Data Mining and Knowledge Discovery (2023) 37:2564–2588
https://doi.org/10.1007/s10618-023-00954-4

Column-coherent matrix decomposition

Nikolaj Tatti[1]

Received: 11 February 2023 / Accepted: 28 June 2023 / Published online: 11 August 2023

Abstract
Matrix decomposition is a widely used tool in machine learning with many applications such as dimension reduction or visualization. In this paper we consider decomposing X, a matrix of size $n \times m$, to a product WS where we require that S, a matrix of size $n \times k$, needs to have consecutive ones property. More specifically, we require that each row of S needs to be in the form of $0, \ldots, 0, 1, \ldots, 1, 0, \ldots, 0$. Such decompositions are particularly meaningful if X is a matrix where each row represents a time series; in such a case the ones in each row in S represent a time segment. We show that the optimization problem is inapproximable. To solve the problem we propose 5 different algorithms. The first two algorithms are based on solving iteratively S while keeping W fixed and then solving W while keeping S fixed. The next two algorithms are based on greedily optimizing a single row in S and the corresponding column in W. The last algorithm first finds the optimal decomposition of with $2k - 1$ *non-overlapping* rows, and then greedily combines the rows until k rows remain. We compare the algorithms experimentally, focusing on the quality of the decomposition as well as the computational time. We show experimentally that our algorithms yield interpretable results in practical time.

Keywords Matrix decomposition · Consecutive ones property · Segmentation · Dynamic program · Approximation algorithm

1 Introduction

Matrix decomposition, where a matrix X of size $n \times m$ is approximated with a product WS, where W and S are both of rank k, has been a standard tool in data mining with applications such as reducing dimensionality and visualization. In order to improve the interpretability of the decomposition, variants have been

Responsible editor: Charalampos Tsourakakis.

✉ Nikolaj Tatti
nikolaj.tatti@helsinki.fi

[1] HIIT, University of Helsinki, Helsinki, Finland

Table 1 Summary of the proposed algorithms

Algorithm	Needed additional parameters	Computational complexity
IExact	Initial W	$\mathcal{O}\left(3^k km + 2^k nm + T\right)$
IHill	Initial W and S	$\mathcal{O}(mnk + T)$
GExact	Initial W and S	$\mathcal{O}\left(m^2 nk\right)$
GEst	Initial W and S, guarantee ϵ	$\mathcal{O}(mnk/\epsilon)$
Merge	Beam width w	$\mathcal{O}\left(knm^2 + wk^3 T\right)$

Here $T \in \mathcal{O}\left(k^3 + mk^2 + nmk\right)$ is the time needed to solve a least squares problem. The second column shows parameters needed in addition to the input matrix and number of components k. The running times are per iteration, except for Merge

proposed such as non-negative matrix factorization (Wang and Zhang 2012), sparse matrix factorization (Gupta et al. 1997), or factorizations that prefer smooth rows (Rallapalli et al. 2010; Roughan et al. 2011; Xiong et al. 2010; Hsiang-Fu et al. 2016; Chen and Cichocki 2005).

In this paper we consider approximating X with WS, where S needs to be a binary matrix with consecutive ones property, that is, each row needs to be of form $0, \ldots, 0, 1, \ldots, 1, 0, \ldots, 0$. Such decomposition is interesting if X has a natural column order, for example, X consists of n time-series, each with m aligned time stamps. In such a case, each row in S corresponds to a range, a potentially interesting feature.

We first show that our problem is **NP**-hard, and even inapproximable. In order to find good decompositions we propose 5 algorithms, see Table 1 for summary.

The first two algorithms are based on the iterative approach where we iteratively update S while fixing W and solve W while fixing S. We show that, unlike finding W, solving S is inapproximable. We propose an exponential time algorithm, IExact, that finds S exactly using a dynamic program. IExact is fixed-parameter tractable and is reasonable for smaller values of k. For large values of k we propose an iterative update approach IHill, where each row of S is optimized while keeping the remaining rows fixed.

The next two algorithms are based on optimizing a row of S and the corresponding column of W simultaneously, while keeping the other rows fixed. The first algorithm GExact solves this subproblem exactly in $\mathcal{O}\left(m^2 n\right)$ time. If m is large, then this approach may be infeasible. Consequently, we also propose an algorithm GEst that $(1 + \epsilon)$-approximates the optimal row in $\mathcal{O}(mn/\epsilon)$ time.

