AUSTRALASIAN JOURNAL OF COMBINATORICS
Volume **29** (2004), Pages 273–300

Complexity of a $\{0,1\}$-matrix problem

Henning Fernau

University of Newcastle
School of Electrical Engineering and Computer Science
University Drive, NSW 2308 Callaghan
Australia
fernau@cs.newcastle.edu.au
Universität Tübingen
Wilhelm-Schickard-Institut für Informatik
Sand 13, D-72076 Tübingen
Germany
fernau@informatik.uni-tuebingen.de

Abstract

We consider the following problem: given an $n \times m$ $\{0,1\}$-matrix M and an integer k, is it possible to get the all-zeros-matrix by merging at most k neighbouring rows or columns? Here, merging means to perform a component-wise AND operation. We prove that this problem is NP-hard but fixed-parameter tractable (taking k as parameter we show that there is an $O(2.6181^k * n * m)$ algorithm) and factor-3-approximable (considered as minimization problem). Moreover, we discuss the relation of this problem to 4-Hitting-Set and Vertex Cover.

1 Introduction

L. Branković communicated to us the following abstract problem arising in the theory of statistical databases, stated below in an abstract and simplified version:

Matrix Row/Column Merging (MRCM)
Given: $n \times m$ $\{0,1\}$-matrix M
Parameter: positive integer k
Question: Is it possible to get the all-zeros-matrix by merging at most k neighbouring rows or columns?
Here, *merging* means to perform a component-wise logical AND.

For short, we will also call such a matrix *k-mergible*. In order to simplify our formulations, a *line* of a matrix is either a row or a column of that matrix. Hence, a matrix is k-mergible if and only if the all-zeros-matrix can be obtained by merging at most k neighbouring lines. A line can be of *type* row or column. We also say that

rows and columns are *opposite types*. Instead of speaking of "two lines of the same type" we will also speak about two *parallel* lines, and "two lines of opposite type" are also referred to as *orthogonal* lines.

An instance of MRCM can be specified as a pair (M, k), where M is a $\{0,1\}$-matrix and k is a positive integer.

In this paper, we will show that MRCM is NP-complete but both constant-factor approximable (when seen as a minimization problem) and tractable in the sense of parameterized complexity. This implies that the corresponding (more complex) database problems are also computationally hard but possibly amenable to approaches as provided by approximation algorithms or parameterized complexity.

The paper is organized as follows: In Sec. 2, we try to familiarize ourselves with the problem, indicating at heuristics that may help solve it. Sec. 3 contains the proof that MRCM is NP-complete. In Sec. 4, we explain a simple factor-4 approximation algorithm based on a local ratio approach. Sec. 5 contains a first search-tree algorithm for MRCM, which is going to be refined in the following sections. In Sec. 7, further deeper relations to 4-Hitting-Set and to vertex cover are explained, which is going to be finally exploited in Sec. 8 to arrive at an $O(2.6181^k mn)$-algorithm for MRCM. This final algorithm is quite simple, but its analysis requires a novel form of book-keeping analysis to cater for decisions which are actually deferred to a later, polynomial-time computation phase. This methodology could be in fact wider applicable than just to the $\{0,1\}$-matrix problem under investigation in this paper.

2 Examples and first considerations

Let us look at some examples to become familiar with MRCM and its peculiarities. Here and in the following examples we will only give the ones in the matrix; the other entries are zero by default.

Example 1: Let us consider the following matrix instance M:

	1	2	3	4	5	6	7	8	9
1	1			1					1
2									
3	1								
4			1	1			1		
5									
6									
7				1					1

Can we eliminate the ones by merging at most $k \leq 3$ neighbouring rows or columns? For example, merging rows 1 and 2, 4 and 3, as well as 6 and 7 leaves us with:

	1	2	3	4	5	6	7	8	9
1/2									
3/4									
5									
6/7									

Observe that the merged rows are written as a list of row numbers. This notation is convenient, since it allows us to talk about, e.g., merging rows 4 and 5 later on (of course, this is not necessary in our example), referring to the original row numbers. Alternatively, we would have to re-number the lines after each merge operation, but this appears to be too awkward to explain examples, although it is quite natural when thinking about implementing solving algorithms. That is why we will later deviate from this convention. A second point we gain by our convention is that the order in which we apply merge operations does not matter at all. In our example, we could describe our solution as $\{\mathrm{RM}(1,2), \mathrm{RM}(3,4), \mathrm{RM}(6,7)\}$. Similarly, we can indicate column mergers by writing $\mathrm{CM}(i,j)$. To simplify our notation, we will also say that the operation $\mathrm{RM}(\cdot,\cdot)$ is of the *type row*, and the operation $\mathrm{CM}(\cdot,\cdot)$ is of the *type column*.

In other words, the given instance is 3-mergible. Is it possible to fix the instance by using only two merging operations? Let us look at the first row. It contains three ones. If all the ones were "fixed" by column mergers, this would need at least three merging operations. Hence, we must fix these ones by merging row 1 with row 2. A similar argument applies to row 4. But now we have used two (enforced) mergers but did not fix the matrix successfully. Hence, the instance is not 2-mergible.

In passing, we have developed a simple rule for MRCM:

Rule 1: If M is k-mergible and contains a line with more than k ones, then this line must be fixed by using merging operations of the type of the line.

This rule is sound: observe that by merging say two columns, at most one 1 per row is eliminated. Hence by induction, performing k column merging operations will fix no more than k ones per row. Therefore, if we have a row ℓ with $(k+1)$ ones (or more), but are only allowed to perform at most k merging operations, then we must select at least one of the operations $\mathrm{RM}(\ell-1,\ell)$ or $\mathrm{RM}(\ell,\ell+1)$ to fix row ℓ.

Here, *fixing a line* does not imply that after the operations necessarily no ones remain, since merging ones with ones let some ones survive. This (simple) observation is crucial when later comparing MRCM with set cover problems like HITTING SET and VERTEX COVER.

Observe that Rule 1 may be "weakened" to derive the following, more greedy heuristic rule:

Rule 1′: Fix some line with the most number of ones first. This line must be fixed by using merging operations of the type of the line.

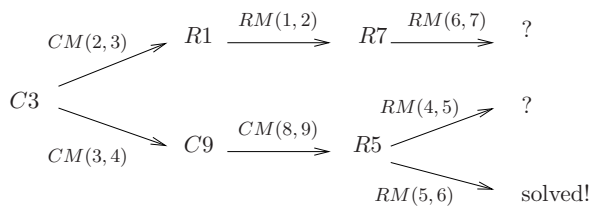
Unlike usual greedy-like rules, this does not give us a polynomial-time algorithm, since in general there will be two possibilities to fix a certain line by operations of the type of the line, the only exception being trials to fix some "border line", a case which hence should be preferred when breaking ties.

Example 2: Let us look at a different example:

	1	2	3	4	5	6	7	8	9
1				1					1
2				1				1	
3			1						
4			1					1	
5	1	1							
6					1	1			
7			1						1

If we ask whether this matrix is 3-mergible, then Rule 1 does not help us reduce the problem.

In fact, Rule 1′ does solve this question. More precisely, the following rather shallow search tree will result:



In this picture, the vertices of the tree are labelled with the lines selected to be fixed, and the edges are labelled with the possible fixing operations. Observe that in this case only three different cases (instead of the eight cases in the worst case) occur due to preferring "border line" fixes. In fact, one of the paths in this search tree gives us the desired answer that the given matrix is indeed 3-mergible.

