor's or equivalent
- Master's or equivalent
- Doctorate or equivalent

The participant is asked to fill in the highest form of education they have enjoyed, not necessarily completed. The background in Computer Science is split into simple 'yes' and 'no' answers. The age buckets available to the participants were the following:

- 12-25
- 26-39
- 40-59
- 60+

After filling out this information, the participant is first shown a small tutorial on the task at hand. They are explained that their task is to colorize all white cells in the table, using rectangles. They are instructed on how to use the interface, specifically the *Add tile* button and the arrows for swapping rows and columns. Secondly, they are given an explanation on the suggestions that they receive during the tiling.

Hosting this web application is done using Firebase. This Google product allows free deployment of Node web applications, up to a reasonable amount of usage. Their free plan was sufficient for this project. The application is then available on the World Wide Web on its own URL based on the project name. For this project those URLs were tiling-scriptie.firebaseapp.com and tiling-scriptie.web.app. Firebase also gives easy access to Firestore, with an easy to use (Angular) API.

Appendix A contains images of all pages of the web application except the actual tiling page. The landing page with the short survey can be seen in Figure 12. The first tutorial page, explaining how to add tiles and how to swap rows and columns, can be seen in Figure 13. The final tutorial page, explaining the undo button and the suggestions by LTM, can be seen in Figure 14. In addition to these pages and the tiling page, there existed also a playground of sorts, located at tiling-scriptie.web.app/playground (and tiling-scriptie.firebaseapp.com/playground). At this playground, people could play the tiling game without their results being sent to the database. That way, people that wanted to try to perfect their tiling could do so without interfering with the research.

7 Experiments

The experiment is conducted by sharing a link to the web application on social media, specifically LinkedIn and Twitter. Also, friends and family were invited to participate. In total, 93 people participated in the experiment. Section 7.2 will discuss the results in more detail, specifically how different groups performed when looking at the number of tiles used for a tiling, and the amount of time they took to produce that tiling. We will refer to the number of tiles that a tiling uses as the *tile count*.

7.1 Experiment Design

The web application described in Sections 5 and 6 was used to conduct the experiment. The data set that the participants were asked to tile was constructed by first starting with an all 0's data set. Then, a number of small, medium, and large tiles were manually added to the data set, where the construction as seen in Section 4.3 is used to ensure that the data set can be more efficiently tiled than always picking the first suggestion. This results in a data set that consists mostly of really obvious tiles, with two places where LTM can be beat. Finally, a small number of arbitrary rows and columns were swapped. Repeatedly picking the first suggestion results in a tiling consisting of 9 tiles. This tiling, constructed in the web application, can be seen in Figure 7. A better tiling is possible, when the pitfall described in Section 4.3 is kept in mind. There are multiple tilings possible that handle this pitfall, one of which can be seen in Figure 8. This tiling uses only 8 tiles.

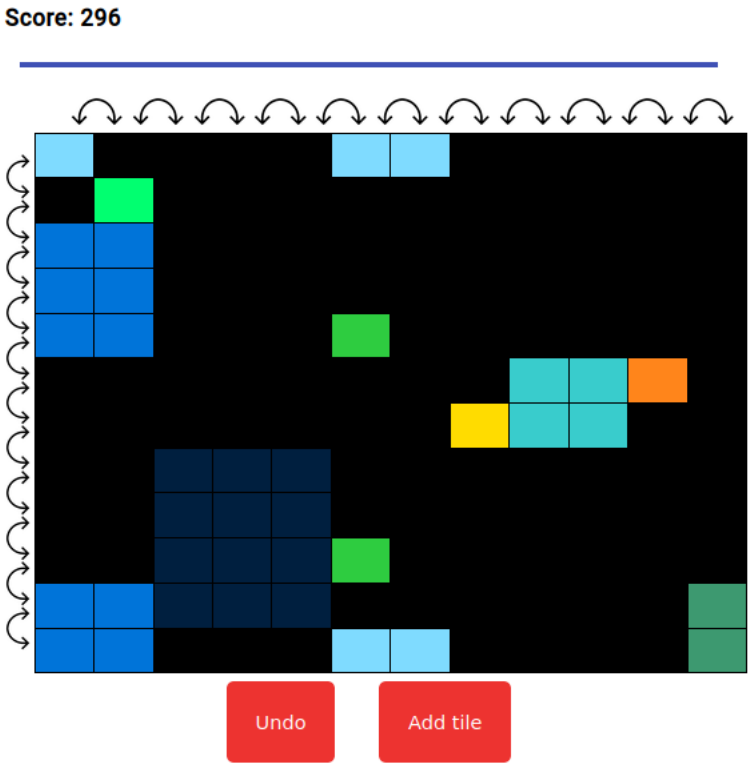


Figure 7: The tiling produced by always taking the first suggestion.

