SIAM J. COMPUT.
Vol. 14, No. 3, August 1985

015

APPROXIMATION OF THE CONSECUTIVE ONES MATRIX AUGMENTATION PROBLEM*

MARINUS VELDHORST†

Abstract. In this publication we will prove a number of negative results concerning the approximation of the NP-complete CONSECUTIVE ONES MATRIX AUGMENTATION problem. We will characterize a large class of simple algorithms that do not find a near optimum augmentation of their input matrices. We will show that there are matrices for which these algorithms find augmentations that are even far from optimal. These results are important for the analysis of a sparse matrix storage scheme.

Key words. consecutive ones property, sparse matrix storage scheme, NP-completeness, approximation algorithms, analysis of algorithms

For many years much research has been done in the design of efficient storage schemes for sparse matrices. Many storage schemes have proposed (cf. [3], [6], [9], [11]). Some of them are rather efficient for each distribution of the nonzero elements over the matrix (cf. [11]). Others are only efficient for specific distributions (cf. [9]). If one has to choose a storage scheme for sparse matrices in a practical problem, the choice will depend e.g. on the operations to be applied to the matrix. In case only matrix-vector products have to be computed the following data structure for sparse matrices may be very efficient:

Use a data structure in which a sparse matrix is stored as a sequence of rows and each row is stored in such a way that all zero elements at both ends will not be stored.

We will call this the *rowmat* data structure and Fig. 1 gives an example how it can be used for a specific matrix.

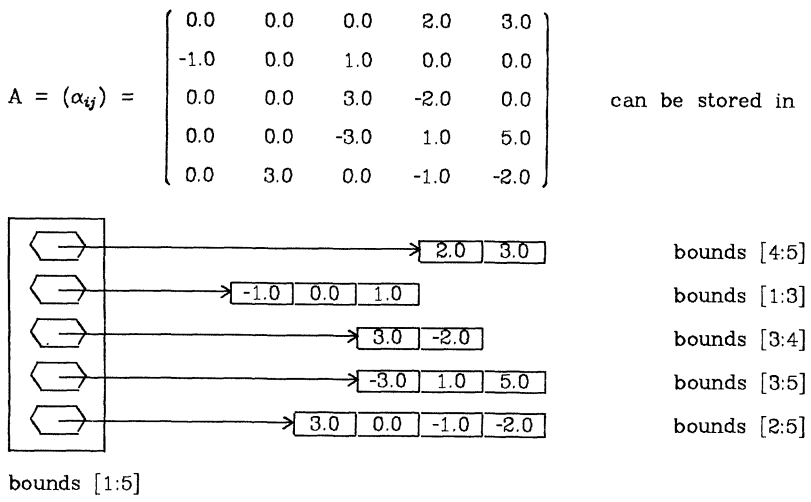


FIG. 1. *Storing a matrix in a rowmat data structure.*

* Received by the editors March 1, 1983. The research for this publication was done while the author was at the Department of Computer Science, University of Utrecht, the Netherlands. This publication was finished when the author was visiting the Department of Electrical Engineering and Computer Science, Princeton University, Princeton, New Jersey, on a grant from the Netherlands organization for the advancement of pure research (Z.W.O.).

† Department of Computer Science, University of Utrecht, Utrecht, the Netherlands.

In this paper we will discuss the storage optimality of the rowmat data structure, i.e. the amount of storage required if we store matrices in rowmat data structures. Because all nonzero elements must be stored, we only need to consider the number of stored zero elements. The rowmat data structure will not prevent zero elements from being stored in memory (see Fig. 1). However in many applications the columns of the matrix may be permuted. Thus we can look for a column permutation such that the permuted matrix can be stored in a rowmat data structure without storing any nonzero elements. Unfortunately there are matrices for which such a column permutation does not exist.

If columns may be permuted, the storage optimality of the rowmat data structure is closely related to the consecutive ones property for rows of a matrix (cf. [4]). As far as we know the relation of the consecutive ones property to a sparse matrix storage scheme has never been explored before. Although the problem to find a column permutation such that a minimum number of zero elements would be stored is NP-complete (cf. [1]), this does not justify the conclusion that the rowmat data structure should not be used in practice. One way to proceed is to design a polynomial time algorithm that finds for each matrix a column permutation such that a (hopefully) small, but not necessarily minimum, number of zero elements will be stored. These algorithms are called polynomial time approximation algorithms.

In this paper we will concentrate on the design of approximation algorithms. In the first section we will review main results about the consecutive ones property (§ 1.1) and their relation to the storage optimality of the rowmat data structure (§ 1.2). For definitions in the field of complexity theory of algorithms we refer to [5]. In §§ 2, 3 and 4 we will deal with approximation algorithms to find good column permutations. In § 2 we will characterize a large class of simple algorithms: the on-line column insertion algorithms. This section also serves as an introduction for the main theorem. In § 3 we will prove the main theorem that states that any algorithm of the class of on-line column insertion algorithms must give arbitrarily bad approximations for an infinite number of matrices. This means that no on-line column insertion algorithm can guarantee a bounded error factor with regard to the minimum number of stored elements if all column permutations of the input matrix are considered. In the last section we will extend this result to much wider classes of approximation algorithms. In analyzing the rowmat data structure we are not interested in the exact value of a nonzero element, but only in the fact that it is nonzero. Therefore, we will only consider $\{0, 1\}$-matrices.

1. Consecutive ones property.

1.1. Consecutive ones property and submatrices.

DEFINITION 1.1. (cf. [4]). An $m \times n$ $\{0, 1\}$-matrix A has the *consecutive ones property for rows* (COR-*property*) if and only if there is an $n \times n$ permutation matrix P such that the ones of $B = AP = (\beta_{ij})_{1 \leqq i \leqq m, 1 \leqq j \leqq n}$ occur consecutively in each row, i.e., for each i $(1 \leqq i \leqq m)$ $\beta_{ij} = 1$ and $\beta_{ik} = 1$ imply $\beta_{ip} = 1$ for all p with $j \leqq p \leqq k$.

DEFINITION 1.2. Let $A = (\alpha_{ij})$ be an $m \times n$ matrix. A $p \times q$ matrix $B = (\beta_{ij})$ $(p \leqq m, q \leqq n)$ is a *permuted submatrix* of A, if there are sets $I = \{i_1, \cdots, i_p\}$ and $J = \{j_1, \cdots, j_q\}$ such that $\beta_{hk} = \alpha_{i_h j_k}$ for all h, k $(1 \leqq h \leqq p, 1 \leqq k \leqq q)$. If $I = \{i_1, i_1 + 1, \cdots, i_1 + p - 1\}$, $J = \{j_1, j_1 + 1, \cdots, j_1 + q - 1\}$, we say B is a *submatrix* of A and we write $B = A[i_1 : i_1 + p - 1, j_1 : j_1 + q - 1]$. We will denote row i of A by $A[i, \ ]$ and column j of A by $A[\ , j]$.

THEOREM 1.1 (cf. [12]). *A $\{0, 1\}$-matrix A has the* COR-*property if and only if no permuted submatrix of A equals one of the matrices given in Fig.* 2.

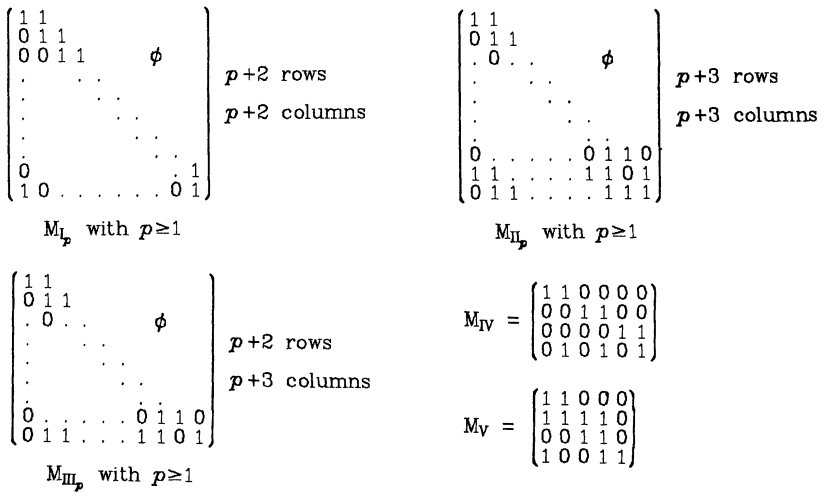


FIG. 2. *Minimal matrices not having the* COR*-property.*

If a matrix A contains a permuted M_i, then we shall refer to A as simply *containing an* M_i.

In the past, research has been done on the problem of detecting the forbidden submatrices (see Fig. 2) in an arbitrary {0, 1}-matrix A. A number of questions can be posed:

Does A contain a forbidden permuted submatrix? By Tucker's theorem the existence of a forbidden permuted submatrix is equivalent to A not having the COR-property. Booth and Lueker (cf. [2]) gave an algorithm to test whether A has the COR-property in $O(m+n+f)$ time (f is the number of nonzero elements in A). The algorithm is on line: the rows of A are processed one by one. It starts with the set S of all column permutations of A. Processing row i means that from S all permutations are eliminated which, when applied to row i, do not place the ones in row i in consecutive order. If S gets empty before all rows of A are processed, then A does not have the COR-property. Observe that this algorithm actually finds the largest p such that $A[1:p, 1:n]$ has the COR-property.