But, there is a similar (but more complicated) rule for helping us here:

Rule 2: If M is k-mergible and contains two neighbouring lines with more than k ones at "different" "locations," then one of these lines must be fixed by using merging operations of the type of the line.

Here, the "location" of a one in a specific row is its column index and the location of a one in a specific column is its row index. Two locations l, l' in two neighbouring lines are "different" if they differ by more than one, i.e., $|l - l'| > 1$. Two locations l, l' in the same line are "different" if they differ by more than one, i.e., $|l - l'| > 0$.

In our example, the rule applies to:

- rows 5 and 6; all four locations (1 and 2 in row 5; 5 and 6 in row 6) are different;

- rows 6 and 7;

- columns 3 and 4; e.g., take locations 3, 4 and 7 in the third column, as well as location 1 in the fourth column.

The rule does *not* apply to:

- rows 4 and 5, since the location 2 in row 5 and the location 3 in row 4 are not different; seen as an isolated problem, rows 4 and 5 can be resolved by using three column merging operations: CM(1,2), CM(2,3), and CM(7,8);

- columns 8 and 9, since the location 2 in column 1 and the location 1 in column 9 are not different; in fact, considering columns 8 and 9 isolatedly, we see that they can be fixed by: RM(1, 2), RM(4,5), and RM(6,7).

Although the second rule does not immediately tell us what to do, it suggests applying the following greedy heuristics:

Algorithm MRCM-Greedy:

```
As long as there are ones in the matrix, do:
   for each merge operation, compute the gain of the operation;
   perform an operation of maximal gain
```

Here, the "gain" of an operation should be the number of ones erased by performing the operation.

In the last example, the maximal gain, namely five, is obtained by performing first CM(3, 4), followed by RM(5, 6) and CM(8, 9). This shows that the given example is 3-mergible.

In fact, it is easily argued that the solution is optimal: reconsider columns 8 and 9. By Rule 2, we know that some of the ones must go by applying a column merging operation. Since column 9 is at the right-most border, the only possible column mergings are CM(7, 8) and CM(8, 9). Since CM(8, 9) erases all the ones that CM(7, 8) does (and some more), there is no reason to pick CM(7, 8). Hence, CM(8, 9) should be performed. Now, Rule 1 applies to the third column. Hence, there is no use in erasing some of the ones in the third column by row mergers. Since Rule 2 tells us to use at least one of {RM(4, 5), RM(5, 6), RM(6, 7)}, and since both RM(4, 5) and RM(6, 7) will only erase, in addition to ones in row 5 or 6, some ones in column 3, we have to choose RM(5, 6). Hence, finally we have to take CM(3, 4) to fix the remaining ones.

Unfortunately, the solution of Algorithm MRCM-Greedy is not always optimal, as the following example shows:

Example 3: Greedy is not optimal:

	1	2	3	4	5	6	7	8	9
0						1		1	
1	1				1		1		
2				1	1				
3		1		1					
4				1					
5						1			
6	1								
7			1					1	
8	1		1		1		1		
9								1	

The heuristic might wish to pick $\mathrm{RM}(0,1)$ and $\mathrm{RM}(8,9)$, although the remaining matrix still needs 4 mergings to get to the all-zero-matrix. In fact,

$$\{\mathrm{CM}(1,2), \mathrm{CM}(3,4), \mathrm{CM}(5,6), \mathrm{CM}(7,8)\}$$

would be an optimal solution from the very beginning.

Remark: In contrast to the—as we will later explain in more detail—related VERTEX COVER problem, where the "corresponding" maximum-degree heuristic can perform arbitrarily bad, we are not aware of a corresponding result in the case of the maximum-gain heuristic for MRCM. Conversely, we were not able to show a constant-factor upper bound when algorithm MRCM-Greedy is considered as approximation algorithm.

So, the problem MRCM seems to be non-trivial. We will learn more about the hardness of MRCM in the next section.

Let us conclude this section by considering a two seemingly pathological cases, arising from the following basic question: what happens if one line is *completely* filled with ones?

Observation 1: A line ℓ completely filled with ones must be fixed by at least one merging operation of the same type as the line, more precisely, performing a merger of lines $\ell - 1$ and ℓ or of ℓ and $\ell + 1$.

Corollary: A matrix completely filled with ones cannot be fixed.

Our observation leads to the following reduction rule:

Rule 3: Consider an instance (M, k) of MRCM. If M contains a line ℓ completely filled with ones, then ℓ must be fixed by either $XM(\ell - 1, \ell)$ or $XM(\ell, \ell + 1)$, where XM is a merging operations of the same type as the line ℓ. Since the matrix M' obtained after applying $XM(\ell - 1, \ell)$ is the same as the matrix obtained after applying $XM(\ell, \ell+1)$, we can safely reduce (M, k) to $(M', k-1)$. Here, M' denotes the matrix obtained by applying $XM(\ell - 1, \ell)$.

Finally, we provide a reduction rule dealing with lines completely filled with zero entries.

Rule 4: If M contains two neighbouring lines ℓ and $\ell+1$ which are completely filled with zeros, then (M,k) can be safely reduced to (M',k), where M' is obtained from M by deleting line ℓ.

Note that a matrix completely filled with ones will reduce to the 1×1 matrix containing one 1 by Rule 3. Conversely, a solvable instance (M,k) of MRCM will finally reduce to the 1×1 matrix containing one 0 by Rule 4 after having arrived at an all-0-matrix by some other methods. More generally, if k is large enough, a matrix M is solvable if and only if it contains a zero entry:

Observation 2: Let $M\in\{0,1\}^{n\times m}$. Then, $(M,m+n-2)$ is a solvable instance of MRCM if and only if M contains a zero entry.

We conclude this section with a simple algebraic observation. Since the logical AND operation is commutative and associative, the componentwise AND operation is commutative and associative, too. This especially implies:

Observation 3: Whenever convenient, we may re-order merging operations. In particular, we may assume that all operations of one type (row mergers or column mergers) are performed at the beginning or at the end.

3 MRCM is NP-complete

We will provide a reduction from 3-SAT which is similar to the textbook reduction of 3-SAT to VERTEX COVER, see [6].

Let us consider an instance of 3-SAT. Let

$$E=\{C_1,\dots,C_m\}$$

be a collection of clauses

$$C_i=\ell(i,1)\vee\ell(i,2)\vee\ell(i,3)$$

(the $\ell(i,j)$ being literals, i.e., variables or negated variables) with variables $x_1,\dots,x_n$. Technically speaking, an instance of 3-SAT is a Boolean expression E in CNF (conjunctive normal form) where each clause has exactly three literals, i.e, E is in 3-CNF. Recall that 3-SAT asks if E is *satisfiable*, i.e., whether or not there exists an assignment of truth values to the variables such that

$$C_1\wedge\cdots\wedge C_m$$

evaluates to true.

We will show how to translate such an instance E into an instance $(r(E),p(E))$ of MRCM such that E is satisfiable if and only if the matrix $r(E)$ is $p(E)$-mergible. We will specify $r(E)$ and the parameter $p(E)$ in what follows.

$$p(E)=n+3m.$$

The matrix $r(E)$ has size $(7m+5n)(9m+3n)$. For convenience and mnemotechnical reasons, we call the rows

$$\begin{aligned} &r_c(i,j), \quad 1 \le i \le m, 1 \le j \le 7 \quad \text{and} \\ &r_x(i,j), \quad 1 \le i \le n, 1 \le j \le 5. \end{aligned}$$

The index c should refer to the fact that these rows basically simulate clauses, and the index x should indicate that the corresponding rows mainly simulate variables.