The last algorithm is a bottom-up algorithm. We start by considering a decomposition WS where the segments of S must also be disjoint. We show that such decomposition with $2k - 1$ components is as good as the original decomposition with k components. We argue that we can solve the former in polynomial time with a dynamic program. Once, we have found the initial solution we greedily merge the rows until k rows remain.

The remainder of the paper is organized as follows. We introduce the notation and define the optimization problem in Sect. 2. In Sect. 3 we show the

NP-hardness of the problem and discuss how to solve the problem exactly. We introduce the algorithms in Sects. 4, 5 and 6. Section 7 is devoted to related work. We present the experimental evaluation in Sect. 8, and conclude the paper with discussion in Sect. 9.

2 Preliminaries and problem definition

Throughout the paper we will use X, a matrix of size $n \times m$, as our input data. We will write $X_{\cdot i}$ and $X_{i\cdot}$ to indicate the ith column and row, respectively. We will write $\|X\|_F^2 = \sum_{ij} X_{ij}^2$ to indicate the (squared) Frobenius norm.

We are interested in approximating X with WS, where S has a particular shape. We say that a binary matrix S is a C1P matrix if its first row is full 1 s, and the remaining rows have *consecutive ones* property: every row is of form $(0, \dots, 0, 1, \dots, 1, 0, \dots, 0)$, that is, the 1 s on every row are all consecutive.

Our main optimization problem is as follows.

Problem 2.1 (Dcmp) Given a real-valued matrix X of size $n \times m$ and an integer k, find a C1P matrix S of size $k \times m$ and a real-valued matrix W of size $n \times k$ such that $\|X - WS\|_F$ is minimized.

We should point out that the reason for having the first row in S equal to 1 is to have $W_{\cdot 1}$ act as a bias term.

In addition, we will consider the two subproblems where either W or S is fixed. We will use these two problems extensively.

Problem 2.2 (Dcmp-W) Given a real-valued matrix X of size $n \times m$ and a C1P matrix S of size $k \times m$, find a real-valued matrix W of size $n \times k$ such that $\|X - WS\|_F$ is minimized.

Problem 2.3 (Dcmp-S) Given a real-valued matrix X of size $n \times m$, a real-valued matrix W of size $n \times k$ find a C1P matrix S of length $k \times m$, such that $\|X - WS\|_F$ is minimized.

3 Exact algorithm for Dcmp

Before introducing more practical algorithms in the following section, let us first show that Dcmp is **NP**-hard, and consider an exponential algorithm for solving the problem exactly.

Proposition 3.1 *Dcmp is* **NP***-hard even for* $n = 1$.

Proof We will prove the claim by reducing Subset-Sum, a known **NP**-complete problem, to Dcmp. In Subset-Sum we are given a multiset of integers $y_1, \dots, y_\ell$

with $y_i > 0$ and an integer T, and are asked whether there is an index set I such $\sum_{i \in I} y_i = T$.

Assume an instance of Subset-Sum. We set X be a matrix of size $1 \times (\ell + 1)$, where $X_{1i} = \sum_{j=1}^{i} y_j$ for $i = 1, \ldots, \ell$. We also set $X_{1(\ell+1)} = T$. Finally, we set $k = \ell$.

We claim that Subset-Sum has a solution if and only if there is C1P-decomposition such that $X = WS$.

Assume that there are W and S such that $X = WS$. Since $X_{12} > X_{11}$, there must be a row in S where the first 1 is at column 2. Similarly, since $X_{13} > X_{12}$, there must be a row in S where the first 1 is at column 3. Applied iteratively, we see that, for each $i = 1, \ldots, \ell$, there is a row in S where the first 1 is at column i. We can shuffle these rows and safely assume that first 1 in the ith row is at column i.

The assumption that $X = WS$ forces $W_{1i} = y_i$. Let $I = \left\{ i \mid S_{i(\ell+1)} = 1 \right\}$ be the indices of rows with 1 s in the last column. Then since $X = WS$, we have $T = X_{1(\ell+1)} = \sum_{i \in I} W_{1i} = \sum_{i \in I} y_i$.