Does A contain a forbidden permuted submatrix with at least k rows? This problem has been proved NP-complete (cf. [13]). Even the problems of detecting whether A contains an M_{I}, M_{II}, M_{III}, respectively, with at least k rows are NP-complete (cf. [14], [13]). The LONGEST PATH problem for graphs can be proven to be polynomially transformable to each of these four problems.

List all forbidden submatrices of A. In [13] an algorithm has been designed which enumerates all forbidden submatrices in time polynomial in the size of A and the number of submatrices listed. The core of this algorithm consists of a subroutine that finds all permuted submatrices that have only nonzero elements on their main diagonal and their first super diagonal. Such a submatrix corresponds to an induced subgraph in a bipartite graph that is a path.

1.2. COR-property and storage optimality. In this section we will investigate the consequences for the optimization of storage if a {0, 1}-matrix does not have the COR-property. If we want to store such a matrix in a rowmat data structure, then we shall have to compromise and store embedded zero elements. In the following paragraphs we will deal briefly with two minimization problems. For related minimization problems we refer to [1], [7] and [10].

1.2.1. Augmentation.

DEFINITION 1.3. Let $A=(\alpha_{ij})$ and $B=(\beta_{ij})$ be $m\times n$ $\{0,1\}$-matrices. B is a *k-augmentation* ($k\in\mathbf{N}$) of A if $\alpha_{ij}=1$ implies $\beta_{ij}=1$ and moreover there are exactly k different pairs (i_p,j_p), $1\leqq p\leqq k$, such that $\alpha_{i_p,j_p}=0$ and $\beta_{i_p,j_p}=1$ $(1\leqq p\leqq k)$.

$\mathrm{Aug}^0(A)=\{A\}$, $\mathrm{Aug}^k(A)=\{B\colon B \text{ is a } k\text{-augmentation of } A\}$.

Suppose there is a $B\in\mathrm{Aug}^k(A)$ that has the COR-property. Let P be an $n\times n$ permutation matrix such that the ones of BP occur consecutively in each row. Then, if we store AP in a rowmat data structure, at most k zero elements of A will be stored. Moreover, if k is the least integer such that $\mathrm{Aug}^k(A)$ contains a matrix with the COR-property, then AP requires k zero elements to be stored. This leads to the following problem.

CONSECUTIVE ONES MATRIX AUGMENTATION:

Instance: a $\{0,1\}$-matrix A; an integer $k\geqq 0$.

Question: is there a p $(0\leqq p\leqq k)$ such that $\mathrm{Aug}^p(A)$ contains a matrix with the COR-property?

THEOREM 1.2 (cf. [1]). *The* CONSECUTIVE ONES MATRIX AUGMENTATION *problem is* NP-*complete.*

This means that it will be very difficult to design a practical algorithm to find the column permutation that is optimal with regard to the number of stored zero elements. In the next sections we will prove negative results concerning the existence of simple schemes for even finding near optimum column permutations. Observe that for every fixed k one can determine in polynomial time whether a $\{0,1\}$-matrix has a k-augmentation with the COR-property. There is only a polynomial number (in the size of the matrix A) of k-augmentations of A, and each k-augmentation can be tested in linear time for the COR-property.

1.2.2. Storing a number of (permuted) submatrices. If an $m\times n$ $\{0,1\}$-matrix A does not have the COR-property, then we may try to divide A into a minimum number of submatrices $A[1:p_1, 1:n]$, $A[p_1+1:p_2, 1:n]$, $\cdots$, such that each submatrix has the COR-property. Each submatrix has its own column permutation such that the ones in each row of the submatrix occur consecutively.

If it is not allowed to permute the rows of A, this problem can be solved in polynomial time (apply the algorithm of Booth and Lueker (cf. [2]) several times), but if the rows of A can be permuted, then this problem is NP-complete (cf. [10]). Even to find a maximum set of rows of A that has the COR-property is NP-complete (cf. [1]). Nevertheless we will show that the following problem can be solved in polynomial time (in the size of A):

Let A have rows $r_1,\cdots,r_m$.

Partition $R=\{r_1,\cdots,r_m\}$ into sets $R_1,\cdots,R_p$ such that

(1) $|R_i|\geqq|R_{i+1}|$ $(1\leqq i\leqq p-1)$,

(2) each R_i $(1\leqq i\leqq p)$ has the COR-property,

(3) for each i $(1\leqq i\leqq p)$ and for each row $r\in R_j$ with $j>i$ the set $R_i\cup\{r\}$ does not have the COR-property.

The following algorithm will solve the problem.

ALGORITHM 1.

(initialize $R_i:=\{r_i\}$ $(1\leqq i\leqq m)$;

while there is a j and an $r\in R_j$ such that for some $i<j$ $R_i\cup\{r\}$ has the COR-property

do $R_j := R_j \backslash \{r\}$; $R_i := R_i \cup \{r\}$;
sort $(R_i)_{1 \leqq i \leqq m}$ according to decreasing number of rows
od ;
let p be the greatest index such that $R_p \neq \emptyset$
)

If this algorithm terminates, $R_1, \cdots, R_p$ satisfy the requirements (1)-(3).

PROPOSITION 1.3. *Algorithm* 1 *terminates and does so within polynomial time.*

Proof. Define:

$$f(R_1, \cdots, R_m) = \sum_{i=1}^{m} |R_i|.\ i \quad (|R_i| \text{ is the number of rows in } R_i).$$

f can only have nonnegative integer values. With each action consisting of a deletion, an addition and an ordering, the value of f decreases. The algorithm has to terminate otherwise f would become negative. When the algorithm has processed the first line, $f(R_1, \cdots, R_m) = \frac{1}{2}m(m+1)$ and the outer loop will be executed at most $\frac{1}{2}m(m+1)$ times. It requires at most polynomial time to perform the instructions in the loop-clause. Thus the algorithm will halt within polynomial time in the size of A. □

2. Examples of approximation algorithms. In the previous section we saw that not every sparse matrix can be stored in a rowmat data structure without storing any zero elements, even if the columns are permuted. Moreover, the problem of finding a column permutation such that a minimum number of zero elements would be stored, turned out to be very hard: the problem is NP-complete. However, these results do not justify the conclusion that the rowmat data structure should not be used in practice. For such a conclusion there should be negative results in the following four directions:

a) We restrict ourselves to a special class C of matrices (which reflects the matrices used in practice) and try to design a polynomial time algorithm that finds for each matrix of C a column permutation such that a minimum number of zero elements will be stored.

b) We try to design a probabilistic ("usually efficient") algorithm that finds for each matrix a column permutation such that a minimum number of zero elements will be stored. For most matrices such an algorithm should run in polynomial time, but for a (hopefully small) number of matrices it may need exponential time.

c) We try to design a polynomial time algorithm that finds for each matrix a column permutation such that a (hopefully) small, but not necessarily minimum, number of zero elements will be stored. These algorithms are called polynomial time approximation algorithms.

d) Some combination of a), b), c).

In the §§ 2-4 we will concentrate on approximation algorithms. The main result is given in § 3. Section 2 serves as an introduction for this main result.

DEFINITION 2.1. Let $A = (\alpha_{ij})$ be an $m \times n$ $\{0, 1\}$-matrix.

$$ones(A) = \sum_{i=1}^{m} \sum_{j=1}^{n} \alpha_{ij} \quad \text{(the number of nonzero elements of } A\text{)},$$

$$store(A) = \sum_{i=1}^{m} (\max \{j: \alpha_{ij} \neq 0\} - \min \{j: \alpha_{ij} \neq 0\} + 1)$$
$$\text{with } \max(\emptyset) = 0 \text{ and } \min(\emptyset) = 1,$$

$$optstore(A) = \min \{store(AP): P \text{ is an } n \times n \text{ permutation matrix}\}.$$

An $m \times n$ matrix B is a *column permutation* of A if $B = AP$ for some $n \times n$ permutation matrix P. B is an *optimum column permutation* of A if $store(B) = optstore(A)$.

DEFINITION 2.2. Let $A=(\alpha_{ij})$ be an $m\times n$ $\{0,1\}$-matrix and B a column permutation of A. Let column i of A be column p_i of B $(1\leqq i\leqq n)$. We say that α_{i_0,j_0} of A is *stored in* B if there are j_1 and j_2 with

$$p_{j_1}\leqq p_{j_0},\quad p_{j_2}\geqq p_{j_0},\quad \alpha_{i_0,j_1}=\alpha_{i_0,j_2}=1.$$

As we have seen in § 1, the problem to find for each matrix A an optimum column permutation B of A, is NP-complete. Here we seek a polynomial time algorithm that finds a column permutation B of A (for each A) such that e.g.

$$\frac{store(B)}{ones(A)}\leqq c$$

for some fixed constant c, because otherwise there are other more appropriate data structures for sparse matrices.

PROPOSITION 2.1. *If such an algorithm exists, then* $c\geqq\frac{3}{2}$.