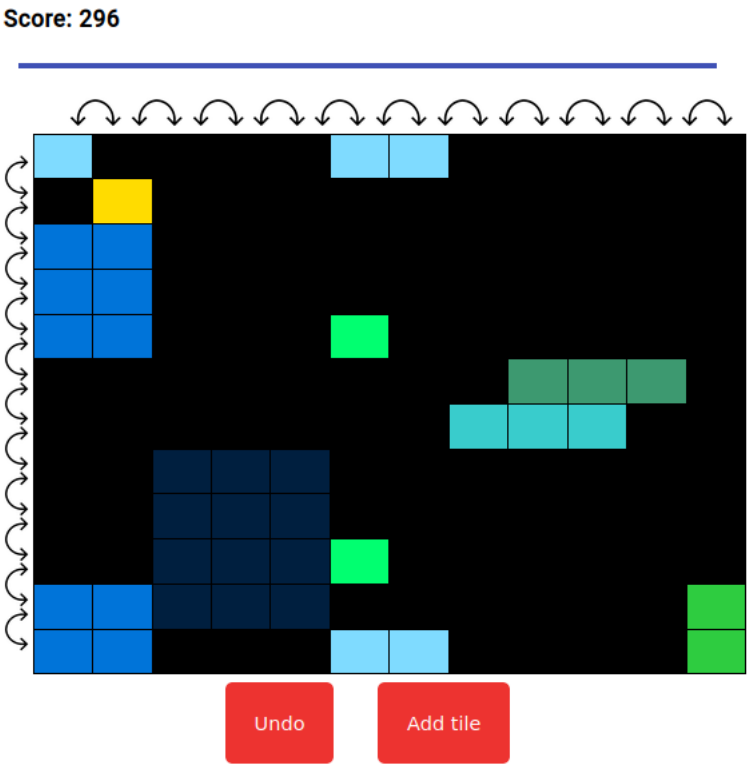


Figure 8: One possible tiling using only eight tiles.

7.2 Results

In total, 93 people participated in the experiment. Of these 93 participants, four (4.3%) managed to tile the data set in 8 tiles. The average tile count for all participants was 11.02, which is more than two tiles worse than always picking the first LTM suggestion. Almost all (93.6%) participants used between 8 and 13 tiles, however there are some outliers above that, with the highest being 23, followed by 19. A visualisation of the distribution of the tile counts can be seen in Figure 9. Of all tilings, only 25.8% were at least as good as always taking the first suggestion. These as 'at least as good' tilings took 405.16 seconds (6.75 minutes) on average.