Similarly, we will use the following columns:

$$\begin{aligned} &c_c(i,j), \quad 1 \le i \le m, 1 \le j \le 9 \quad \text{and} \\ &c_x(i,j), \quad 1 \le i \le n, 1 \le j \le 3. \end{aligned}$$

There are three different cases causing ones to appear in the matrix; each matrix index not mentioned this way has zero as entry.

1. A clause $C_i = \ell(i,1) \vee \ell(i,2) \vee \ell(i,3)$, $1 \le i \le m$, causes the following ones to appear in the matrix:
   (a) $(r_c(i,4), c_c(i,2))$
   (b) $(r_c(i,2), c_c(i,3))$
   (c) $(r_c(i,4), c_c(i,4))$
   (d) $(r_c(i,5), c_c(i,6))$
   (e) $(r_c(i,6), c_c(i,8))$
2. Each variable x_k, $1 \le k \le n$, we introduce a one at
   $$(r_x(k,3), c_x(k,2))$$
   in the matrix.
3. Then, there are "interconnections" between the lines dedicated to clauses and the lines dedicated to variables. This way, the actual Boolean expression (in 3-CNF) is codified.
   - If $\ell(i,j) = x_k$, then there is a one at $(r_x(k,2), c_c(i,2^j))$.
   - If $\ell(i,j) = \overline{x_k}$, then there is a one at $(r_x(k,4), c_c(i,2^j))$.

Let us consider a typical "clause gadget" in more details. In Table 1, we consider

$$c = x_1 \vee \overline{x_2} \vee x_3$$

and give both the clause gadget and the interconnection gadget. The reader may convince her/himself that the following discussion also applies to a more general situation. For clarity, only one-entries are given in Table 1.

We will use this matrix to show our first claim:

	$c_c(1)$	$c_c(2)$	$c_c(3)$	$c_c(4)$	$c_c(5)$	$c_c(6)$	$c_c(7)$	$c_c(8)$	$c_c(9)$
$r_c(1)$									
$r_c(2)$			1						
$r_c(3)$									
$r_c(4)$		1		1					
$r_c(5)$						1			
$r_c(6)$								1	
$r_c(7)$									
$r_x(1,1)$									
$r_x(1,2)$		1							
$r_x(1,3)$									
$r_x(1,4)$									
$r_x(1,5)$									
$r_x(2,1)$									
$r_x(2,2)$									
$r_x(2,3)$									
$r_x(2,4)$				1					
$r_x(2,5)$									
$r_x(3,1)$									
$r_x(3,2)$								1	
$r_x(3,3)$									
$r_x(3,4)$									
$r_x(3,5)$									

Table 1: An example for the clause and interconnection gadgets

Claim 1: The above matrix becomes 3-mergible if one of the ones in the "variable rows" is fixed by merging rows. Moreover, the other mergings will only affect lines indexed c.

Let us consider cases to prove this claim:

1. Assume that $(r_x(1,2), c_c(2))$ is fixed by merging rows. Without loss of generality, this would mean that we merge rows $r_x(1,2)$ and $r_x(1,3)$, since $r_x(1,1)$ contains only zeros and $r_x(1,3)$ contains a one in the "variable gadget" (which is not shown).—We will encounter this "w.l.o.g.-argument" repeatedly below.

   Then, merging $c_c(3)$ with $c_c(4)$, $c_c(7)$ with $c_c(8)$ and $r_c(4)$ with $r_c(5)$ will fix the whole matrix.

2. Assume that $(r_x(2,4), c_c(4))$ is fixed by merging rows. Without loss of generality, this would mean that we merge rows $r_x(2,3)$ and $r_x(2,4)$. Then, merging $c_c(2)$ with $c_c(3)$, $c_c(7)$ with $c_c(8)$ and $r_c(4)$ with $r_c(5)$ will fix the whole matrix.

3. Assume that $(r_x(3,2), c_c(8))$ is fixed by merging rows. Without loss of generality, this would mean that we merge rows $r_x(3,2)$ and $r_x(3,3)$. Then, merging

$c_c(2)$ with $c_c(3)$, $c_c(3)$ with $c_c(4)$ and $r_c(5)$ with $r_c(6)$ will fix the whole matrix.

This first claim is important, since fixing one of the ones in the interconnection gadget by row merging corresponds to satisfying the clause by setting the corresponding variable appropriately. What happens if, according to this interpretation, a certain clause is not satisfied? This means that the ones set according to the interconnection gadgets must be fixed (w.l.o.g.) by column merges.

Claim 2: If the interconnection gadgets are fixed by column merges, then the matrix given in Table 1 is 4-mergible but not 3-mergible.

Namely, w.l.o.g., we can assume that $c_c(2)$ merges with $c_c(3)$ and $c_c(3)$ merges with $c_c(4)$, as well as $c_c(8)$ merges with $c_c(7)$ (or $c_c(9)$). Then, there is one remaining one at $(r_c(5), c_c(6))$ which must and can be fixed arbitrarily. This shows the second claim.

Finally, we have to argue that also in the case when a clause is "multiply satisfied" we still need three mergings to fix the clause gadget itself.

	$c_c(1)$	$c_c(2)$	$c_c(3)$	$c_c(4)$	$c_c(5)$	$c_c(6)$	$c_c(7)$	$c_c(8)$	$c_c(9)$
$r_c(1)$									
$r_c(2)$			1						
$r_c(3)$									
$r_c(4)$		1		1					
$r_c(5)$						1			
$r_c(6)$								1	
$r_c(7)$									

Table 2: The clause gadget itself is 3-mergible at the best

Claim 3: The matrix given in Table 2 is 3-mergible but not 2-mergible:

To see this, consider the one at $(r_c(6), c_c(8))$. The best way to fix this is to merge $r_c(5)$ and $r_c(6)$, since otherwise, no other one would disappear. The remaining three ones can be repaired by two further merges, but cannot disappear using only one merging operation.

Using these three claims, we are now ready to prove the main result of this section:

Theorem 1: MRCM is NP-complete.

Firstly, it is rather easy to see that MRCM can be nondeterministically solved in polynomial time: given an instance (M, k), one guesses at most k merging operations and then verifies (deterministically in polynomial time) that this way all ones disappear.

Secondly, let us argue about hardness. More specifically, we show that the function r mapping 3-SAT instances onto MRCM instances is a reduction.

On the one hand, assume that E is satisfiable. We have to show that $r(E)$ is $p(E)$-mergible. Let $X = \{x_1, \ldots, x_n\}$ be the variables appearing in $E = \{C_1, \ldots, C_m\}$. Let $\alpha : X \to \{0, 1\}$ be a satisfying assignment of E. Then, the following mergings show that $r(E)$ is indeed $p(E)$-mergible.

1. If $\alpha(x_k) = 1$, then merge $r_x(k, 2)$ and $r_x(k, 3)$. This way, both the 1 located at $(r_x(k, 3), c_x(k, 2))$ and the ones located at $(r_x(k, 2), c_c(i, 2^j))$ (referring to $\ell(i, j) = x_k$ in clause C_i) will disappear.

2. If $\alpha(x_k) = 0$, then merge $r_x(k, 3)$ and $r_x(k, 4)$. Hence, both the 1 located at $(r_x(k, 3), c_x(k, 2))$ and the ones located at $(r_x(k, 4), c_c(i, 2^j))$ (referring to $\ell(i, j) = \overline{x_k}$ in clause C_i) will disappear.

3. Since α is a satisfying assignment, the first two steps guarantee that, for each clause, it is true that the conditions of Claim 1 are satisfied. Hence, for each clause, we can do with three additional mergings.