To prove the other direction, assume that there is a set I such that $\sum_{i \in I} y_i = T$. Set $S_{ij} = 1$ if $i \leq j \leq \ell$, or if $i \in I$ and $j = \ell + 1$. Set $W_{1i} = y_i$. Then $X = WS$. □

The proof of Proposition 3.1 shows that Dcmp is not only **NP**-hard but also inapproximable.

Corollary 3.1 *Let* $\mathrm{OPT}(X, k)$ *be the optimal cost for Dcmp. Assume that there is a polynomial-time algorithm* $\mathrm{ALG}(X, k)$ *finding a decomposition with a cost that yields an approximation guarantee* $\mathrm{OPT}(X, k) \geq \alpha(X, k)\mathrm{ALG}(X, k)$, *for some* $\alpha > 0$. *Then* $\mathbf{P} = \mathbf{NP}$. *This holds even if we limit X to have* $n = 1$.

Proof The proof in Proposition 3.1 showed that it is **NP**-complete to test whether there is a C1P-decomposition such that $\|X - WS\|_F^2 = 0$. This immediately implies that there is no algorithm that yields multiplicative approximation guarantee as we would be able to use the algorithm to test whether there is a decomposition with no error. □

We can provide more fine-grained complexity result if we assume that Strong Exponential Time Hypothesis (SETH) holds.[1]

Corollary 3.2 *Assume that SETH holds. Assume that we can k-decompose X, a matrix of size* $n \times m$, *in* $f(\Delta, n, m, k)$ *time, where* $T = \max_{i,j} \left| X_{ij} \right|$. *Then, for each* $\epsilon > 0$, *there is* δ *such that* $f(T, 1, k + 1, k) \notin \mathcal{O}\left(T^{1-\epsilon} 2^{\delta k}\right)$.

Proof The claim follows from the proof of Proposition 3.1 and Theorem 1 in (Abboud et al. 2022). □

[1] SETH (Impagliazzo and Paturi 2001) states that there is no algorithm solving k-SAT of size m with n variables in $\mathcal{O}(poly(m)2^{cn})$ time, where $c < 1$ is a constant.

Proposition 3.1 states that we cannot solve Dcmp in polynomial time, unless $\mathbf{P} = \mathbf{NP}$. However, we can find the exact solution by enumerating all possible matrices S and solve the sub-problem Dcmp-W.

Note that Dcmp-W is a standard multivariate least squares problem with the optimal solution being

$$W = XS^T(SS^T)^{-1} \quad .$$

Here, we assume that SS^T is invertible. If this does not hold, we can delete the dependent rows, and set the corresponding columns in W to 0.

Assuming, for simplicity, a naive implementations of matrix multiplication and inversion, solving Dcmp-W can be done in $\mathcal{O}\left(k^3 + mk^2 + nmk\right)$ time. There are $\binom{m+1}{2} \in \mathcal{O}\left(m^2\right)$ different rows of length m with consecutive ones property. A C1P matrix of size $k \times m$ has $k-1$ such rows. Consequently, there are $\mathcal{O}\left(m^{2k-2}\right)$ such matrices, leading to the following result.

Proposition 3.2 *The solution for Dcmp can be found in* $\mathcal{O}\left(m^{2k-2}(k^3 + mk^2 + nmk)\right)$ *time.*

We should point out that this enumeration is not practical unless m and k are both small. We present this result mainly to contrast the exponential solution of Dcmp-S in the next section.

4 Iterative algorithm

A natural approach for obtaining a good decomposition is an iterative method, shown in Algorithm 1, where we first fix W and find optimal S (Dcmp-S) and then fix S and solve W (Dcmp-W). This is repeated until convergence. Note that we need to provide initial W as a parameter. We will use Merge, described in Sect. 6, for initial W.

Algorithm 1: IExact(X, W), iterative method for decomposing X. The parameter W is the initial weight matrix.

```
while changes do
    S ← solution for Dcmp-S;
    W ← solution for Dcmp-W;
```

As we saw in the previous section, we can solve Dcmp-W. In this section, we consider two approaches for solving Dcmp-S. Unfortunately, Dcmp-S, unlike Dcmp-W, is an **NP**-hard problem. We will show how to solve Dcmp-S in fixed parameter tractable time, and also provide a polynomial-time variant, IHill, that optimizes rows of S one at a time while keeping the remaining rows fixed.