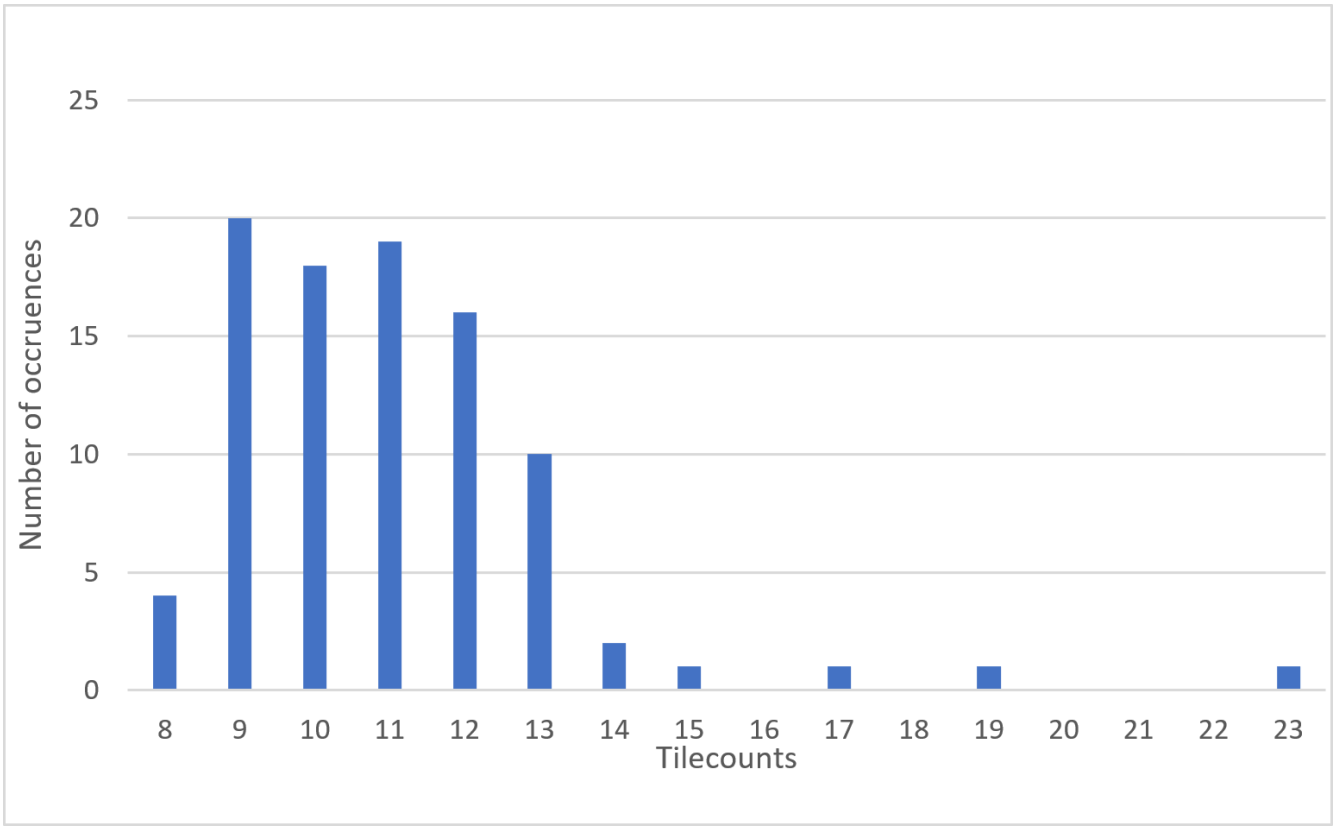


Figure 9: Tile count distribution.

An interesting aspect of the tilings can be seen when we look at the average time taken for every occurring tile count. The tilings consisting of the most tiles (namely, using 19 and 23 tiles) were also, on average, the ones that took the longest to construct. The average time taken across all tile counts was 7 minutes and 8 seconds. The fastest participant was done in only 18 seconds, whereas the slowest took 45 minutes and 53 seconds. Interestingly, both of these extremes have a tile count of 11. It is, however, very probable that the slowest participant did not spend all their time actually tiling. Between two of their tiles was a gap of 30 minutes, suggesting that the participant was not actively engaged with application for the entire duration of the experiment. This cannot be stated for certain however, as no information was recorded for the tiles that have been undone, making it possible that this participant spent these 30 minutes trying various ways of tiling the data set.

A visualisation of the time taken for all tilings can be seen in Figure 10. First, the number of milliseconds was converted to minutes by dividing by $1000 * 60$, rounded to the nearest integer. This graph looks similar to the tile count graph, but that is merely a coincidence: the correlation between time spent and the tile count is 0.015, with a t-score of 0.14 and a p-value of 0.88. There does not seem to be a correlation between the amount of time participants spent on constructing the tiling and the number of tiles used.