Now, it is easy to see that $r(E)$ is indeed $p(E) = n + 3m$-mergible.

On the other hand, assume that $r(E)$ is k-mergible.

1. According to Claim 3, for each "clause gadget", we need at least three merging operations to make all ones disappear. Since all clause gadgets are independent of each other, this means that at least $3m$ mergings are necessary to fix all clause gadgets.

2. The n ones appearing in "variable gadgets" are also independent of each other (and of the clause gadgets), which means that at least n further mergings are necessary.

Our reasoning so far shows that $k \geq p(E)$. When does equality hold? Assume now $k = p(E)$. This means that all ones in the "interconnection gadgets" must be fixed in passing when erasing the ones in the clause gadgets and in the variable gadgets. Especially, this implies that each one appearing in a variable gadget is erased by one row merging operation.

We are now going to construct a variable assignment α. Consider a variable x_k and interpret the case when the one located at $(r_x(k, 3), c_x(k, 2))$ disappears due to the merger of rows $r_x(k, 2)$ and $r_x(k_3)$ by setting $\alpha(x_k) = 1$. Similarly, interpret the merger of rows $r_x(k, 3)$ and $r_x(k_4)$ by setting $\alpha(x_k) = 0$.

According to Claim 2 and the above considerations, assuming that $k = p(E)$ implies that for no "interconnection gadget" pertaining to a specific clause C_i, all ones are fixed by column mergings (otherwise, $k > p(E)$). Hence, each interconnection gadget is fixed by using at least one row merger. By the way the interconnection gadget was constructed, this means that each clause is evaluates to true by the variable assignment α. Hence, E is satisfiable.

Let us finally remark that the MRCM instance constructed to a given 3-SAT instance is always reduced.

4 Approximation algorithm

In the preceding section, we have shown that MRCM is NP-complete. This means that a deterministic polynomial-time algorithm cannot be expected. Nonetheless, hard problems have to be dealt with in practice. The usual techniques for overcoming the mentioned difficulties are:

- (meta-)heuristics,
- approximation, and
- parameterized algorithms.

In the following, we will focus on the latter two techniques. In fact, Sec. 2 can be read as providing some heuristics for MRCM. The approximation technique aims at finding polynomial-time algorithms that do not necessarily find the optimal solution to MRCM (seen as a minimization problem, where the goal is to minimize the number of merging operations for a given matrix instance) but "only" some solution which is not "too bad". More precisely, and in contrast with mere heuristics, there is a guaranteed bound on the maximal relative "error" the approximation algorithm will make in the worst case.

As the main result of this section, we will prove:

Theorem 2': MRCM is approximable by a factor of four.

This means that there is an algorithm that, given an instance M of MRCM, returns a set of merge operations which is at most four times as large as the optimal one.

The algorithm itself is quite simple and in fact similar to the factor-2-approximation for VERTEX COVER algorithm attributed to Gavril, see [1, p. 17]:

Algorithm MRCM-Approx:

Set $S := \emptyset$.
As long as there is a one in the matrix M do:
  Pick an index i, j such that $M[i, j] = 1$;
  $S := S \cup \{\mathtt{RM}(i-1, i), \mathtt{RM}(i, i+1), \mathtt{CM}(j-1, j), \mathtt{CM}(j, j+1)\}$;
  Update the matrix M accordingly
Return S

Obviously, this algorithm will finally find a feasible solution S. What about the quality of S? Each time S is updated in the loop, one out of the four merging operations added to S must be performed. Hence, the ratio follows. More formally speaking, we can apply a local ratio argument as worked out by Bar-Yehuda [2]. The obvious details are left to the reader.

5 A fixed parameter approach

A naive brute-force algorithm for MRCM simply tests all $\binom{n+m-2}{k}$ possible solving strategies, given an $n \times m$ matrix M and a parameter k.

In contrast, in the fixed parameter approach, we are looking for algorithms with running time $f(k)p(n)$, where f is an arbitrary function and p is some polynomial. Then, we call a problem *fixed parameter tractable*. Further details can be found in the monograph [3].

As main result of this section, we show that MRCM is fixed parameter tractable. More precisely, we can prove:

Theorem 3': MRCM can be solved in time $O(4^k mn)$, given an $n \times m$ matrix M and a parameter k.

The following algorithm is basically an attempt to turn MRCM-approx into a search-tree algorithm.

Algorithm MRCM-SEARCHTREE:

```
search-MRCM(M, k, S):
If M has no ones, then
  output S and return TRUE
else if k > 0 then
  pick some index (i, j) such that M[i, j] = 1;
  //branch according to the four fixing possibilities:
  search-MRCM(RM(i − 1, i)(M), k − 1, S ∪ {RM(i − 1, i)});
  search-MRCM(RM(i, i + 1)(M), k − 1, S ∪ {RM(i, i + 1)});
  search-MRCM(CM(j − 1, j)(M), k − 1, S ∪ {CM(j − 1, j)});
  search-MRCM(CM(j, j + 1)(M), k − 1, S ∪ {CM(j, j + 1)});
else return FALSE
```

At the start, we call search-MRCM$(M, k, \emptyset)$, where M is the input matrix and k the given parameter. In the above algorithm and the following algorithms, if op is a merging operation and M some matrix, then op(M) denotes the matrix obtained from M after having applied the merging operation op. Moreover, obvious refinements are necessary to cover the situation that the chosen index (i, j) lies on the border of the matrix; this basically reduces the branches by one or two (in the "corner case"). This will be used to improve this first version of a search-tree algorithm in the next section.

A further heuristic idea which connects with VERTEX COVER and hence is a good connection to Sec. 7 concludes our considerations here. Namely, it is possible to interpret a given matrix $M \in \{0,1\}^{n \times m}$ as (a "quarter" of) the adjacency matrix of a bipartite undirected graph B_M. In B_M, we have row vertices $r_1, \ldots, r_n$ and column vertices $c_1, \ldots, c_m$. r_i and c_j are connected by an edge if and only if $M[i, j] = 1$. Let us associate $\{r_i, r_{i+1}\}$ to the operation RM$(i, i+1)$, and similarly $\{c_i, c_{i+1}\}$ to the operation CM$(i, i+1)$. Let $A(S)$ denote the set of vertices of B_M associated to a set S of merging operations.

Then, we can show:
Lemma: Let (M, k) be an instance of MRCM and let S be a solution to this instance. Then, $A(S)$ is a vertex cover set of B_M with $|A(S)| \leq 2k$.

The algorithmic good news about this lemma is that VERTEX COVER can be solved in polynomial time on bipartite graphs. Hence, that lemma can be used to *prune the search tree* exhibited before: whenever the problem has been reduced to (M', k') such that the minimum vertex cover of $B_{M'}$ is larger than $2k'$, the search can stop, since there can never be a solution to the MRCM problem on that branch according to the lemma.[1]

6 Improvements to approximation and search tree algorithms

In this section, we are going to show how to improve the approximation algorithm to a factor-3-approximation and the parameterized algorithm to an $O(3^k mn)$ algorithm. The idea is to exploit the better approximation behaviour (and better branching) at ones in "borderlines". The key is the following reduction rule:

0-borderline-rule: If (M, k) is an instance of MRCM with a *0-borderline*, i.e., either the first row or the last row or the first column or the last column is completely consisting of zeros, then (M, k) can be reduced to (M', k), where M is obtained from M by deleting the 0-borderline.

Lemma: The 0-borderline-rule is sound for matrices M which contain more than two lines of the type of the considered 0-borderline.