We should point out that it is possible that after the update S has a smaller rank than k. In that case we remove the dependent rows and add new rows containing a single 1 so that S will have a rank of k.

4.1 Exact solution for Dcmp-S

Let us first prove the hardness of Dcmp-S.

Proposition 4.1 *Dcmp-S is* **NP***-hard even for* $n = m = 1$.

Proof The proof is a simpler version of the proof of Proposition 3.1. We reduce Subset-Sum to Dcmp.

Assume an instance of Subset-Sum. We set $n = m = 1$ and $X_{11} = T$. We also set $k = \ell + 1, W_{11} = 0$ and $W_{1i} = y_{i-1}$ for $i = 1, \ldots, \ell + 1$.

Then it immediately follows that Subset-Sum has a solution if and only if there is C1P matrix S such that $X = WS$. □

Moreover, the argument given in Corollary 3.1 can be used to show that Dcmp-S is also inapproximable.

Corollary 4.1 *Let* $\mathrm{OPT}(X, W)$ *be the optimal cost for Dcmp-S. Assume that there is a polynomial-time algorithm* $\mathrm{ALG}(X, W)$ *finding a decomposition with a cost that yields an approximation guarantee* $\mathrm{OPT}(X, W) \geq \alpha(X, W)\mathrm{ALG}(X, W)$, *for some* $\alpha > 0$. *Then* $\mathbf{P} = \mathbf{NP}$. *This holds even if we limit X to have* $n = m = 1$.

Futhermore, we can use SETH to provide a more fine-grained complexity statement.

Corollary 4.2 *Assume that SETH holds. Assume that we can solve Dcmp-S for X and W in* $f(\Delta, n, m, k)$ *time, where* $T = \max_{i,j} \left|X_{ij}\right|$, $n \times m$ *is the size of X and* $n \times k$ *is the size of W. Then, for each* $\epsilon > 0$, *there is* δ *such that* $f(T, 1, 1, k) \notin \mathcal{O}\left(T^{1-\epsilon} 2^{\delta k}\right)$.

Proof The claim follows from the proof of Proposition 4.1 and Theorem 1 in (Abboud et al. 2022). □

Next we will show that even though Dcmp-S is inapproximable, the problem is fixed parameter tractable. More precisely, we will show that we can solve Dcmp-S in $\mathcal{O}\left(3^k km + 2^k nm\right)$ time.

We can solve Dcmp-S exactly with a dynamic program by computing an array $q[A, B, j]$, where $A \subseteq B \subseteq \{2, \ldots, k\}$ are two index sets and $j \in [m]$.

In order to define $q[A, B, j]$, assume that we are given an index $j = 1, \ldots, m$ and two index sets $A \subseteq B \subseteq \{2, \ldots, k\}$. Write X' to be the first j columns of X. We define $q[A, B, j]$ to be the cost of the optimal C1P decomposition WS of X' such that the rows corresponding to B have at least one 1 and the rows corresponding to A do *not* have 1 at the jth column, that is,

$$B = \left\{ i \mid S_{i\ell} = 1 \text{ for some } \ell \leq j \right\} \quad \text{and} \quad A = \left\{ i \in B \mid S_{ij} = 0 \right\} \quad .$$

Note that, by definition, $S_{ij} = 1$ if and only if $i \in B \setminus A$.

Proposition 4.2 *The array q can be computed with the dynamic program,*

$$\begin{aligned} q[A,B,j] =& \left\| X_{\cdot j} - W_{\cdot 1} - \sum_{\ell \in B \setminus A} W_{\cdot \ell} \right\|_F^2 + r[A,B,j], \quad \text{where} \\ r[A,B,j] =& \min_{\substack{A' \subseteq A \\ A \subseteq B' \subseteq B}} q[A',B',j-1], \end{aligned} \tag{1}$$

and $q[A,B,0] = 0$ as the boundary case.

Proof Assume a matrix S of size $k \times j$. Let S' be S without its last column. Define two functions $b(S) = \left\{ i \geq 2 \mid S_{i\ell} = 1 \text{ for some } \ell \leq j \right\}$ and $a(S) = \left\{ i \in b(S) \mid S_{ij} = 0 \right\}$.