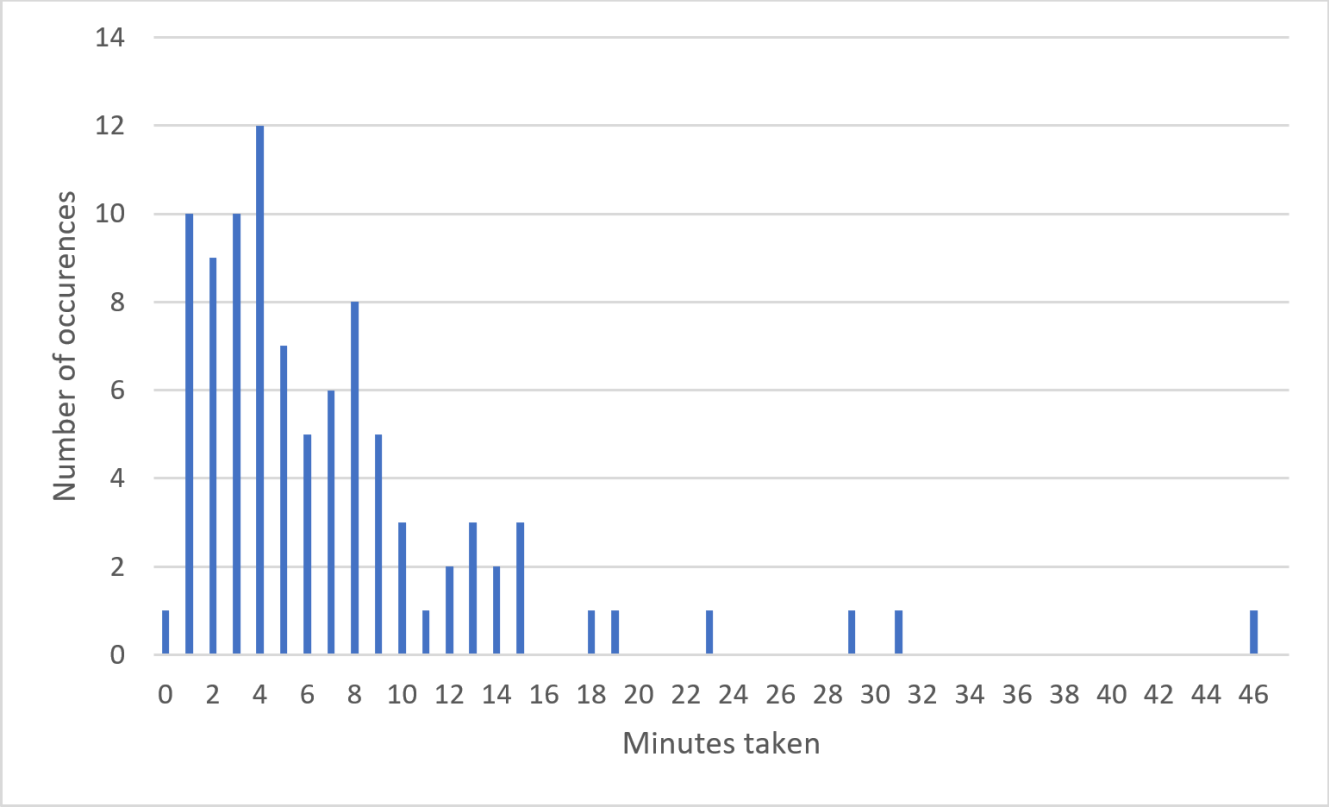


Figure 10: Time taken distribution.

This can also be seen in Figure 11, which shows a scatter plot of the tile counts and the time taken (in minutes) of the participants. The dotted line is a trend line, which shows the correlation between the data. We can see that there is virtually no correlation between tile count and the time taken. This graph also shows that most participants use less than 15 minutes and less than 15 tiles.