Proof. Consider the special case of M_{I_n} (cf. § 1.1).

$$optstore(M_{I_n})=store(M_{I_n})=2(n+1)+n+2,\qquad ones(M_{I_n})=2n+4.$$

Therefore, for every column permutation B_n of M_{I_n} we have

$$\frac{store(B_n)}{ones(M_{I_n})}\geqq\frac{optstore(M_{I_n})}{ones(M_{I_n})}=\frac{3n+4}{2n+4}.$$

Note that for $c'<\frac{3}{2}$ there is an n such that $store(B_n)/ones(M_{I_n})>c'$. Hence $c\geqq\frac{3}{2}$. □

However, it is more realistic to compare $store(B)$ with $optstore(A)$ than with $ones(A)$. If $store(B)$ is close to $optstore(A)$, then B is considered a "good" column permutation of A. Therefore, to analyze an algorithm X, we will use as a criterion the magnitude of the ratio

$$\frac{store(B)}{optstore(A)}\quad\text{with } B \text{ a column permutation of } A \text{ found by } X. \tag{4}$$

Moreover, we are interested in the asymptotic behavior of X.

DEFINITION 2.3. Let $f:\mathbf{N}\to\mathbf{R}$ be a mapping. Let X be an algorithm that takes $\{0,1\}$-matrices as input and that returns a column permutation of its input. X is an *f-approximation algorithm* (for the CONSECUTIVE ONES MATRIX AUGMENTATION problem) if there is an $N\in\mathbf{N}$ such that for all $m\times n$ $\{0,1\}$-matrices A $(m, n\geqq N)$

$$\frac{store(X(A))}{optstore(A)}\leqq f(optstore(A)).$$

As we have seen, we are only interested in c-approximation algorithms with c some constant. It should be nice if there are algorithms of that kind with a running time that is polynomial in the length of their input.

DEFINITION 2.4. A $\{0,1\}$-matrix A is said to be *clean* if the following five conditions are satified:

(i) each column of A contains at least one nonzero element,
(ii) each row of A contains at least two nonzero elements,
(iii) Each row of A contains at least one zero element,
(iv) no two rows of A are equal,
(v) no two columns of A are equal.

DEFINITION 2.5. Let A be an $m \times n$, B an $m \times p$, C an $m \times (n+1)$ and D an $m \times (n+p)$ $\{0, 1\}$-matrix and let u be a $\{0, 1\}$-sequence of length m.

$$C = A \textbf{ concat } u \quad \text{if } C[\ , 1:n] = A \text{ and } C[\ , n+1] = u.$$

$$D = A \textbf{ concat } B \quad \text{if } D[\ , 1:n] = A \text{ and } D[\ , n+1:n+p] = B.$$

Now we will give two examples of straightforward approximation algorithms that are not c-approximation algorithms for any c.

ALGORITHM 2.

```
proc BESTFIT = (matrix A)matrix:
co let A be an m × n {0, 1}-matrix co
(initialize B as the m × 0 {0, 1}-matrix;
  for k from 1 to n
  do initialize C as the m × k {0, 1}-matrix with only nonzero elements;
    for j from 1 to k
    do matrix D = B[ , 1:j − 1] concat A[ , k] concat B[ , j:k − 1];
      if store(D) < store(C) then C := D fi
    od; B := C
  od;
return B
)
```

BESTFIT processes the columns of A one by one; processing column k means that it will be inserted in B such that the current columns of B do not lose their relative order; column k is inserted in B just where it causes the least number of elements of B (including column k) to be stored.

PROPOSITION 2.2. BESTFIT *is not a c-approximation algorithm for any* $c \geqq 1$.

Proof. We have to prove that for each $N \in \mathbf{N}$ and each $c \in \mathbf{R}$ $(c \geqq 1)$ there is a $\{0, 1\}$-matrix A with at least N rows and columns such that

$$\frac{store(\text{BESTFIT}(A))}{optstore(A)} > c.$$

Let $N \in \mathbf{N}$, $c \in \mathbf{R}$ $(c \geqq 1)$; let

$$A_{p,k} = \begin{pmatrix} 0 \ldots 0 & 1 \ldots 1 & 1 \ldots 1 \\ 1 \ldots 1 & . \quad . & 0 \ldots 0 \\ . \quad . & . \quad . & . \quad . \\ . \quad . & . \quad . & . \quad . \\ . \quad . & . \quad . & . \quad . \\ . \quad . & . \quad . & . \quad . \\ . \quad . & . \quad . & . \quad . \\ . \quad . & 1 \ldots 1 & 0 \ldots 0 \\ 1 \ldots 1 & 0 \ldots 0 & 1 \ldots 1 \end{pmatrix}$$

$p+1 \quad p+1 \quad k$

$p+2$ rows,
$2p+k+2$ columns,
$k \geqq 2, p \geqq 1$.

$optstore(A_{p,k}) = store(A_{p,k}) = (p+1)^2+(p+1)(p+2)+2k$. If BESTFIT is applied to $A_{p,k}$, it returns

$$\text{BESTFIT}(A_{p,k}) = \begin{pmatrix} 1\ldots.1 & 1\ldots.1 & 0\ldots.0 \\ .\quad . & 0\ldots.0 & 1\ldots.1 \\ .\quad . & .\quad . & .\quad . \\ .\quad . & .\quad . & .\quad . \\ .\quad . & .\quad . & .\quad . \\ .\quad . & .\quad . & .\quad . \\ .\quad . & .\quad . & .\quad . \\ 1\ldots.1 & 0\ldots.0 & .\quad . \\ 0\ldots.0 & 1\ldots.1 & 1\ldots.1 \end{pmatrix}$$

$p+1 \qquad k \qquad p+1$

and

$$\frac{store(\text{BESTFIT}(A_{p,k}))}{optstore(A_{p,k})} = \frac{2(p+1)^2+(p+2)k}{(p+1)^2+(p+1)(p+2)+2k} = \frac{2\frac{p+1}{p+2}+\frac{k}{p+1}}{\frac{p+1}{p+2}+1+\frac{2k}{(p+1)(p+2)}} \approx \frac{p+2}{2}$$

if k large compared with p. Thus, with p and k large enough, we have $p+2 \geqq N$, $2p+2+k \geqq N$ and $(p+2)/2 > c$. We conclude that BESTFIT is not a c-approximation algorithm for any $c \in \mathbf{R}$. □

Observe that the columns of $A_{p,k}$ are sorted by nonincreasing number of nonzero elements. BESTFIT has a rather bad performance because it compares the one column to be inserted with the (maybe many) columns already processed. It may decide to store the zero elements of column $A[\ , q]$ (A arbitrary) rather than to change not-stored zero elements of BESTFIT$(A[\ , 1:q-1])$ into stored zero elements in BESTFIT$(A[\ , q])$. This may happen even when $A[\ , q]$ has many zero elements and is identical to many other columns in the matrix A. Consider, for example, column $2p+3$ in the matrix $A_{p,k}$ as used in the proof of Proposition 2.2.

BESTFIT has a tendency to insert the last columns somewhere in the middle of the matrix obtained so far, which may lead to many stored zero elements that would not be stored in the optimum column permutation. Thus Algorithm 3 below in which the columns are ordered by decreasing number of zero elements, may perform better. As for identical columns we have the following lemma.

LEMMA 2.3. *Let A be an $m \times n$ $\{0,1\}$-matrix. Then there is an optimum column permutation B of A such that identical columns of B are consecutive in B.*

Proof. Let C be an optimum column permutation of A and suppose not all identical columns of C are consecutive. Thus there are $j_0, j_1, j_2 \in \mathbf{N}$ $(j_0 < j_1 < j_2)$ such that the columns j_0 and j_2 of C are identical and the columns j_0 and j_1 of C are not identical. Without loss of generality we can assume that the number k of stored elements of column j_0 is not greater than the number of stored elements of column j_2. If we delete column j_2 and insert it just beside column j_0 (thus obtaining a $\{0,1\}$-matrix C') then exactly k elements of column j_2 of C are stored in C'. Then:

$$store(C') = k + store(C[\ , 1:j_2-1]\ \textbf{concat}\ C[\ , j_2+1:n]) \leqq store(C).$$

Because C was assumed to be an optimum column permutation of A, we have $store(C') = store(C)$. Thus we can permute the columns of C in such a way that the

number of stored elements will not increase and identical columns will be consecutive. This gives us another optimum column permutation of A. □

DEFINITION 2.6. Let A be an $m \times n$ matrix. We say that A has a *row partition* $K^r = (0 = k_0^r < k_1^r < \cdots < k_p^r = m)$ and a *column partition* $K^c = (0 = k_0^c < k_1^c < \cdots < k_q^c = n)$ if A can be partitioned into $(A_{ij})_{1 \leqq i \leqq p, 1 \leqq j \leqq q}$ such that for all i, j $(1 \leqq i \leqq p, 1 \leqq j \leqq q)$ $A_{ij} = A[k_{i-1}^r + 1 : k_i^r, k_{j-1}^c + 1 : k_j^c]$. In this case $rowstrip(A, i, K^r) = A[k_{i-1}^r + 1 : k_i^r, \ ]$ and $colstrip(A, j, K^c) = A[\ , k_{j-1}^c + 1 : k_j^c]$.

DEFINITION 2.7.