Proof: If (M', k) is a YES-instance, M is k-mergible, as well. Conversely, let (M, k) be a YES-instance. Let $S = \{m_1, \ldots, m_k\}$ be a sequence of merging operations solving M. If S does not contain a merger with the 0-borderline omitted to obtain M', then there is surely a "corresponding" sequence of mergers for solving M', as long as M' contains more than one line. If S contains a merger with the 0-borderline, say m_1, we claim that there is a merger m_1' such that $S' = S \cup \{m_1'\} \setminus \{m_1\}$ is a solution of (M, k), hence also yielding a solution of (M', k). W.l.o.g., and to simplify the text, let $m_1 = \mathrm{RM}(1, 2)$. Namely, if we replace m_1 with $m_1' = \mathrm{RM}(2, 3)$, then we can observe:

- After applying m_1, the "new" first row of the obtained matrix M_1 is a 0-borderline, while the "new" second row of M_1 is just the "old" third row of M.
- After applying m_1', the first row is unchanged, i.e., it is a 0-borderline. The "new" second row in M_1' is obtained by merging the second and third row of M.

Since the other rows are not affected by the mergings under consideration, the only difference between M_1 and M_1' lies in the second row. Since the second row of M_1' can

[1] L. Branković recently found a way to use the B_M construction to get a factor-2-approximation for a given MRCM instance M.

be obtained by merging the second row of M_1 with some other row, it is clear that, whenever $M_1[2,i] = 0$, then $M_1'[2,i] = 0$. Hence, M_1' is "easier" than M_1, implying that the sequence $S \setminus \{m_1\}$ which solves M_1 will also solve M_1'.

By intercalating the use of the 0-borderline-rule with

- pick a one on a borderline and fix it by taking the three possible fixes into the solution (in the case of an approximation) OR with
- pick a one on a borderline and branch according to the three possible fixes (in the case of a parameterized algorithm),

we obtain the following two strengthened theorems:

Theorem 2: MRCM is approximable by a factor of three.

Theorem 3": MRCM can be solved in time $O(3^k mn)$, given an $n \times m$ matrix M and a parameter k.

We like to further improve on Theorem 3" in the following. One idea would be, as "usual" with parameterized algorithm design, to dive into a further case analysis. We give the flavour of this idea in the following:

Again, in order to simplify formulations, we will talk more specifically about rows and columns, but their "meaning" can be always interchanged. Similarly, "first" can be also read as "last" by symmetry.

Observation 1: An instance (M,k) which contains two ones (say in rows x and y) in the first column can be fixed by using the following branch:

1. Solve $(M_1, k-1)$, where M_1 is obtained from M by CM$(1,2)$.
2. Solve $(M_2, k-2)$, where M_2 is obtained from M by RM$(x-1,x)$ and RM$(y-1,y)$.
3. Solve $(M_3, k-2)$, where M_3 is obtained from M by RM$(x-1,x)$ and RM$(y,y+1)$.
4. Solve $(M_4, k-2)$, where M_4 is obtained from M by RM$(x,x+1)$ and RM$(y-1,y)$.
5. Solve $(M_5, k-2)$, where M_5 is obtained from M by RM$(x,x+1)$ and RM$(y,y+1)$.

The running time T of this recursion alone can be estimated as

$$T(k) \leq T(k-1) + 4T(k-2) + O(1).$$

Therefore, $T(k) \approx 2.5616^k$ in this case. Observe that the "branching behaviour" is improved if x and y are neighbours or if there are more than 2 ones in a borderline.

Applying the "nice branch" of Observation 1 to all borderlines (intercalated with the 0-borderline-rule) yields an instance where at most one one is in each borderline.

Unfortunately, the further case analysis we did was not strong enough to finally improve on the overall performance of the algorithm. Still, we were able to improve on Theorem 3" by using different ideas exhibited in the next section.

7 Further relations to VERTEX COVER

As already indicated, many of the above results are quite analogous to the ones obtained for VERTEX COVER. Is this just by chance? No, not at all. We will exhibit two relations in the following, where the second one will allow us to further strengthen Theorem 3".

7.1 4-HITTING SET

In fact, the way we found our NP-hardness results was mostly thinking about MRCM as a variant of 4-HITTING SET which can be seen as a generalization of VERTEX COVER to hypergraphs where each hyperedge is incident to (at most) four vertices.

More formally, we are facing the following problem:

4-HITTING SET (4HS)
Given: A hypergraph $G = (V, E)$ where each hyperedge is incident to (at most) four vertices
Parameter: positive integer k
Question: Is it possible to find a cover $C \subseteq V$ with $|C| \leq k$, i.e., each hyperedge is incident to at least one vertex in C?

By trivial reduction from VERTEX COVER, it is easy to see that 4HS is NP-hard. Moreover, its 4-approximability and the existence of an $O(4^k|G|)$ algorithm are relatively straightforward. We refer to [11] for an improved parameterized algorithm in this case.

In fact, it is tempting to model MRCM as a special case of 4HS, namely by considering the merging operations as vertices and introducing hyperedges whenever a one has to be repaired.

Due to further interactions between ones to be fixed in the given MRCM instance M, this translation into 4HS is only true when the immediate horizontal and vertical neighbours of each one in M are zeros. We were able to show that this can be enforced in time $O(3^k mn)$, but then the best-known algorithm for 4HS is currently worse than the $O(3^k mn)$, so that we did not go in that direction any further. If better algorithms for 4HS were known, than it might be an idea to revisit these connections once more.

Since in the reduction proving NP-hardness of MRCM we never introduced consecutive ones in a line, the same proof can be used to show NP-hardness of the following bipartite version of 4HS which could be of independent interest:

Bipartite 4-Hitting Set (B4HS)
Given: A hypergraph $G = (V = V_1 + V_2, E)$, a linear order on V_1 and V_2, where each hyperedge is incident to four vertices, two of them are neighbours in V_1 and the other two neighbours in V_2
Parameter: positive integer k
Question: Is it possible to find a cover $C \subseteq V$ with $|C| \leq k$, i.e., each hyperedge is incident to at least one vertex in C?

Corollary: B4HS is NP-complete.

In fact, the reduction proving NP-hardness of MRCM was obtained by showing NP-hardness of B4HS in a suitable way, since this problem appears to be intuitively more appealing. The reader is encouraged to draw the hypergraphs corresponding to the clause and interconnection gadgets in the given NP hardness proof in order to discover the similarities with the well-known construction for showing NP-hardness of vertex cover.

7.2 vertex cover

Sometimes, it is useful to look at special cases of the problem under consideration. In our case, considering the following sub-problem turns out to be fruitful:

Matrix Row Merging (MRM)
Given: $n \times m$ $\{0,1\}$-matrix M
Parameter: positive integer k
Question: Is it possible to get the all-zeros-matrix by merging at most k neighbouring rows?

Obviously, an instance of MRM M has a (feasible) solution iff there is at least one 0 in each column of M, see Obs. 1 in Sec. 2.

This can be seen best by exhaustively applying the following reduction rule, given some MRM instance (M, k):

Row merging rule: If the first row of M contains a one, merge that row with the second row to obtain an instance $(M', k-1)$. If the last row of M contains a one, merge that row with the penultimate row to get an instance $(M', k-1)$.

The correctness of that rule is evident: a one in the first row can only be fixed by RM(1,2). Call an instance *row-reduced* if the row merging rule is not applicable. A row-reduced MRM instance is not feasibly solvable if and only if it consists of only one row which contains a one somewhere.