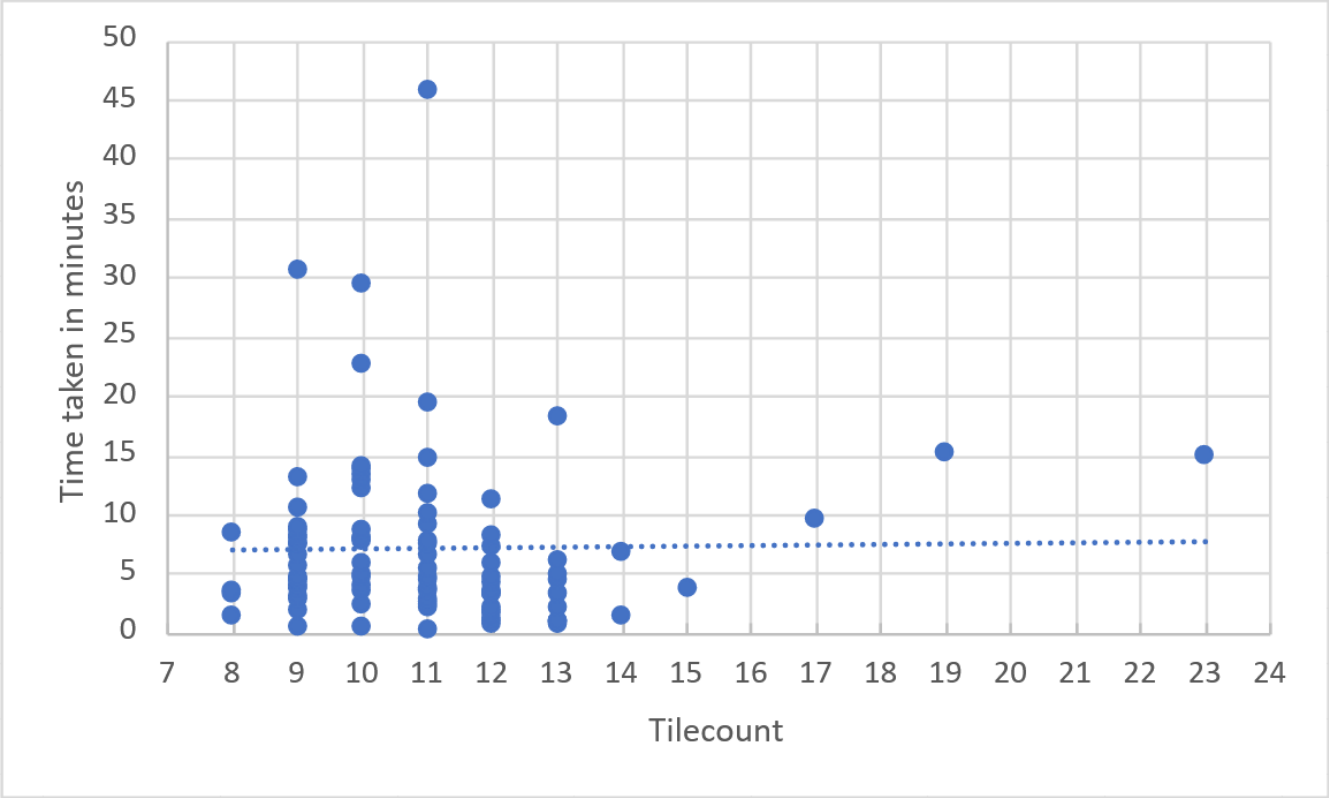


Figure 11: Scatter plot of tile counts and time taken in minutes.

More data can be found in Tables 1, 2 and 3. Table 1 shows the differences between the various academic levels of the participants. It looks like the average tile count and the average time taken slightly decreases with higher education (but the upper two groups are quite small).

Academic level	Occurrences	Average time taken in minutes	Average tile count
High school	4	16.55	13.50
Vocational education	5	5.20	11.40
Bachelor's	25	7.57	10.96
Master's	39	7.73	11.03
Doctorate	20	4.05	10.50

Table 1: Occurrences, average time taken and the average tile count for each academic level.

Table 2 shows the differences in tiling statistics between the age groups. Here, it also seems like the older participants are, the better their tiling is on average. There were no participants of 60 years or older, so there is no data for this age group.

Age range	Occurrences	Average time taken in minutes	Average tile count
12-25	45	7.98	11.51
26-39	39	6.70	10.72
40-59	9	4.84	9.89
60+	0	-	-

Table 2: Occurrences, average time taken and the average tile count for different age groups.

Table 3 shows the difference in tiling performance between participants with a Computer Science background, compared to participants without such a background. Participants with a CS background are actually significantly better (two-sample t-test, assuming unequal variances, $p = 0.029$) than the participants without such a background. Their average is only (about) one tile better, and moreover, still one tile worse than taking the first suggestion every time. A single tile is relatively important however, as the data set is quite small, and the number of tiles is also small.

CS background	Occurrences	Average time taken in minutes	Average tile count
No	60	7.58	11.37
Yes	33	6.32	10.39

Table 3: Occurrences, average time taken and the average tile count for people with and without a CS background.