(i) An $m \times n$ matrix A is a *block diagonal matrix* with k blocks if it can be partitioned into $A = (A_{ij})_{1 \leqq i \leqq p, 1 \leqq j \leqq q}$ such that $\min(p, q) = k$ and $A_{ij} = 0$ for all $i \neq j$ $(1 \leqq i \leqq p, 1 \leqq j \leqq q)$. *Block* i of A is the matrix A_{ii} $(1 \leqq i \leqq k)$.

(ii) An $m \times n$ matrix $A = (\alpha_{ij})$ is a *permuted block diagonal matrix* with k blocks if there are $m \times m$ and $n \times n$ permutation matrices P and Q such that PAQ can be partitioned into a block diagonal matrix with k blocks.

Without proof we state:

LEMMA 2.4. *Let A be an $m \times n$ $\{0, 1\}$-matrix with row partition $K^r = (k_0^r, \cdots, k_p^r)$. Then*:

$$store(A) = \sum_{i=1}^{p} store(rowstrip(A, i, K^r)).$$

If A is a block diagonal matrix with row partition K^r and column partition $K^c = (0 = k_0^c, \cdots, k_q^c)$, then:

(i) *for each i $(1 \leqq i \leqq \min(p, q))$*:

$$store\ (rowstrip(A, i, K^r)) = store(colstrip(A, i, K^c)) = store(A_{ii}),$$

(ii)
$$\begin{aligned} optstore(A) &= \sum_{i=1}^{p} optstore(rowstrip(A, i, K^r)) \\ &= \sum_{i=1}^{q} optstore(colstrip(A, i, K^c)) \\ &= \sum_{i=1}^{\min(p,q)} optstore(A_{ii}). \end{aligned}$$

ALGORITHM 3.

proc BESTFITDECR = **(matrix** A**)matrix:**

(Let B be a column permutation of A such that the number of zero elements per column in B is nonincreasing. Moreover, identical columns of A are consecutive in B.

Then perform BESTFIT (B) with the modification that identical columns are inserted simultaneously and are kept in consecutive order in the result of BESTFIT (B)

)

BESTFITDECR processes the matrices $A_{p,k}$ $(p \geqq 2)$ of the proof of Proposition 2.2 very well:

$$\frac{store(\text{BESTFITDECR}\,(A_{p,k}))}{optstore(A_{p,k})} = \frac{store(A_{p,k})}{optstore(A_{p,k})} = \frac{store(A_{p,k})}{store(A_{p,k})} = 1.$$

BESTFIT works rather well for a block diagonal matrix A: the columns of each columnstrip of A remain consecutive in BESTFIT (A). However, BESTFIT can have

a bad performance for permuted block diagonal matrices, which will not necessarily be improved by BESFITDECR.

PROPOSITION 2.5. BESTFITDECR *is not a c-approximation algorithm for any* $c \geqq 1$.

Proof. We have to prove that for each $N \in \mathbf{N}$ and each $c \in \mathbf{R}$ $(c \geqq 1)$ there is an $m \times n$ $\{0, 1\}$-matrix A $(m, n \geqq N)$ such that

$$\frac{store(\text{BESTFITDECR}(A))}{optstore(A)} > c.$$

Let $N \in \mathbf{N}$, $c \in \mathbf{R}$ $(c \geqq 1)$. Let A_k be a block diagonal matrix with $2k$ blocks A_{ii} with

$$A_{ii} = M_{I_1} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \quad (1 \leqq i \leqq 2k).$$

Let $B_k = (\beta_{ij})$ be the $6k \times 7k$ $\{0, 1\}$-matrix with $B_k[\ , 1:6k] = A_k$ and for each i $(1 \leqq i \leqq k)$ $\beta_{3(i-1)+1,6k+i} = \beta_{3k+3(i-1)+1,6k+i} = 1$ (see Fig. 3). B_k is a permuted block diagonal

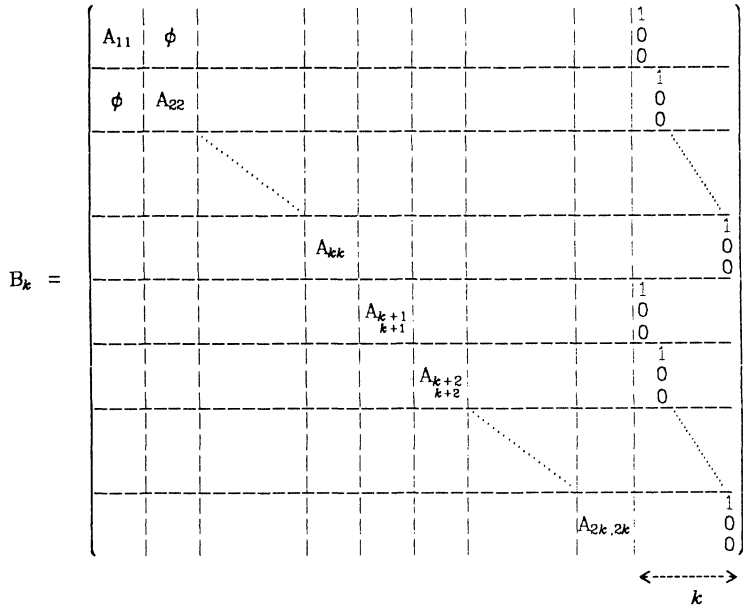


FIG. 3. *Matrices used in the proof of Proposition* 2.5.

matrix with k blocks. All columns of B_k are different and each column contains two nonzero elements. Let $K^c = (0, 3, 6, \cdots, 6k, 7k)$. BESTFITDECR returns the columns of B_k in the ordering:

$$colstrip(B_k, 2k, K^c), \quad B_k[\ , 7k], \quad colstrip(B_k, 2k-1, K^c), \quad B_k[\ , 7k-1], \quad \cdots,$$

$$B_k[\ , 6k+2], \quad colstrip(B_k, k+1, K^c), \quad B_k[\ , 6k+1], \quad colstrip(B_k, k, K^c),$$

$$colstrip(B_k, k-1, K^c), \quad \cdots, \quad colstrip(B_k, 1, K^c).$$

Thus

$$\begin{aligned} &store(\text{BESTFITDECR}(B_k)) \\ &\quad = \sum_{i=1}^{k} store(A_{ii}) + \sum_{i=k+1}^{2k} (store(A_{ii})+1) + ones(B_k[\ , 6k+1:7k]) \\ &\qquad\qquad + \sum_{i=1}^{k} (3(k-1)+i-1) \\ &\quad = \frac{k}{2}(7k+27). \end{aligned}$$

On the other hand, $optstore(B_k) = 2k + \sum_{i=1}^{2k} opstore(A_{ii}) = 2k + 14k = 16k$. With k large enough we have

$$\frac{store(\text{BESTFITDECR}(B_k))}{optstore(B_k)} = \frac{7k^2+27k}{32k} > c.$$

Hence, BESTFITDECR is not a c-approximation algorithm for any $c \geqq 1$. □

DEFINITION 2.8. Let A be an $m \times n$ $\{0, 1\}$-matrix and u a $\{0, 1\}$-sequence of length m. An $m \times (n+1)$ $\{0, 1\}$-matrix B is an *insertion matrix for u in A* if for some $y \in \mathbf{N}$ $(0 \leqq y \leqq n)$

$$B = A[\ , 1:y] \textbf{ concat } u \textbf{ concat } A[\ , y+1:n].$$

DEFINITION 2.9. Let X be an algorithm that takes $\{0, 1\}$-matrices as input and that returns a column permutation of its input. X is an *on-line column insertion algorithm* if for each $m \times (n+1)$ $\{0, 1\}$-matrix A, $X(A)$ is an insertion matrix for $A[\ , n+1]$ in $X(A[\ , 1:n])$.

PROPOSITION 2.6. BESTFITDECR *is not an on-line column insertion algorithm.*

Proof. Let

$$A = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}, \quad \text{BESTFITDECR}(A) = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix},$$

$$\text{BESTFITDECR}(A[\ , 1:3]) = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

and therefore, BESTFITDECR(A) is not an insertion matrix for $A[\ , 4]$ into BESTFITDECR$(A[\ , 1:3])$. □

We could have defined the notion of an on-line column insertion algorithm in another way.

DEFINITION 2.10. Let X be an algorithm that takes $\{0, 1\}$-matrices as input and that returns a column permutation of its input. X has *property P* if for every $m \times n$

$\{0, 1\}$-matrix A and every $k \in \mathbf{N}$ $(1 \leqq k \leqq n)$, $X(A)$ and $X(A[\ , 1:k])$ satisfy (5):

(5) let column i $(1 \leqq i \leqq n)$ of A be column p_i of $X(A)$; let column i $(1 \leqq i \leqq k)$ of $A[\ , 1:k]$ be column q_i of $X(A[\ , 1:k])$; for all i and j $(1 \leqq i, j \leqq k)$ we have

$$p_i < p_j \quad \text{if and only if} \quad q_i < q_j.$$

LEMMA 2.7. *X is an on-line column insertion algorithm if and only if X has property P.*

Proof.