Conversely, every row-reduced instance (M, k) having one 0 in each column can be converted into an equivalent instance (G_M, k) of vertex cover:

The vertices of G_M are the row merging operations. For simplicity, write $r_i = \mathrm{RM}(i, i+1)$. So, $V(G_M) = \{r_1, \ldots, r_{m-1}\}$. Put an edge between r_i and r_j, $i < j$, if and only if there is a column c in M in which $M[r, c] = 1$ for all $i < r \leq j$. The correctness of this translation is seen by the following lemma.

Lemma: Let M be a row-reduced matrix having more than one row.

1. If S is a set of row mergers solving M, then S is a vertex cover of G_M.
2. If S is a vertex cover of G_M, then S is a solution of M.

For the proof, it is crucial to observe that a sequence of ones $M[r,c] = 1$ for $i < r \leq j$ can be fixed by any $j-i+1$ out of the $j-i+2$ row mergers $\{r_i, \ldots, r_{j+1}\}$.[2] Conversely, the vertices $\{r_i, \ldots, r_{j+1}\}$ will form a clique in G_M. Therefore, (at least) $j-i+1$ out of these clique vertices belong to a feasible cover set.

Further properties of the translation are:

1. If G_M is split into two components, say there is no path between r_i and r_{i+1}, this happens iff M's $i+1$th row contains no ones.
2. Each connected component C has a sequence of consecutive rows as its vertex set, i.e., C can be described by (i,j) such that $C = \{r_\ell \mid i \leq \ell \leq j\}$. Moreover: $\{r_\ell, r_{\ell+1}\}$ are all edges in G_M, with $i \leq \ell < j$, forming the *backbone* of G_M.
3. If $\{r_\ell, r_{\ell''}\}$ is an edge in G_M, then for all $\ell \leq \ell' \leq \ell''$, $\{r_\ell, r_{\ell'}\}$ and $\{r_{\ell'}, r_{\ell''}\}$ are edges in G_M (interval property).[3]

In other words, G_M forms an *interval graph*, a notion introduced by G. Hajos [8]. Further important properties were exhibited in [5, 9], also see the book [4].[4] On these graphs, VERTEX COVER can be solved in polynomial time, since interval graphs are cocomparability graphs and hence perfect graphs. Improving on the "standard" $O(|V|^{2.5})$-algorithm for computing vertex covers in perfect graphs by maximum matchings, dynamic programming can be used to derive a linear time algorithm[5] (also see [7, 10]):

Lemma: Given a feasible instance M of MRM, MINIMUM VERTEX COVER for G_M can be solved in linear time.

Proof: Assume w.l.o.g. that G_M is connected. Let $V = \{r_1, \ldots, r_{m-1}\}$ be the vertex set of G_M. Let $C(r_i)$ be some minimum vertex cover of $G(\{r_i, \ldots, r_{m-1}\})$. Let $c(r_i) = |C(r_i)|$. Then, clearly, $C(r_{m-1}) = \emptyset$ and $c(r_{m-1}) = 0$. For consistency, we will also set $C(r_m) = \emptyset$ and $c(r_m) = 0$. We now claim that

$$c(r_i) = \min\{1 + c(r_{i+1}), \delta_i(r_i) + c(r_{i+\delta_i(r_i)+1})\}.$$

Here, δ_i denotes the degree of the corresponding vertex within the induced graph $G(\{r_i, \ldots, r_{m-1}\})$. The correctness of this formula is best seen by considering how to compute the corresponding cover sets $C(r_i)$. Assume that $C(r_{i+1}), \ldots, C(r_m)$ are already computed (induction hypothesis). Then, for $C(r_i)$ we obtain two subcases:

[2] Confer also the box fixing lemma in the next section.

[3] Basically, this shows that G_M is also a chordal graph.

[4] A good online source of information on intersection graphs is the "Journey through Intersection Graph County" by Erich Prisner, available at:
`http://www.math.uni-hamburg.de/spag/gd/mitarbeiter/prisner/Pris/Rahmen.html`

[5] We provide a version of this algorithm to keep the exposition of the overall algorithm self-contained.

1. If $r_i \in C(r_i)$, then all edges incident with r_i are covered, so that exactly those edges in $G(\{r_{i+1}, \ldots, r_{m-1}\})$ remain to be covered. Hence, we get

$$\{r_i\} \cup C(r_{i+1})$$

as one candidate cover.

2. Alternatively, $r_i \notin C(r_i)$. Then, all neighbours of r_i within the induced graph $G(\{r_i, \ldots, r_{m-1}\})$ are in $C(r_i)$. Due to the interval property, these neighbours are $r_{i+1}, \ldots, r_{i+\delta_i(r_i)}$. The only uncovered edges belong therefore to $G(\{r_{i+\delta_i(r_i)+1}, \ldots, r_{m-1}\})$. This makes

$$\{r_{i+1}, \ldots, r_{i+\delta_i(r_i)}\} \cup C(r_{i+\delta_i(r_i)+1})\}$$

an alternative candidate cover.

Let us re-consider Example 3; the corresponding row-reduced instance looks as follows:

	1	2	3	4	5	6	7	8	9
1									
2				1	1				
3		1		1					
4				1					
5						1			
6	1								
7			1					1	
8									

For simplicity, we labelled the row obtained by merging the first two rows 1, and the row obtained by merging the last two rows 8.

The connected graph G_M has vertices $\{r_1, \ldots, r_7\}$. Besides the backbone connections we have the following edges: $\{r_1, r_3\}$, $\{r_1, r_4\}$, and $\{r_2, r_4\}$. The dynamic programming solution for a minimum vertex cover on G_M can be summarized in tabular form:

	C	c
r_7	$\emptyset$	0
r_6	$\{r_6\}$	1
r_5	$\{r_6\}$	1
r_4	$\{r_4, r_6\}$	2
r_3	$\{r_4, r_6\}$	2
r_2	$\{r_2, r_4, r_6\}$	3
r_1	$\{r_1, r_2, r_4, r_6\}$	4

Overall, this translates into the following "rows only" solution to that instance: RM(0,1) (enforced by row merging rule), RM(1,2), RM(2,3), RM(4,5), RM(6,7) (these four operations inferred by the vertex cover algorithm), RM(8,9) (enforced by row merging rule). So, as we already saw when first discussing this example, row

mergings aren't a very good choice here. It is easy to check that for the corresponding "column merging only" instance, the solution derived in Example 3 gives a vertex cover for the "column merging graph."

Observe that conversely each interval graph can be converted into a row-reduced MRM instance; the interior of an interval basically corresponds to a sequence of ones in a certain column of the MRM instance. This makes the following lemma more generally applicable.

Lemma: An MRM instance can be completely solved (i.e., reduced to some $1 \times m$-matrix) by applying the row merging rule and the 0-borderline rule and Rule 4 from Sec. 2.

Proof: We consider a $1 \times m$-matrix as "completely solved;" more precisely, the given instance is a YES-instance if the obtained reduced matrix is a 1×1-matrix containing a zero, and it is a NO-instance if the obtained reduced matrix is a $1 \times m$-matrix containing a one.

The first row of an MRM instance with more than one row either contains a one (triggering the row merging rule) or only zeros. If the first row contains only zeros and the second row contains only zeros, then Rule 4 is applicable. If the first row contains only zeros and the second row contains a one, then either the matrix does not contain more than two rows, in which case the row merging rule is applicable, or it has more than two rows, in which case the 0-borderline rules applies.

So, we have shown that any MRM instance will be finally reduced to some $1 \times m$-matrix, since any instance with more than one row triggers (at least) one reduction rule. If the obtained $1 \times m$-matrix only consists of zeros, Rule 4 can be used to finally arrive at a 1×1-matrix having one zero.