7.3 Discussion

It turns out that some of the participants were able to perform better at tiling than greedily using the 'best' suggestion by LTM. There are, however, some caveats. Only 4.3% of the participants managed to succeed in this goal. Also, it is improbable that this approach scales well, as human attention span becomes a limiting factor. Already for this small data set, the average time spent on tiling was more than 7 minutes. In combination with the fact that the average tiling was worse (about 11 tiles) than the greedy suggestions by LTM (9 tiles), it does not seem likely that for larger data sets, humans would perform well enough to justify the amount of time spent.
Some more data points would have been helpful in understanding and interpreting the results. It could be insightful to record for every tile in a tiling if it was a suggestion by LTM or not. That way, we might have been able to see how the suggestions were used, which is not generally possible now. Looking at a specific tiling, we might be able to reconstruct such information, but an in-depth look is not feasible for 93 participants. If we had such data, we could analyze if the number of suggestions the participant used correlates (positively, or negatively) with the total tile count.
Another limitation of this research is that we have only had the participants tile a single data set. Including more data sets might have enabled us in finding situations where the human-computer hybrid performs better or worse.

8 Conclusions and Further Research

8.1 Conclusion

In this thesis we have tried to determine if a human-computer hybrid can outperform the LTM tiling algorithm in completely tiling a data set. To research this question, a web application was built that allows users to construct a tiling. Furthermore, LTM was implemented and used to suggest tiles to the participant. It turns out that, yes, a human-computer hybrid approach can outperform LTM at tiling a binary data set. In our research, $4,3\%$ of the participants managed to construct a tiling for our test data set consisting of only 8 tiles, while iteratively using LTM's best suggestion results in 9 tiles. However, the average number of tiles used by the participants (11.02) is worse than only using LTM. Moreover, the average amount of time needed to complete these tilings surpassed 7 minutes, making it very unlikely that this approach will perform well in real world applications.

8.2 Further Research

The results of our research open up some new questions. A natural follow-up research would be finding out whether this approach to tiling scales to larger data sets. We have only considered a fairly small (12 by 12) database, but any real world application would be very likely to involve vastly larger databases. As we essentially try to solve an $\mathcal{NP}$-hard problem, it would also be interesting to see if this method of human-computer cooperation could work well on other $\mathcal{NP}$-hard problems. Another possibility for further research concerns the suggestions made to the user. Right now, the suggested tiles of LTM are ranked by their added value $val(T)$. Finding a different heuristic that performs better when combined with a human data miner may be possible. Finally, it might be possible to increase the performance of the human participant by adding features to the user interface. A possible addition could be a way to automatically sort the data set, as it has been shown that spectral orderings might help humans to find good (combinatorial) tiles that are not immediately obvious [GMS04].

9 Acknowledgements

I would like to thank my thesis supervisor Matthijs van Leeuwen, for his patience and advice. Also much gratitude to Lincen Yang for being the second reader of this thesis. Additionally, I would like to thank everybody that participated in the trial for my experiment, for finding errors that I did not see, and all 93 participants of the experiment for their time. Finally, I want to thank Jasper van Riet and Sander Ronde for their help with Angular and JavaScript in general, and Tess Remmerswaal for proofreading.

References

[Dar21] DarkRiDDeR. *js-table-cell-selector*. GitHub, 2021. Publication Title: GitHub repository.

[DB11] Tijl De Bie. Maximum entropy models and subjective interestingness: an application to tiles in binary databases. *Data Mining and Knowledge Discovery*, 23(3):407–446, November 2011.

[GGM04] Floris Geerts, Bart Goethals, and Taneli Mielikäinen. Tiling Databases. In Einoshin Suzuki and Setsuo Arikawa, editors, *Discovery Science*, Lecture Notes in Computer Science, pages 278–289, Berlin, Heidelberg, 2004. Springer.

[GMS04] Aristides Gionis, Heikki Mannila, and Jouni K. Seppänen. Geometric and Combinatorial Tiles in 0–1 Data. In Jean-François Boulicaut, Floriana Esposito, Fosca Giannotti, and Dino Pedreschi, editors, *Knowledge Discovery in Databases: PKDD 2004*, Lecture Notes in Computer Science, pages 173–184, Berlin, Heidelberg, 2004. Springer.

[Hol13] Andreas Holzinger. Human-Computer Interaction and Knowledge Discovery (HCI-KDD): What Is the Benefit of Bringing Those Two Fields to Work Together? In Alfredo Cuzzocrea, Christian Kittl, Dimitris E. Simos, Edgar Weippl, and Lida Xu, editors, *Availability, Reliability, and Security in Information Systems and HCI*, Lecture Notes in Computer Science, pages 319–328, Berlin, Heidelberg, 2013. Springer.