$\Leftarrow$: Suppose X has property P. Let A be an $m \times n$ $\{0, 1\}$-matrix and $k = n - 1$. Let $(p_i)_{1 \leqq i \leqq n}$ and $(q_i)_{1 \leqq i \leqq n-1}$ as in (5) and $y = p_n$. Then for all i with $q_i \leqq y - 1$ we have $p_i = q_i$ and for all i with $q_i \geqq y$, $p_i = q_i + 1$. Thus $X(A)$ is an insertion matrix for $A[\ , n]$ in $X(A[\ , 1:n-1])$. This holds for every matrix A and therefore, X is an on-line column insertion algorithm.

$\Rightarrow$: Suppose X is an on-line column insertion algorithm. Let A be an $m \times n$ $\{0, 1\}$-matrix and $k \in \mathbf{N}$ $(1 \leqq k \leqq n)$. Consider $A[\ , 1:h]$ with $k \leqq h \leqq n$. Let column i $(1 \leqq i \leqq h)$ of $A[\ , 1:h]$ be column $p_{h,i}$ of $X(A[\ , 1:h])$. $X(A[\ , 1:h+1])$ is an insertion matrix for $A[\ , h+1]$ in $X(A[\ , 1:h])$, thus:

$$\text{for all } i, j\ (1 \leqq i, j \leqq h)\ p_{h,i} < p_{h,j} \text{ if and only if } p_{h+1,i} < p_{h+1,j}.$$

With induction it is easy to prove that

$$\text{for all } i, j\ (1 \leqq i, j \leqq k)\ p_{k,i} < p_{k,j} \text{ if and only if } p_{n,i} < p_{n,j}.$$

Hence, X has property P. □

Using this lemma, one can for every $m \times n$ $\{0, 1\}$-matrix A determine $X(A[\ , 1:k])$ from $X(A)$ for every $k \leqq n$, since the relative ordering of columns of $A[\ , 1:k]$ as they appear in $X(A)$ is inherited.

COROLLARY 2.8. *Let $A = (\alpha_{ij})$ be an $m \times n$ $\{0, 1\}$-matrix and X an on-line column insertion algorithm. If for some $k \leqq n$ and i_0, j_0 $(1 \leqq i_0 \leqq m, 1 \leqq j_0 \leqq k)$ α_{i_0,j_0} (considered as an element) of $A[\ , 1:k]$ is stored in $X(A[\ , 1:k])$, then α_{i_0,j_0} (considered as an element) of A is stored in $X(A)$.*

3. A negative result. Now the question arises whether there is some on-line column insertion algorithm that is a polynomial time c-approximation algorithm for the CONSECUTIVE ONES MATRIX AUGMENTATION problem. In Theorem 3.2 we will answer this question negatively. But Corollary 3.3 states more. To obtain a negative answer, it will be sufficient to show that for each on-line column insertion algorithm X and for each $c \geqq 1$ and each $N \in \mathbf{N}$ there is an $m \times n$ matrix $A_{X,c,N}$ $(m, n \geqq N)$ such that

$$\frac{store(X(A_{X,c,N}))}{optstore(A_{X,c,N})} > c$$

which proves that, in fact no on-line column insertion algorithm can be a c-approximation algorithm (for some fixed c) at all. Theorem 3.2 states that $A_{X,c,N}$ can be chosen to be of arbitrary size (for fixed X and c) and Corollary 3.3 states that for fixed c and N $A_{X,c,N}$ can be chosen such that the following conditions are satisfied:

(i) the number of rows and columns of $A_{X,c,N}$ does not depend on the on-line column insertion algorithm X,

(ii) let X and X' be two on-line column insertion algorithms. Then $A_{X,c,N}$ and $A_{X',c,N}$ are equal possibly except for their last columns.

In order to prove this we will use a block diagonal matrix A of k $m \times m$ blocks. The number of elements in a block is bounded from above by m^2 and in a rowstrip by km^2. We will use blocks that all have an *optstore* value bounded by $3m$. A is chosen such that for each of its column permutations B it is possible that many elements of a rowstrip of B will be stored if one extra column is inserted in B.

THEOREM 3.1. *Let $N \in \mathbf{N}$, $c \in \mathbf{R}$ $(c \geqq 1)$. Then there is a clean square matrix A with $n \geqq N$ rows such that for each column permutation B of A we have:*

*if $store(B)/optstore(A) \leqq c$ then there is a $\{0, 1\}$-sequence u_B of length n such that for each matrix C that is an insertion matrix for u_B in B, $store(C)/optstore(A$ **concat** $u_B) > c$.*

Proof. Let $N \in \mathbf{N}$ and $c \in \mathbf{R}$ $(c \geqq 1)$. Let $f: \mathbf{N} \to \mathbf{N}$ be any function such that
(i) $\lim_{k \to \infty} f(k) = \infty$,
(ii) there are a $c' > 0$ and a $K' \in \mathbf{N}$ such that for all $k \geqq K'$, $f(k)/k < \frac{1}{3} - c'$.
For example: $f(k) = \lceil k/4 \rceil$, $f(k) = \lceil \log_2 k \rceil$, etc.
Choose $k' \in \mathbf{N}$ $(k' \geqq 3)$ such that for all $k \geqq k'$ we have

$$f(k) > c \quad \text{and} \quad \frac{k - 3f(k)}{3k}(1 + 2f(k)) > c. \tag{6}$$

Such a k' exists. Let $k \geqq k'$. Choose $m' \in \mathbf{N}$ $(m' \geqq 3)$ such that for all $m \geqq m'$ we have

$$mk \geqq N \quad \text{and} \quad \frac{k(3m-2) + m + m^2 f(k)}{k(3m-2) + m^2 - m + 2} > c.$$

Let $m \geqq m'$. Thus:

$$\begin{gathered} mk \geqq N, \quad m \geqq 3, \quad k \geqq 3, \quad \frac{k(3m-2) + m + m^2 f(k)}{k(3m-2) + m^2 - m + 2} > c, \\ \frac{1 + 2f(k)}{3} > c \quad \text{and} \quad \frac{k - 3f(k)}{3k}(1 + 2f(k)) > c. \end{gathered} \tag{7}$$

Observe that the fifth inequality of (7) follows from the second inequality of (6).

Let A be a block diagonal matrix with k blocks $M_1, \cdots, M_k$ and

$$M_i = M_{I_{m-2}} \quad (1 \leqq i \leqq k). \tag{8}$$

Let $K = (0, m, 2m, \cdots, mk)$ be the corresponding row and column partition. Then we have:

$$optstore(M_i) = 3m - 2 \ (1 \leqq i \leqq k), \qquad optstore(A) = (3m-2)k. \tag{9}$$

Let B be any column permutation of A with column j of A column p_j of B $(1 \leqq j \geqq mk)$.

Let

$$\begin{aligned} pmin(i) &= \min \{p_j : (i-1)m \leqq j \leqq im\}, \qquad (1 \leqq i \leqq k) \quad \text{and} \\ pmax(i) &= \max \{p_j : (i-1)m \leqq j \leqq im\}, \qquad (1 \leqq i \leqq k). \end{aligned} \tag{10}$$

Claim.

$$store(rowstrip(B, i, K)) \geqq m + 2(pmax(i) - pmin(i)) \quad \text{for all } 1 \leqq i \leqq k. \tag{11}$$

Proof of Claim. Consider column t in $rowstrip(B, i, K)$ with $pmin(i) \leqq t \leqq pmax(i)$. It is either a column of M_i or it is a zero column. When it is a zero column, we have

$pmin(i) < t < pmax(i)$ and at least two of its zero elements are stored in $rowstrip(B, i, K)$. There are exactly $pmax(i) - pmin(i) - m + 1$ zero columns in $rowstrip(B, i, K)$ between the columns $pmin(i)$ and $pmax(i)$. Hence

$$store(rowstrip(B, i, K)) \geqq optstore(M_i) + 2(pmax(i) - pmin(i) - m + 1).$$

With (9) this proves the claim.

We are interested in the values of $pmin(i) - pmax(j)$.

Let $q_0 = \min\{pmax(i): 1 \leqq i \leqq k\}$, $q_1 = \max\{pmin(i): 1 \leqq i \leqq k\}$.

Thus exactly one columnstrip of A has all its columns at the left of column $q_0 + 1$ in B and exactly one columnstrip of A has all its columns at the right of column $q_1 - 1$ in B. There are three cases.

Case 1. $q_1 \leqq q_0 - mf(k)$.

Claim 1. With m and k satisfying (7) and $q_1 \leqq q_0 - mf(k)$ we have

$$\frac{store(B)}{optstore(A)} > c.$$

Proof of Claim 1. $q_1 \leqq q_0 - mf(k)$. Thus:

$$\text{for all } i \quad pmax(i) \geqq pmin(i) + mf(k) \qquad (i \leqq i \leqq k).$$

Using (11) this results in $store(rowstrip(B, i, K)) \geqq m + 2mf(k)$ so that (with (7) and (9)):

$$\frac{store(B)}{optstore(A)} \geqq \frac{mk + 2mkf(k)}{(3m-2)k} \geqq \frac{m + 2mf(k)}{3m} = \frac{1 + 2f(k)}{3} > c.$$

Hence Claim 1 is proved and Case 1 satisfies the theorem.

Case 2. $|q_1 - q_0| < mf(k)$.