Assume two sets $A \subseteq B \subseteq \{2, \ldots, k\}$. We claim the following: S is a C1P matrix with $A = a(S)$ and $B = b(S)$ if and only if (1) S' is a C1P matrix with (2) $a(S') \subseteq A$ and $A \subseteq b(S') \subseteq B$, and (3) $S_{ij} = 1$ when $i > 1$ iff $i \in B \setminus A$. The claim implies that S yields $q[A, B, j]$ if and only if S' yields $r[A, B, j]$, proving the proposition.

We prove first the *only if* direction. Condition (1) follows since S is C1P matrix. Condition (3) follows from the definitions of a and b. Finally, by definition, $A = a(S) \subseteq b(S') \subseteq b(S) = B$, and since S is C1P, $a(S') \subseteq a(S)$.

Next, we prove the *if* direction. Any $i \in b(S)$ if and only if $i \in b(S')$ or $S_{ij} = 1$. Thus, $b(S) = (B \setminus A) \cup b(S') = (B \setminus A) \cup A = B$. Secondly, by definition of the last column, we have $a(S) = A$. Since S' is C1P, the only C1P violation can occur if $S_{ij} = 1$ for $i \in a(S')$. Since $a(S') \subseteq A$, this cannot happen. □

Here, the first term corresponds to the error coming from the jth column with indices in $B \setminus A$ corresponding to the 1 s in the jth column of S. The term $r[A, B, j]$ is the optimal error up to the $(j-1)th$ column such that the C1P requirement is met.

Once we have computed q, we can obtain the optimal cost of Dcmp-S by computing

$$\min \{ q[A,B,m] \mid A \subseteq B \} \quad . \tag{2}$$

In order to recover S, let us first write $A^{(m)}$ and $B^{(m)}$ to be the minimizers for Eq. 2. During the dynamic program, we store the minimizers of $r[A, B, j]$ Eq. 1 for every A, B, and j. We then iteratively define $A^{(j-1)}$ and $B^{(j-1)}$ to be the sets responsible for $r[A^{(j)}, B^{(j)}, j]$. The optimal S is then defined by setting $S_{ij} = 1$ if and only if $i \in B^{(j)} \setminus A^{(j)}$.

Next we show the running time needed to solve S.

Proposition 4.3 *Dcmp-S can be solved in $\mathcal{O}\left(3^k km + 2^k nm\right)$ time.*

Proof We can precompute the vectors $\sum_{\ell \in S} W_{\cdot \ell}$ for every S in $\mathcal{O}\left(2^k n\right)$ time. Once, precomputed we can compute the first term in Eq. 1 in $\mathcal{O}(n)$ time. There are $\mathcal{O}\left(2^k\right)$

different subsets $S = B \setminus A$ and there are m different j. Consequently, computing the needed error terms requires $\mathcal{O}\big(2^k(k+nm)\big)$ time in total.

Consider now $r[A, B, j]$. We can show that

$$\begin{aligned} r[A,B,j] &= \min(q[A,B,j-1], \alpha, \beta), \quad \text{where} \\ \alpha &= \min\{r[A \setminus \{a\}, B, j] \mid a \in A\}, \\ \beta &= \min\{r[A, B \setminus \{b\}, j] \mid b \in B \setminus A\}, \end{aligned}$$

holds. This identity allows us to compute $r[A, B, j]$ in $\mathcal{O}(k)$ time assuming we have computed $r[A', B', j]$ for every subset A' and B'.

Since an index in $[k]$ can either belong to A, or belong to $B \setminus A$, or be outside of both sets, there are $\mathcal{O}\big(3^k\big)$ valid pairs of (A, B). Consequently, are $\mathcal{O}\big(3^k m\big)$ cells in r. Thus computing r requires $\mathcal{O}\big(3^k km\big)$ time and computing q requires $\mathcal{O}\big(3^k km + 2^k nm\big)$ time. □

We should point out that unlike the exact exponential algorithm given in Sect. 3 this algorithm may be practical as long as k is small. These cases may be particularly useful if we are using the obtained W to visualize the rows of X in a plane.

Proposition 4.3 shows that Dcmp-S is a fixed-parameter tractable problem. It is not known whether the main problem Dcmp is also fixed-parameter tractable. We conjecture that some techniques used by Fomin et al. (2020) can be adopted to develop a fixed-parameter tractable algorithm.