[noa17] colors.css palette | ColorsWall.com, December 2017.

[QB11] Alexander J. Quinn and Benjamin B. Bederson. Human computation: a survey and taxonomy of a growing field. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 1403–1412, New York, NY, USA, May 2011. Association for Computing Machinery.

[TV12] Nikolaj Tatti and Jilles Vreeken. Discovering Descriptive Tile Trees. In Peter A. Flach, Tijl De Bie, and Nello Cristianini, editors, *Machine Learning and Knowledge Discovery in Databases*, Lecture Notes in Computer Science, pages 9–24, Berlin, Heidelberg, 2012. Springer.

Appendices

A Web Application

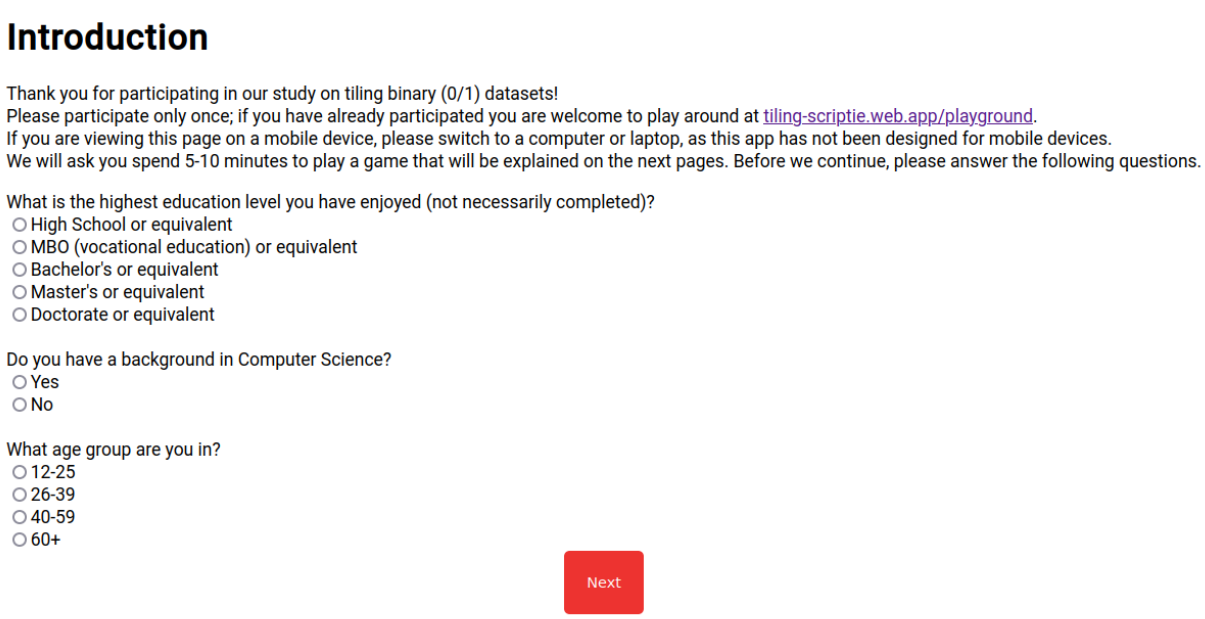
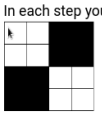


Figure 12: Landing page with short survey.

Tiles

Initially, you will see a table with white and black cells (representing a matrix with 1s and 0s, respectively). Your goal is to colorize all white cells.
At the start it will look like this:

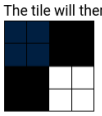
In each step you can select a rectangle of white cells. This is done by clicking on a cell and dragging the mouse across the table.



We call such a rectangle, containing only white cells, a *tile*. You can add a tile by clicking the following 'Add tile' button.

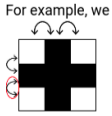
Add tile

The tile will then be colorized:

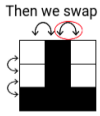


To finish tiling this particular example, we need to add an additional tile covering the remaining white cells:

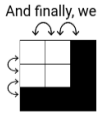
And we're done!
The order of the rows and columns does not matter, so we can shuffle them around to find tiles! Use the arrow icons near the table to swap rows and columns
For example, we first swap these rows:

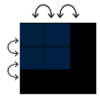


Then we swap columns 2 and 3:

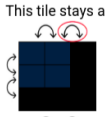


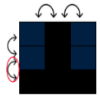
And finally, we have a nice (contigious, completely white) tile to add:

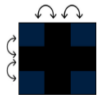




This tile stays a tile, even when we swap the rows and columns again.