Claim 2. With m and k satisfying (7) and $|q_1 - q_0| < mf(k)$ we have

$$\frac{store(B)}{optstore(A)} > c.$$

Proof of Claim 2. First assume $mf(k) < \min(q_0, q_1) < \max(q_0, q_1) < mk - mf(k)$. We divide B into five parts, as shown in Fig. 4.

$$\begin{aligned}
R_1 &= B[\ , & 1 & : \min(q_0, q_1) - mf(k) - 1], \\
R_2 &= B[\ , & \min(q_0, q_1) - mf(k) & : \min(q_0, q_1) - 1\ ], \\
R_3 &= B[\ , & \min(q_0, q_1) & : \max(q_0, q_1)\ ], \\
R_4 &= B[\ , & \max(q_0, q_1) + 1 & : \max(q_0, q_1) + mf(k)\ ], \\
R_5 &= B[\ , & \max(q_0, q_1) + mf(k) + 1 & : mk\ ].
\end{aligned}$$

R_2, R_3 and R_4 together contain at most $3mf(k)$ columns. B contains mk columns so that R_1 and R_5 together at least $mk - 3mf(k)$ columns.

Let

$$I_{MIN} = \{i: pmin(i) < \min(q_0, q_1) - mf(k)\} \quad \text{and}$$

$$I_{MAX} = \{i: pmax(i) > \max(q_0, q_1) + mf(k)\}.$$

Then $|I_{MIN} \cup I_{MAX}| \geqq k - 3f(k)$. For each $i \in I_{MIN} \cup I_{MAX}$: $pmax(i) - pmin(i) \geqq mf(k)$.

With (11) this gives the result:

$$\text{for all } i \in I_{MIN} \cup I_{MAX}: \qquad store(rowstrip(B, i, K)) \geqq m + 2mf(k).$$

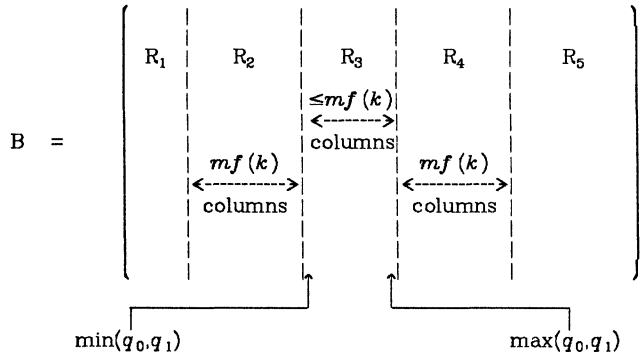


FIG. 4

Hence:

$$store(B) \geqq \sum_{i \in I_{MIN} \cup I_{MAX}} store(rowstrip(B, i, K))$$

$$\geqq |I_{MIN} \cup I_{MAX}| \cdot (m + 2mf(k)) \geqq (k - 3f(k)) \cdot (m + 2mf(k)).$$

With (7):

$$\frac{store(B)}{optstore(A)} \geqq \frac{(k-3f(k)) \cdot (m+2mf(k))}{(3m-2)k} \geqq \frac{(k-3f(k)) \cdot (1+2f(k))}{3k} > c.$$

Now we have proved Claim 2 for the general case that $mf(k) < \min(q_0, q_1) < \max(q_0, q_1) < mk - mf(k)$.

Next assume $\min(q_0, q_1) \leqq mf(k)$ or $\max(q_0, q_1) \geqq mk - mf(k)$. The two special cases ($\min(q_0, q_1) \leqq mf(k)$ and $\max(q_0, q_1) \geqq mk - mf(k)$) are similar; thus we will only deal with $\min(q_0, q_1) \leqq mf(k)$. Now we divide B in 4 parts:

$$R_2 = B[\ ,\ 1\ :\ \min(q_0, q_1) - 1\ ],$$

$$R_3 = B[\ ,\ \min(q_0, q_1)\ :\ \max(q_0, q_1)\ ],$$

$$R_4 = B[\ ,\ \max(q_0, q_1) + 1\ :\ \max(q_0, q_1) + mf(k)\ ] \quad \text{and}$$

$$R_5 = B[\ ,\max(q_0, q_1) + mf(k) + 1\ :\ mk\ ].$$

R_2, R_3 and R_4 together contain at most $3mf(k)$ columns, and therefore R_5 contains at least $mk - 3mf(k)$ columns and there are at least $k - 3f(k)$ columnstrips of A with a column in R_5. Let

$$I_{MAX} = \{i: pmax(i) > \max(q_0, q_1) + mf(k)\}.$$

Then $|I_{MAX}| \geqq k - 3f(k)$. For each $i \in I_{MAX}$: $pmax(i) - pmin(i) > mf(k)$. With (11):

$$store(B) \geqq \sum_{i \in I_{MAX}} store(rowstrip(B, i, K)) \geqq (k - 3f(k)) \cdot (m + 2mf(k))$$

and with (7): $store(B)/optstore(A) > c$.

Now Claim 2 is proven, and hence Case 2 satisfies the theorem.

Case 3. $q_1 \geqq q_0 + mf(k)$.

Claim 3. With m and k satisfying (7) and $q_1 \geqq q_0 + mf(k)$ there is a $\{0, 1\}$-sequence u_B of length mk such that for each matrix C that is an insertion matrix for u_B into B, we have $store(C)/optstore(A \textbf{ concat } u_B) > c$.

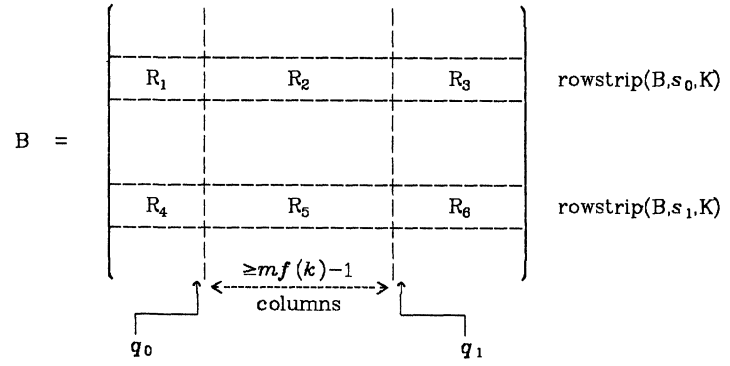


FIG. 5

Proof of Claim 3. *Let* $q_1 - q_0 \geqq mf(k)$. With the definition of q_0 and q_1 there are s_0 and s_1 $(1 \leqq s_0 \leqq k, 1 \leqq s_2 \leqq k)$ such that $q_0 = pmax(s_0)$ and $q_1 = pmin(s_1)$. Then we have: $s_0 \neq s_1$ and $pmin(s_1) - pmax(s_0) \geqq mf(k)$. Let us divide B as in Fig. 5.

Because of (10) and the choice of s_0 and s_1, the submatrices R_2, R_3, R_4 and R_5 are all zero matrices. Let u_B be a $\{0, 1\}$-sequence of length mk defined by:

$$u_B[j] = \begin{cases} 1 & \text{if } (s_0 - 1)m < j \leqq s_0 \cdot m, \\ 1 & \text{if } (s_1 - 1)m < j \leqq s_1 \cdot m, \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

(13) A **concat** u_B is a permuted block diagonal matrix with $k-1$ blocks.

$$\begin{aligned} optstore(A \textbf{ concat } u_B) &\leqq \sum_{\substack{i=1 \\ i \neq s_0, s_1}} optstore(rowstrip(A, i, K)) + 2\left(m + 1 + \sum_{i=3}^{m+1} i\right) \\ &= (k-2)(3m-2) + m^2 + 5m - 2. \end{aligned} \tag{14}$$

Let C be an insertion matrix for u_B in B. Thus, there is a $y \in \mathbf{N}$ $(0 \leqq y \leqq mk)$ such that

$$B[\ , 1:y] = C[\ , 1:y],$$

$$u_B = C[\ , y+1],$$

$$B[\ , y+1:mk] = C[\ , y+2:mk+1].$$

If $y \leqq q_0$ then all zero elements of R_5 are stored in C though they are not stored in B. R_5 contains at least $m^2 f(k) - m$ zero elements.

If $y \geqq q_1 - 1$ then all zero elements of R_2 are stored in C though they are not stored in B. R_2 contains at least $m^2 f(k) - m$ zero elements.

If $q_0 < y \leqq q_1$, then the same number of zero elements will be stored in C, but now they are distributed over R_2 and R_5. None of these zero elements are stored in B.

Thus:

$$\begin{aligned} store\,(C) &\geqq optstore(B) + ones(u_B) + |\{\text{elements of } R_2 \text{ and } R_5 \text{ stored in } C\}| \\ &\geqq k(3m-2) + 2m + m^2 f(k) - m = k(3m-2) + m^2 f(k) + m. \end{aligned}$$

With (14) and (7), this gives:

$$\frac{store(C)}{optstore(A \textbf{ concat } u_B)} > \frac{k(3m-2) + m + m^2 f(k)}{k(3m-2) + m^2 - m + 2} > c.$$

This ends the proof of Claim 3 and the proof of Theorem 3.1. □

As a direct consequence we have:

THEOREM 3.2. *Let X be an algorithm that takes $\{0, 1\}$-matrices as input and that returns a column permutation of its input. Suppose there is a $c \in \mathbf{R}$ such that X is a c-approximation algorithm. Then X is not an on-line column insertion algorithm.*

However, Theorem 3.1 states more about c-approximation algorithms.