So, in our case the derived reduction rules immediately yield a deterministic solving algorithm, without the need of dynamic programming at all. This could be of independent interest, since it gives a simple deterministic linear-time algorithm for computing minimum vertex covers or maximum independent sets in interval graphs.

Let us write down this algorithm more formally:

Algorithm MRM-SOLVER:

```
Let M,k be an MRM instance.
Call S:=solve-MRM(M,k,∅,0).
If M contains a one and S=∅, then M,k is a NO-instance.
Otherwise, M is a YES-instance, solved by S.

solve-MRM(M,k,S,offset) returns set of mergings:
  Let r be the number of rows in M.
  If ((r=1) ∨ (k≤ 0)) and M contains a one, return ∅.
  If r=1 and M contains only zeros, return S.
  //Otherwise: r>1
  If the first row of M contains a one, then
    op:=RM(1,2);
    return solve-MRM(op(M),k−1,S∪{RM(1+offset,2+offset),offset+1).
```

```
  else // the first row of M contains only zeros
    if the second row of M contains only zeros or if r > 2 then
      op:=delete first row;
      return solve-MRM(op(M),k,S,offset+1).
    else // r = 2 and the second=last row contains a one
      op:=RM(1,2);
      return solve-MRM(op(M),k−1,S∪{RM(1+offset,2+offset),offset+1).
```

Here, the variable `offset` counts the number of recursive calls.

Reconsidering again Example 3 as an MRM instance, the preceding algorithm will generate, when called with parameter 6, the following solution (in that order): RM(0,1), RM(2,3), RM(3,4), RM(4,5), RM(6,7), RM(8,9).

8 The final parameterized algorithm

Having found this simple solution for row (or column) mergers only, we can now formulate an algorithm for which we can show a further improved running time:

Algorithm MRCM-PARAM:

```
Let (M,k) be the given MRCM instance.
A is an initially empty auxiliary matrix.
Call S:=solve-MRCM(M,k,∅,0,k,A,1,0).
If M contains a one and S = ∅, then M,k is a NO-instance.
Otherwise, M is a YES-instance, solved by S.

solve-MRCM(M,k,S,ℓ,A,a,offset) returns set of mergings:
// ℓ is an auxiliary parameter.
// a is pointing to the ''next'' free row in A
Let c be the number of columns of M.
   Apply:
   a) // 0-borderline-rule to columns:
      if the first column of M contains only 0s and if c > 2 then
        op:=delete first column;
        return solve-MRCM(op(M),k,S,ℓ,A,a+1,offset+1).
   b) if the first column of M contains 2s but no 1s then do
        copy the first column of M into the as column of A,
           thereby turning 2s into 1s;
        op:=delete first column;
        return solve-MRCM(op(M),k,S,ℓ,A,a+1,offset+1).

  // After applying a) and b),
  // the first column of M contains a 1 or c ≤ 2.
  // The remaining simple cases are solved with the MRM-solver
  If (c ≤ 1) ∨ (ℓ < 1) then do
    copy all columns from M into A, starting at position a,
```

```
      thereby turning 2s into 1s;
  return solve-MRM(A, k, S, 0).

// Branch according to two subcases
op:=CM(1,2);
S_update := S ∪ {CM(1 + offset, 2 + offset)};
S' :=solve-MRCM(op(M),k − 1,S_update,ℓ − 1,A,a,offset+1).

If S' ≠ ∅ then return S'
else do // look into alternative branch
  if c = 2 then do // the only remaining simple case
    copy all columns from M into A, starting at position a,
       thereby turning 2s into 1s;
    return solve-MRM(A, k, S, 0).
  else // c > 2 Hence, the first column of M contains a one.
    λ := 0;
    set the ath column of A to be the all-zero-vector;
    for each row i in the first column of M with M[i, 1] = 1 do
      λ := λ + 1;
      for each column j > 1 of M with M[i, j] = 1 do
        M[i, j] := 2;
    for each row i in the first column of M with M[i, 1] ≠ 0 do
       A[i, a] := 1;
    op:=delete first column;
    return solve-MRCM(op(M),k,S,ℓ − λ/2,A,a + 1,offset+1).
```

The idea of this algorithm is pretty simple and similar to the previous case: a one in the first column can be either solved by a column merger or by some row mergers. The trick is that we defer solving by row mergers till the very end, which is possible due to Observation 3 in Sec. 2. Observe that $\lambda \geq 1$ due to the fact that M contains at least one one in the first column.

We allow ourselves to override some ones in M by twos, namely, if we decide that the corresponding row containing that one should be resolved by a row merger. Then, we should not consider the possibility of repairing a column containing only zeros and twos by column mergings (at all). Rather, in this sense a two would be treated as a zero by the 0-borderline-rule within M, deferring the solution to the polynomial-time phase for G_A. This justifies part b) of the reduction in M. Still, if we decide to merge the first column with the second which may contain a two, then this two is treated as if it were a one. More precisely, merging within M is then done according to the following algebraic laws:

- $0 \quad \text{AND} \quad X = 0 \qquad \text{for } X = 0, 1, 2;$
- $X \quad \text{AND} \quad X = X \qquad \text{for } X = 0, 1, 2.$

Note that we will never merge a one with a two, since all ones in a row will be turned into twos (when we assume that that row is fixed by row merging) or not. Hence, the given rules cover all cases that may occur.

The idea behind the new variable ℓ is that for each row which we decide to be resolved by row mergers only, we can be sure to put aside at least one half of a parameter value, since one row merger can completely resolve at most two rows to be fixed by row mergers only. This "book-keeping trick" (which might be in fact wider applicable: use simple approximation bounds to account for cases to be resolved later in a deterministic, polynomial-time fashion) gives us the following recursion:

$$T(\ell) \leq T(\ell - 1) + T(\ell - 1/2) + O(1)$$

Assuming $T(\ell) = c^\ell$ yields $c = (3 + \sqrt{5})/2 \approx 2.6181$. Alternatively, $c = \phi^2 = \phi + 1$, where $\phi \approx 1.6181$ is the number of the golden ratio.

Hence, we may state as our final version:

Theorem 3: MRCM can be solved in time $O(2.6181^k mn)$, given an $n \times m$ matrix M and a parameter k.

Let us follow the work of this algorithm by having another look at the first example, solving it as instance $(M, 3)$. For clarity, we put the value of the `offset` variable into each table.

`offset` = 0	1	2	3	4	5	6	7	8	9
1	1			1					1
2									
3	1								
4			1	1			1		
5									
6									
7				1					1

The first branch would take CM(1,2), and one application of the 0-borderline-rule leaves us with a new instance $(M\{1\}, 2)$, using curly brackets to denote the different subcases:

`offset` = 2	1	2	3	4	5	6	7
1		1					1
2							
3							
4	1	1			1		
5							
6							
7		1					1

Alternatively, the second branch would copy the first column of M into A and colour some 1s into 2s, arriving at the following matrix instance $(M\{2\}, 2)$, after one application of the 0-borderline-rule:

offset = 2	1	2	3	4	5	6	7
1		2					2
2							
3							
4	1	1			1		
5							
6							
7		1					1

To shortcut the exposition, let us now focus on what further happens with this second branch, i.e., with $(M\{2\}, 2)$: Again, we have CM(1,2) as first alternative, yielding as next instance $(M\{2.1\}, 1)$. Notice that since the current offset is two, the operation CM(3,4) will be put into the partial solution in this branch. So, we arrive at:

offset = 3	1	2	3	4	5	6
1						2
2						
3						
4	1			1		
5						
6						
7						1

Obviously, this is a dead end (it cannot be repaired by only one merging operation). Alternatively, deferring the row merging yields $(M\{2.2\}, 1.5)$:

offset = 3	1	2	3	4	5	6
1	2					2
2						
3						
4	2			2		
5						
6						
7	1					1

The matrix A looks up to now as follows:

	1	2
1	1	
2		
3	1	
4		1
5		
6		
7		

Since column mergers alone surely would not resolve $(M\{2.2\}, 1.5)$, let us follow up the situation assuming row mergers. At an intermediate state, in $M\{2.2\}$ all ones would be turned into twos. The copy operations would then finally turn A besides one missing 0-column) into the original matrix M. `solve-MRM`$(A, 3, \emptyset, 0)$ will then return as solution

$$S = \{\mathrm{RM}(1,2), \mathrm{RM}(3,4), \mathrm{RM}(6,7)\}.$$

Of course, there are many places where we could improve on this algorithm, at least in practice:

1. At the beginning, we can use the rules derived in Sec. 2 in a kind of preprocessing phase. Especially, Rules 1 and 2 may give some (advantageous) 2^k-branching behaviour for "lines with many ones."