4.2 Hill-climbing algorithm for Dcmp-S

If k is large, the dynamic program given in the previous section becomes impractical. In such cases, we propose the following optimization strategy. Instead of optimizing all rows of S simultaneously, we select one row to optimize while keeping the remaining rows fixed, as shown in Algorithm 2. We repeat this step for every row.

Algorithm 2: IHill(X, W, S), iterative hill-climbing method for decomposing X. The parameters W and S is the initial decomposition.

```
while changes do
    foreach i = 2..., k do
        S_i. ← optimal ith row minimizing the error;
    W ← solution for Dcmp-W;
```

We can find the optimal row $S_{i\cdot}$ using a similar dynamic program as in the previous section, except now the array is going to be significantly smaller. More formally, let $j \in [m]$ be an index. We define $q[0, j]$ to be the error of the first j columns with $S_{i\cdot}$ being 0, $q[1, j]$ to be the optimal error of the first j columns with $S_{ij} = 1$, and $q[2, j]$ to be the optimal error of the first j columns with $S_{ij} = 0$

and $S_{i\cdot}$ not being full of 0 s. Naturally, when computing $q[\cdot, j]$ we require that $S_{i\cdot}$ satisfies the consecutive ones property.

To simplify the notation, let us write $E = X - WS'$, where S' is equal to the current S except that $S'_{i\cdot} = 0$.

Proposition 4.4 *The array q can be computed using the dynamic program,*

$$q[0,j] = \left\|E_{\cdot j}\right\|_F^2 + q[0,j-1],$$
$$q[1,j] = \left\|E_{\cdot j} - W_{\cdot i}\right\|_F^2 + \min(q[0,j-1], q[1,j-1]),$$
$$q[2,j] = \left\|E_{\cdot j}\right\|_F^2 + \min(q[1,j-1], q[2,j-1]),$$

and $q[0,0] = q[1,0] = q[2,0] = 0$ as the boundary case.

Proof We will prove the claim by induction over j. The case $j = 1$ is trivial. Assume that the claim holds for $j - 1$.

The case $q[0, j]$ is trivial.

Let S be responsible for $q[1, j]$. Write $\Delta = \left\|E_{\cdot j} - W_{\cdot i}\right\|_F^2$. If S_{ij} contains the only 1 on the row i, the cost is $q[1,j] = \Delta + q[0,j-1]$. Otherwise, since S is C1P, $S_{i(j-1)} = 1$ and $q[1,j] = \Delta + q[1,j-1]$. The value $q[1, j]$ is the minimum of the two cases, proving the case for $q[1, j]$.

Let S be responsible for $q[2, j]$. Write $\Delta = \left\|E_{\cdot j}\right\|_F^2$. If $S_{i(j-1)} = 1$, the cost is $q[2,j] = \Delta + q[1,j-1]$. Otherwise, $S_{i(j-1)} = 0$ but the row i is not a zero vector, consequently, $q[2,j] = \Delta + q[2,j-1]$. The value $q[2, j]$ is the minimum of the two cases, proving the case for $q[2, j]$. □

Once q is computed, the optimal error is equal to $\alpha = \min_x q[x, m]$. In order to restore the corresponding $S_{i\cdot}$, let us define $t_j = 1$ if $q[1,j] < q[2,j]$, and 0 otherwise. Let us also define $s_j = 1$ if $q[0,j] < q[1,j]$, and 0 otherwise. If $\alpha = q[0, m]$, then the corresponding $S_{i\cdot}$ is all zeros. Otherwise, the last column containing 1 s is equal to $b = \max\left\{j \mid t_j = 1\right\}$. To extract the starting point of 1 s, we set $a = \max\left\{j < b \mid s_j = 1\right\} + 1$, or $a = 1$ is the set is empty.

Next we show the time needed to update S.

Proposition 4.5 *Updating S in IHill requires $\mathcal{O}(mnk)$ time.*

Proof For a fixed i, the matrix E can be computed in $\mathcal{O}(mn)$ time by maintaining the matrix, say, $Z = X - WS$, and using the identity $E = Z + W_{\cdot i}S_{i\cdot}$. Computing a single entry in q requires $\mathcal{O}(n)$ time due to error term. Since there are $\mathcal{O}(m)$ cells in q, we can find optimal row in $\mathcal{O}(nm)$ time. Since, there are $\mathcal{O}(k)$ different values for i, the result follows. □

5 Greedy algorithm

In the previous section both algorithms kept W fixed while optimizing S. In this section we consider approaches that update W and S at the same time.