COROLLARY 3.3. *Let X satisfy the conditions of Theorem 3.2. Then for all $N \in \mathbf{N}$ and for every algorithm Y that takes $\{0, 1\}$-matrices as input and that returns a column permutation of its input, there is an $m \times n$ matrix B $(m, n \geqq N)$ such that $X(B)$ is not an insertion matrix for $B[\ , n]$ in $Y(B[\ , 1:n-1])$.*

COROLLARY 3.4. *Let $N \in \mathbf{N}$, $c \in \mathbf{R}$ $(c \geqq 1)$. Then there are $m, n \geqq N$ and an $m \times n$ matrix A such that for every two on-line column insertion algorithms X and X' there are $m \times (n+1)$ matrices B and C with*

$$\frac{store(X(B))}{optstore(B)} > c, \quad \frac{store(X'(C))}{optstore(C)} > c \quad and \quad B[\ , 1:n] = C[\ , 1:n] = A.$$

Theorem 3.2 is the special case of Corollary 3.3 with X replaced for Y. Observe that Corollary 3.3 holds even if Y finds the optimum column permutation of its input.

It actually is the insertion of the last column of its input that prevents an on-line column insertion algorithm to be a c-approximation algorithm for some $c \in \mathbf{R}$. If we design an approximation algorithm that first determines some column permutation D of all but the last column u of its input (such that D is independent of u) and then inserts u in D, then this algorithm is not a c-approximation algorithm for any $c \in \mathbf{R}$.

With Corollary 3.3 we have described a large class of approximation algorithms that are not c-approximation algorithms. This class contains, among others, the on-line column insertion algorithms.

4. Extensions to other classes of approximation algorithms. In the previous section we saw that many approximation algorithms can provide bad approximations to the optimal storage (in a rowmat data structure) of a column permutation of a matrix. In particular this bad result is obtained for some block diagonal matrices. Knowing this, one can try to design more sophisticated approximation algorithms. In this section we will extend Theorem 3.1 and, consequently, Theorem 3.2 and the Corollaries 3.3 and 3.4, and describe larger classes of approximation algorithms that still do not contain a c-approximation algorithm for any $c \in \mathbf{R}$.

4.1. Preprocessing to block diagonal form. The class of matrices used in the proof of Theorem 3.1 merely consists of block diagonal and permuted block diagonal matrices (see (8) and (13)). For each block diagonal matrix there is an optimum column permutation that is block diagonal too. Moreover, as we have seen before (Algorithm 2), there is an on-line column insertion algorithm that preserves this block diagonal structure if applied to a block diagonal matrix. Thus it seems reasonable to permute the rows and columns of a permuted block diagonal matrix to block diagonal form before applying any on-line column insertion algorithm. Unfortunately this does not give a c-approximation algorithm for any $c \in \mathbf{R}$. In the proof of Theorem 3.1 we could have chosen a slightly different A': A' has also nonzero elements in the positions $(m+1, m)$, $(2m+1, 2m), \cdots, ((k-1)m+1, (k-1)m)$ and $(1, km)$. With this change the blocks of A have been made loosely connected in A'. The column u_B in the proof would then make a stronger connection between two loosely connected blocks of A' that are far away from each other in the column permutation B. Preprocessing A' into block diagonal form does not change A' because it is already in this form (one block).

To prove negative results concerning more sophisticated preprocessors, we introduce the concept of a separator of a matrix.

DEFINITION 4.1. Let $p \in \mathbf{N}$ and A an $m \times n$ $\{0, 1\}$-matrix with $p \leqq n$.

(i) A set of p columns of A is a *separator* of A if the matrix A with the columns deleted, is a permuted block diagonal matrix with at least two blocks.

(ii) Let S be a smallest separator of A. A is in *connected order* if the deletion of the columns of S from A results in a block diagonal matrix.

A matrix A with a small separator is almost a permuted block diagonal matrix. If, in addition to this, A is in connected order, A can be considered almost block diagonal. The idea is that preprocessing an input matrix into connected order, would improve the worst-case behavior of on-line column insertion algorithms. However, the next theorem states that if an on-line column insertion algorithm is combined with a preprocessing algorithm that permutes matrices in connected order, this will not give rise to a c-approximation algorithm for any $c \in \mathbf{R}$.

THEOREM 4.1. *Let $N \in \mathbf{N}$, ε, $c \in \mathbf{R}$ $(c \geqq 1, \varepsilon > 0)$. Then there exist an $n \in \mathbf{N}$ and a square $\{0, 1\}$-matrix A such that for each column permutation B of A:*

*if $store(B)/optstore(A) \leqq c$ then there is a $\{0, 1\}$-sequence u_B of length $n^{1+\varepsilon}$ such that $store(C)/optstore(A$ **concat** $u_B) > c$ for any matrix C that is an insertion matrix for u_B in B. A and A **concat** u_B satisfy the following conditions:*

(i) *they are clean;*
(ii) *they have $n^{1+\varepsilon} \geqq N$ rows;*
(iii) *they are in connected order;*
(iv) *they do not contain a separator of size $< n^{1-\varepsilon}$.*

Proof. Because the proof of this theorem is similar to the proof of Theorem 3.1 and uses the same kind of arguments, we will only give an outline. Let $f: N \to \mathbf{N}$ be a function with $f(n) = \Omega(n)$ and there are $\delta > 0$ and $K \in \mathbf{N}$ such that for all $k \geqq \mathbf{K}$ $(3f(k) - 1)/3k - \frac{1}{3} > \delta$. Let M, U and A be as in Fig. 6. Then we have:

(i) The smallest separator of $A_{n,k,d}$ contains $2d - 2$ columns,

(ii) $nd \leqq optstore(M_i) \leqq (d-1)n + (n-d+1)d$,

(iii) $kd(\frac{1}{2}(d-1) + n) \leqq optstore(A_{n,k,d})$ and $optstore(A_{n,k,d}) \leqq k(3nd - \frac{1}{2}d^2 + \frac{1}{2}d - 2n) - n(d-1) - \frac{1}{2}d(d-1)$.

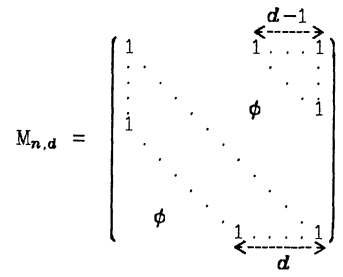
Let B be a column permutation of $A_{n,k,d}$. $pmax(i)$ and $pmin(i)$, $1 \leqq i \leqq k$, are defined as in (10). Without proof we state:

Claim. $store(rowstrip\ (B, i, K)) \geqq d(pmax(i) - pmin(i) + 1)$ for all i $(1 \leqq i \leqq k)$.

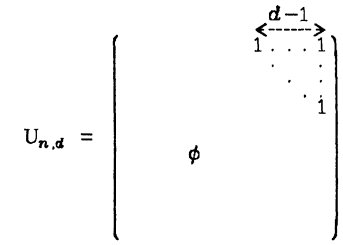
Let $q_0 = \max\{pmin(i): 2 \leqq i \leqq k\}$, $q_1 = \min\{pmax(i): 2 \leqq i \leqq k\}$. The rest of the proof is quite the same as the proof of Theorem 3.1, except that it requires more calculations. We only show where the connected order plays a role. $A_{n,k,d}$ does not have a separator of size less than $2d - 2$. Deletion of the last $d - 1$ columns of the first and last columnstrip of $A_{n,k,d}$, results in a block diagonal matrix with two blocks. Moreover, the sequence u_B that is defined will not have nonzero elements in the first n positions, because $pmin(1)$ and $pmax(1)$ do not play any role in the value of q_0 and q_1. Therefore, deletion of the same $2d - 2$ columns of $A_{n,k,d}$ **concat** u_B once again results in a block diagonal matrix with two blocks. □

Remark. Theorem 4.1 even holds if we put the additional restriction on A (and A **concat** u_B) that the columns of a smallest separator are consecutive in A (and A **concat** u_B).

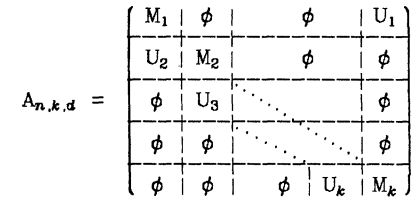
Thus, preprocessing a matrix into connected order does not help us to find c-approximation if we use an on-line column insertion algorithm. Of course, it is impossible to prove negative results for all combinations of preprocessing and on-line



an $n \times n$ $\{0,1\}$-matrix in which each row and column has exactly d non-zero elements,



an $n \times n$ $\{0,1\}$-matrix,



with $M_i = M_{n,d}$ $(1 \leq i \leq k)$

$U_i = U_{n,d}$ $(1 \leq i \leq k)$.

FIG. 6

column insertion algorithms: one of these combinations gives for each matrix the optimum column permutation. But each preprocessing algorithm that does not change the column number of u_B if applied to A **concat** u_B (see (12)), does not improve the performance of any on-line column insertion algorithm.