For every tile you add, you gain points. Tiles that cover more white cells are worth more points, so try to find the largest tiles! The game ends when all white cells have been covered by a tile.

Next

Figure 13: First part of the tutorial, explaining how to add tiles.

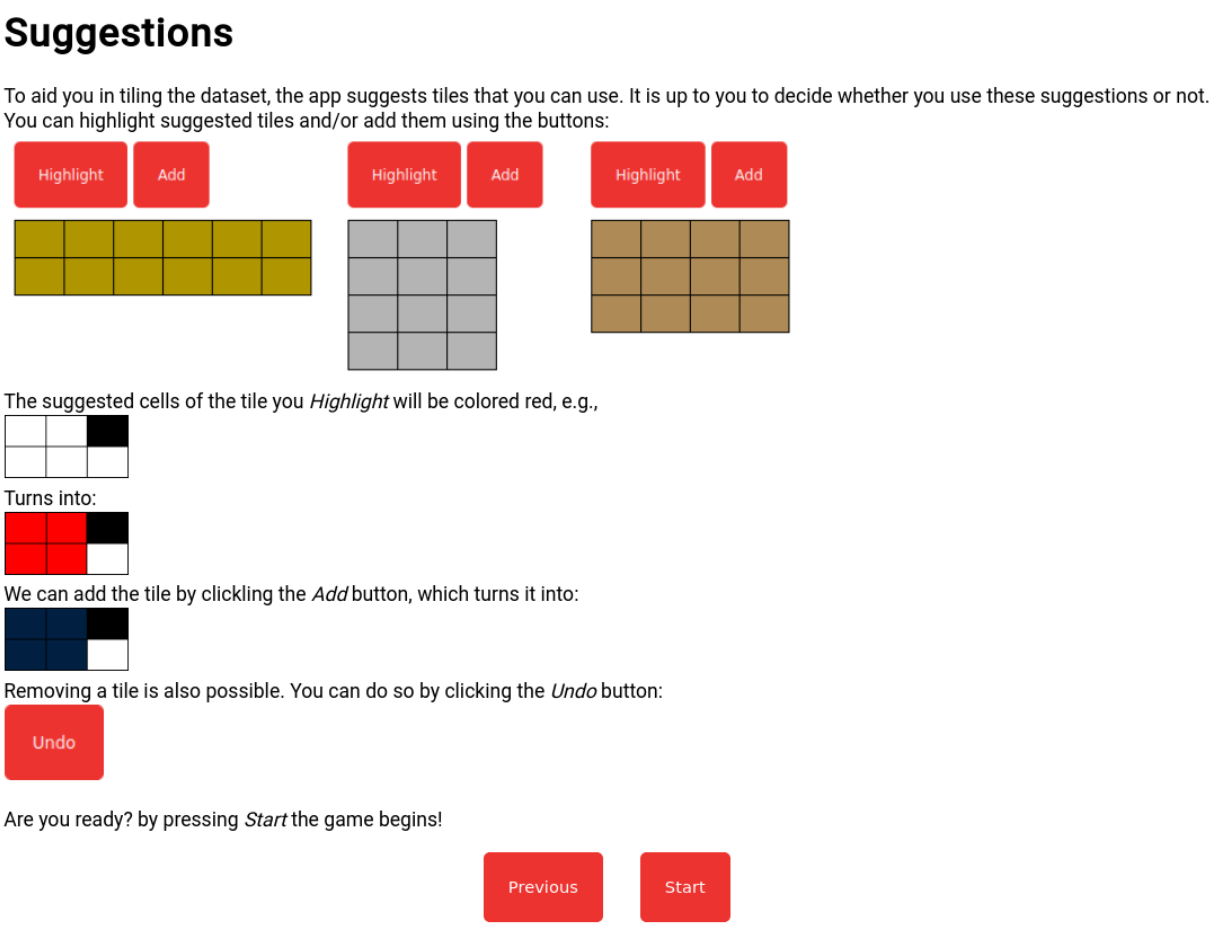


Figure 14: Final part of the tutorial, explaining the suggestions and the undo button.