2. Heuristics can be used to prune search branches as early as possible, see, e.g., the lemma derived in Sec. 5.

3. In the same spirit, we can improve on the estimate for λ by repeatedly running `solve-MRM`.

4. Applying Observation 1 from Sec. 6 also slightly improves on the running time of our final algorithm, although this gain might be overridden by the additional (implementational) complexity, since now row mergers are to be implemented "inbetween", affecting both M and A.

5. The 0-borderline-rule can be also applied to rows. Again, this may affect both M and A.

In this more practical spirit, we mention in the following a couple of more rules which we encountered but could not make use of for improving our analysis:

- Whenever the instance (M, k) contains a one at (i, j) such that in the k-Manhattan-neighbourhood of (i, j) there are only ones, (M, k) is a NO-instance. This rule can be used for pruning search-tree branches.

- If ℓ' and ℓ'' are the neighbouring lines of line ℓ such that the line vectors v' and v corresponding to line index ℓ' and ℓ obey $v' \leq v$ (comparison by $0 < 1$, componentwisely extended), then never merge lines ℓ and ℓ' but prefer to merge ℓ and ℓ'' instead.

  To understand this rule, observe that merging rows ℓ and ℓ' gives the line vector v' again (due to $v' \leq v$), with neighbouring line ℓ'', while merging ℓ and ℓ'' keeps the "first" line ℓ' unchanged and produces a neighbouring line which is, as a merger of line ℓ'' with something, never greater than ℓ''.

  In fact, the 0-borderline-rule might be viewed as special case of the preceding rule.

As a rule of thumb, it would be also advantageous to make as many "good branches" as possible in the very beginning, bringing the parameter in the generated subinstances as much down as possible. Typical candidates for a good branching are:

- lines with many ones;
- agglomerations of ones.

To understand what is meant by the second point, we give some results in the following. These are also of importance to derive the "parameterized reduction" to 4-HITTING SET as mentioned in the previous section, since it may be used to destroy situations where a one is neighbour of another one.

Box fixing lemma: If M contains an $r \times c$ submatrix completely filled with ones (i.e., an $r \times c$ *box*), then, in order to fix M, either at least r row merging operations or at least c column merging operations are necessary. More precisely, if $i+1, \ldots, i+r$ describe the rows and $j+1, \ldots, j+c$ the columns of the box B, then either r out of the $r+1$ row merging operations $\mathrm{RM}(i, i+1), \ldots, \mathrm{RM}(i+r, i+r+1)$ or c out of the $c+1$ column merging operations $\mathrm{CM}(j, j+1), \ldots, \mathrm{RM}(j+c, j+c+1)$ are necessary to fix all ones in B.

This justifies the following box branching rule: If M contains a box of size $r \times c$ with $r+c \geq 3$, then branch according to the $r+1$ possibilities for row mergings and the $c+1$ possibilities for column mergings.

If $r+c=3$, we arrive at a running time described by the recurrence

$$T(k) \leq 3T(k-2) + 2T(k-1) + O(1),$$

which can be solved by $T(k) = 3^k$. Obviously, "larger boxes" may yield much better branches. For example, in the case $r=3$ and $c=1$,

$$T(k) \leq 4T(k-3) + 2T(k-1) + O(1),$$

which means that $T(k) \leq 2.60^k$, and $r=c=2$ gives

$$T(k) \leq 6T(k-2) + O(1),$$

implying that $T(k) \leq 2.45^k$.

9 Conclusions

This paper demonstrated how a problem emanating from the theory of statistical databases can be attacked by making use of parameterized algorithmics or of approximation algorithms.

In passing, we developed a novel book-keeping technique for computing the behaviour of parameterized algorithms which might be of independent interest. An interesting related question is here whether a similar technique can be also used to assess the approximation behaviour of (similarly designed) approximation algorithms, at least in some randomized setting. This would not be too surprising due to the usually closely linked nature of approximation and search-tree algorithms (if the search-tree branches don't get too tricky, but this is not the case in our problem).

Conversely, this "matrix puzzle" can be also seen as a sort of solitaire game, easily implementable on a computer. Here, it might be also interesting to consider different "merging rules"; at present, we considered the rule of ANDing neighbouring rows or columns, but other Boolean operations are thinkable as well. It is not immediately clear whether, seen on a classical complexity scale, these problems would still be NP-complete, constant-factor approximable and fixed parameter tractable. Possibly, some of them are easier (up to trivial for the constant boolean functions), while others might be PSPACE-hard.

Last but not least, the MRCM problem also contains a "didactic" component: since it is easy to formulate and since its NP-hardness proof is quite similar to the one found for VERTEX COVER in textbooks introducing the Theory of Computation, it might be a good candidate for an example explained in a tutorial accompanying a lecture on Computation Theory.

Acknowledgment: We like to thank Ljiljana Branković, Stephan Chalup, Mike Fellows, Pablo Moscato and Christian Sloper for interesting discussions on this topic. Special thanks to Ljiljana for communicating us the problem. We express our hope that this paper will stir further fruitful cooperation on database-related problems. The author is indebted to the referee because he pointed to some flaws in the submitted version of the paper.

References

[1] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, M. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation.* Springer, 1999.

[2] R. Bar-Yehuda. One for the price of two: a unified approach for approximating covering problems. *Algorithmica*, 27:131–144, 2000.

[3] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer, 1999.

[4] P. C. Fishburn. Interval Orders and Interval Graphs. Wiley, 1985.

[5] D. R. Fulkerson and O. A. Gross. Incidence matrices and interval graphs. *Pacific J. Math.* 15:835–855, 1965.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability.* Freeman, 1979.

[7] U. I. Gupta, D. T. Lee and J. Y. T. Leung. Efficient algorithms fo interval graphs and circular-arc graphs. *Networks* 12:459–467, 1982.

[8] G. Hajós. Über eine Art von Graphen. *Internat. Math. Nachr.* 11, 1957, Problem 65.

[9] P. C. Gilmore and A. J. Hoffman. A characterization of comparability graphs and of interval graphs. *Canad. J. Math.* 16:539–548, 1964.

[10] W. L. Hsu and K. H. Tsai. Linear time algorithms on circular-arc garphs. *Information Processing Letters* 40:123–129, 1991.

[11] R. Niedermeier and P. Rossmanith. An efficient fixed parameter algorithm for 3-hitting set. *J. Discrete Algorithms*, To appear, 2002.

(Received 19 Dec 2002; revised 28 Aug 2003)