More specifically, we propose optimizing a single row, say ℓ, of S, the corresponding column $W_{\cdot\ell}$, and the first column $W_{\cdot 1}$,[2] while keeping the remaining cells in W and S fixed. The algorithm, given in Algorithm 3 enumerates over ℓ until no gain is possible.

Algorithm 3: GExact(X, W, S, k), greedy approach for decomposing X. The parameters W and S is the initial decomposition.

```
while changes do
    foreach ℓ = 2 ..., k do
        optimize S_ℓ., W_·1, and W_·ℓ ;
```

In addition to GExact, we consider a faster variant, Est, that provides $(1+\epsilon)$ approximation of the optimal row.

5.1 Solving the greedy step exactly

Our next step is to solve the optimization problem in Greedy exactly and as quickly as possible.

To this end, fix ℓ. Let us write $R = X - W'S$, where W' are the current weights except that $W_{\cdot 1}$ and $W_{\cdot\ell}$ are set to 0. In other words, we consider the differences between X and the decomposition without the first and the ℓth dimension.

Assume that we have selected a new $S_{\ell\cdot}$ such that 1 s start at the ith column and end at the jth column. Then it is straightforward to see that the optimal error is equal to $a(i,j) + b(i,j)$, where

$$a(i,j) = \sum_{x=i}^{j} \left\|R_{\cdot x} - \alpha\right\|_F^2, \quad b(i,j) = \sum_{\substack{x < i, \text{ or} \\ x > j}} \left\|R_{\cdot x} - \beta\right\|_F^2, \tag{3}$$

where

[2] Recall that the first column is essentially the bias term of the decomposition.

$$\alpha = \frac{1}{j-i+1}\sum_{x=i}^{j} R_{\cdot x}, \quad \beta = \frac{1}{m-j+i-1}\sum_{\substack{x<i, \text{ or} \\ x>j}} R_{\cdot x}$$

Here, $a(i,j)$ is the error of the columns between i and j, and $b(i,j)$ is the error of the remaining columns. Moreover, this error is achieved if we set

$$W_{\cdot 1} = \beta, \quad \text{and} \quad W_{\cdot \ell} = \alpha - \beta \quad .$$

To find the optimal $S_{\ell\cdot}$, we simply test every index pair $i \leq j$, leading to the following proposition.

Proposition 5.1 *A single iteration of* GEXACT *runs in* $\mathcal{O}\left(m^2 nk\right)$ *time.*

Proof For a fixed ℓ, the matrix R can be computed in $\mathcal{O}(mn)$ time by maintaining the matrix, say, $Z = X - WS$, and using the identity $R = Z + W_{\cdot\ell}S_{\ell\cdot} + W_{\cdot 1}S_{1\cdot}$.

We can compute $a(i,j)$ and $b(i,j)$ in $\mathcal{O}(n)$ time by precomputing the cumulative sums and the second moments. That is, we first precompute in $\mathcal{O}(nm)$ time the functions $f(x) = \sum_{y \leq x} R_{\cdot y}$ and $g(x) = \sum_{y \leq x} R^2_{\cdot y}$, where the square is applied element-wise. Then we can compute $a(i,j)$ in $\mathcal{O}(n)$ time with the standard identity

$$\alpha = \frac{f(j) - f(i-1)}{j-i+1} \quad \text{and}$$
$$a(i,j) = e^T(g(j) - g(i-1) - (j-i+1)\alpha^2),$$

where the square is applied element-wise and e is a vector of 1 s of length k. The computation of $b(i,j)$ is similar.

Since there are $\mathcal{O}\left(m^2\right)$ pairs of (i,j) and $\mathcal{O}(k)$ different values of ℓ, the claim follows. □

5.2 Approximating the greedy step fast

Finding the optimal row requires $\mathcal{O}\left(m^2 n\right)$ time which may be impractical for large values of m. In this section we design an algorithm that $(1+\epsilon)$-approximates the optimal row in $\mathcal{O}(mn/\epsilon)$ time.