4.2. Mixers and on-line column insertion algorithms. Another kind of more sophisticated approximation algorithms can be obtained by incorporating a "small mixer" in an on-line column insertion algorithm. On-line column insertion algorithms often can be formulated in such a way that the columns are processed one by one. The columns that are processed do not loose their relative order with the insertion of a next column. If we incorporate a "small mixer", we allow that the columns already processed are permuted slightly just before column i will be inserted.

DEFINITION 4.2. Let A be an $m \times n$ $\{0, 1\}$-matrix, u an $\{0, 1\}$-sequence of length m and $s: \mathbf{N} \to \mathbf{N}$ a function with $s(k) \leqq k$ for all $k \in \mathbf{N}$. Let B a column permutation of A **concat** u and column i of A **concat** u is column p_i of B $(1 \leqq i \leqq n+1)$.

B is an *s-mix*1 *insertion matrix for* u in A if there are $n - s(n)$ numbers $i_1 < i_2 < \cdots < i_{n-s(n)} < n$ such that

(15) $\quad i_j < i_k$ if and only if $p_{i_j} < p_{i_k}$ $(1 \leqq j \leqq n - s(n), 1 \leqq k \leqq n - s(n))$.

B is an *s-mix*2 *insertion matrix for* u in A if for all j $(1 \leqq j \leqq n)$ we have:

$$\text{if } p_j < p_{n+1} \text{ then } |p_j - j| \leqq s(n) \text{ and if } p_j > p_{n+1} \text{ then } |p_j - 1 - j| \leqq s(n).$$

DEFINITION 4.3. Let X be an algorithm that takes $\{0, 1\}$-matrices as input and that returns a column permutation of its input. Let $s: \mathbf{N} \to \mathbf{N}$ be a function with $s(k) \leqq k$

for all $k \in \mathbf{N}$. X is an *on-line column s-mix*1 (resp. *s-mix*2) *insertion algorithm* if for each $m \times n$ $\{0, 1\}$-matrix A, $X(A)$ is an s-mix1 (resp. s-mix2) insertion matrix for $A[\ , n]$ in $X(A[\ , 1:n-1])$.

In an on-line column s-mix1 insertion algorithm at most $s(n-1)$ columns may be deleted from $X(A[\ , 1:n-1])$ and inserted somewhere else in this matrix before inserting column n. In particular, an s-mix1 insertion algorithm allows that the first column of $X(A[\ , 1:n-1])$ will be the last column of $X(A)$, in case $s(n-1) \geqq 1$. The way these $s(n-1)$ columns are mixed and once again inserted, will have to depend on column n, otherwise Corollary 3.3 states that we do not have a better approximation algorithm.

In an on-line column s-mix2 insertion algorithm all columns of $X(A[\ , 1:n-1])$ are allowed to be permuted, but the new column number will differ at most $s(n-1)$ from the old column number. With $s(n-1) < n-2$, this means that the first column of $X(A[\ , 1:n-1])$ will never be the last column of $X(A)$. For the same reason as above we are only interested in mix2 algorithms that depend on column n of A. Unfortunately, if $s(n)$ is not large enough, no on-line column s-mix1 or s-mix2 insertion algorithm is a c-approximation algorithm.

THEOREM 4.2. *Let* $N \in \mathbf{N}$, ε, $c \in \mathbf{R}$ $(c \geqq 1, \varepsilon > 0)$. *Then there are an* $n \in \mathbf{N}$ *and a clean square* $\{0, 1\}$*-matrix* A *with* $n^{1+\varepsilon} \geqq N$ *rows such that for each column permutation* B *of* A *we have*:

> *If* $store(B)/optstore(A) \leqq c$ *then there is a* $\{0, 1\}$*-sequence* u_B *of length* $n^{1+\varepsilon}$ *such that for every function* $s: \mathbf{N} \to \mathbf{N}$ *with* $s(k) \leqq k^{1-\varepsilon}$ $(k \geqq 2)$, $store(C)/optstore(A$ **concat** $u_B) > c$ *for every matrix* C *that is an s-mix*1 *insertion matrix for* u_B *in* B.

THEOREM 4.3. *Theorem* 4.2 *with "mix*1*" replaced by "mix*2*"*.

We only sketch the proofs of these theorems, because they are similar to the proof of Theorem 3.1.

Proof of Theorem 4.2. Use the matrix A given in (8), with M_i an $n \times n$ matrix. Let B be a column permutation of A. Let $k = n^{\varepsilon}$.

We only have to look at the case $store(B)/optstore(A) \leqq c$ in which we have to provide u_B. Define u_B as in (12). Let B be divided as in Fig. 5.

Because $s(n^{1+\varepsilon}) < n$, there is at least one column of each columnstrip of A of which the column number is one of the $i_1, \cdots, i_{n^{1+\varepsilon}-s(n^{1+\varepsilon})}$ of (15). Let us delete all columns p_i of B of which i does not occur in $i_1, \cdots, i_{n^{1+\varepsilon}-s(n^{1+\varepsilon})}$, resulting in a matrix B'. Then there surely is a column of $colstrip(A, s_0, K)$ in the left part of B' and one of $colstrip(A, s_1, K)$ in the right part of B'. Inserting u_B in B' causes at least $n(nf(k) - s(n^{1+\varepsilon}) - 1)$ zero elements to be stored. Inserting the deleted columns can make things only worse. With suitable f and n large enough, ratio (4) is violated. □

Proof of Theorem 4.3. Let A, B and u_B as above in the proof of Theorem 4.2. After the mixing of B has been done, column $pmax(s_0)$ of B is at most $s(n^{1+\varepsilon})$ positions to the right and column $pmin(s_1)$ of B $s(n^{1+\varepsilon})$ to the left. Thus inserting u_B causes at least $n(nf(k) - 2s(n^{1+\varepsilon}) - 1)$ zero elements to be stored. With suitable f and n large enough, ratio (4) is violated. □

COROLLARY 4.4. *Let* X *be a c-approximation algorithm. Then* X *is neither an on-line column s-mix*1 *nor an on-line column s-mix*2 *insertion algorithm for any* $\varepsilon \in \mathbf{R}$ $(\varepsilon > 0)$ *and* $s: \mathbf{N} \to \mathbf{N}$ *with* $s(k) \leqq k^{1-\varepsilon}$.

COROLLARY 4.5. *Let* $c \in \mathbf{R}$ $(c \geqq 1)$. *Let* X *be a c-approximation algorithm and let* $N \in \mathbf{N}$ *such that* $store(X(A))/optstore(A) \leqq c$ *for all matrices* A *with at least* N *rows and* N *columns. Then for every algorithm* Y *that takes* $\{0, 1\}$*-matrices as input and*

that returns a column permutation of its input, there is an $m \times n$ *matrix* B $(m, n \geqq N)$ *such that* $X(B)$ *is neither an s-mix*1 *nor an s-mix*2 *insertion matrix for* $B[\ , n]$ *in* $Y(B[\ , 1:n-1])$ *for any* $\varepsilon \in \mathbf{R}$ $(\varepsilon > 0)$ *and* $s: \mathbf{N} \to \mathbf{N}$ *with* $s(k) \leqq k^{1-\varepsilon}$.

Acknowledgments. The author is grateful to J. van Leeuwen for the many discussions and would like to thank J. van Leeuwen, A. A. Schoone and D. P. Dobkin for their careful reading of the manuscript.

REFERENCES

[1] K. S. BOOTH, *PQ-tree algorithms*, Ph.D. thesis, Univ. California, Berkeley, 1975.

[2] K. S. BOOTH AND G. S. LUEKER, *Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms*, J. Comput. System. Sci., 13 (1976), pp. 335–379.

[3] G. VON FUCHS, J. R. ROY AND E. SCHREM, *Hypermatrix solution of large sets of symmetric positive-definite linear equations*, Comp. Meth. Appl. Mech. Eng., 1 (1972), pp. 197–216.

[4] D. R. FULKERSON AND O. A. GROSS, *Incidence matrices and interval graphs*, Pacif. J. Math., 15 (1965), pp. 835–855.

[5] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability, A Guide To The Theory of* NP-*completeness*, W. H. Freeman, San Francisco, 1979.

[6] J. A. GEORGE AND J. W. H. LIU, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[7] S. P. GHOSH, *Data Base Organization for Data Management*, Academic Press, New York, 1977.

[8] M. C. GOLUMBIC, *Algorithmic Graph Theory And Perfect Graphs*, Academic Press, New York, 1980.

[9] A. JENNINGS, *A compact storage scheme for the solution of simultaneously equations*, Comput. J., 9 (1966), pp. 281–285.

[10] L. T. KOU, *Polynomial complete consecutive information retrieval problems*, this Journal, 6 (1977), pp. 67–75.

[11] J. K. REID, *Solutions of linear systems of equations: direct methods (general)*, in Sparse Matrix Techniques, V. A. Barker, ed., Lecture Notes in Mathematics 572, Springer-Verlag, Berlin, 1977, pp. 102–129.

[12] A. TUCKER, *A structure theorem for the consecutive ones property*, J. Combin. Th., 12(B) (1972), pp. 153–162.

[13] M. VELDHORST, *An analysis of sparse matrix storage schemes*, Mathematical Centre Tract 150, Mathematical Centre, Amsterdam, the Netherlands, 1982.

[14] M. YANNAKAKIS, unpublished results.