The main idea of the algorithm is as follows: Recall the definition of a and b as given in Eq. 3. Let (i^*, j^*) be the index pair minimizing $a(i,j) + b(i,j)$. Assume that we know σ that is larger than but close to $a(i^*, j^*)$. If we were to select the largest j such that $a(i^*, j) \leq \sigma$, then $a(i^*, j)$ is close to $a(i^*, j^*)$ and $b(i^*, j) \leq b(i^*, j^*)$ since $j^* \leq j$.

We will see later that, given σ, enumerating *every* maximal interval such that $a(i,j) \leq \sigma$ can be done in linear time. The issue is that we do not know σ. Instead we compute an upper bound of $a(i^*, j^*)$, say, u and a decrement δ and test every $\sigma = u - r\delta$, where r is an integer. For each tested pair (i, j) we

compute $a(i,j) + b(i,j)$, and store the smallest cost, say ρ, as well as the pair that yielded ρ, which is the final output of the algorithm. See Algorithm 4 for the pseudo-code.

Algorithm 4: $\textsc{Est}(a, b, u, \delta)$, sub-routine for solving approximately $\min_{i \leq j} a(i,j) + b(i,j)$.

```
ρ ← ∞; σ ← u;
while σ ≥ 0 do
    i ← j ← 1;
    while i < m or j < m do
        ρ ← min(ρ, a(i, j) + b(i, j));
        if j < m and a(i, j + 1) ≤ σ then
            j ← j + 1;
        else
            i ← i + 1;
    σ ← σ − δ;
return (i, j) responsible for ρ;
```

Algorithm 5: $\textsc{GEst}(X, W, S, k, \epsilon)$, greedy approximative approach for decomposing X. The parameters W and S is the initial decomposition, and ϵ is the approximation guarantee of the optimization step.

```
while changes do
    foreach ℓ = 2 . . . , k do
        τ ← min_{i≤j} max(a(i, j), b(i, j));          // use MaxSeg(a, b)
        (i, j) ← Est(a, b, 2τ, ετ);
        update S_ℓ., W.1, and W.ℓ using (i, j);
```

We still need to select u and δ such that the approximation guarantee holds and there are not too many values of σ that need to tested. To the end, we set $u = 2\tau$ and $\delta = \epsilon\tau$, where $\tau = \min_{i \leq j} \max(a(i,j), b(i,j))$. Proposition 5.2 shows that these values are suitable. The complete pseudo-code is shown in Algorithm 5.

Before proving correctness we need to show that we can compute τ in linear time. The algorithm, MaxSeg, for finding τ is given in Algorithm 6. The algorithm enumerates index pairs in the following manner: if currently $a(i,j) < b(i,j)$

we increase j as this potentially decreases $\max(a(i,j),b(i,j))$, otherwise we increase i.

Algorithm 6: MaxSeg(a,b), linear-time sub-routine for finding $\min_{i \leq j} \max(a(i,j),b(i,j))$.

```
1 i ← j ← 1;
2 τ ← ∞;
3 while i < m or j < m do
4 |   τ ← min(τ, max(a(i,j), b(i,j)));
5 |   if j < m and a(i,j) < b(i,j) then
6 |   |   j ← j + 1;
7 |   else
8 |   |   i ← i + 1;
9 return τ;
```

Let us now prove the correctness of MaxSeg.

Lemma 5.1 *Let* $\tau = \text{MaxSeg}(a,b)$. *Then*

$$\tau = \min_{i \leq j} \max(a(i,j),b(i,j)) \quad .$$

Moreover, MaxSeg(a,b) *runs in* $\mathcal{O}(mn)$ *time*.

Proof Let i^* and j^* be the indices yielding the smallest error $\max(a(i,j),b(i,j))$. If there are ties, we select the smallest possible indices. Let i and j be the variables used in MaxSeg. To prove the lemma, we show by induction that MaxSeg has visited (i^*,j^*), or $i \leq i^*$ and $i \leq j^*$. The induction base $i = j = 1$ is trivial.

In order to prove the general case assume that the claim holds for (i,j). If MaxSeg has been visited, then we have nothing to prove. Assume otherwise, then by the induction assumption $i \leq i^*$ and $j \leq j^*$.

If $i = i^*$ and $j = j^*$, then we have nothing to prove.

If $i < i^*$ and $j < j^*$, then the induction step follows trivially since we increase i or j by 1.

Assume $i = i^*$ and $j < j^*$.

If $a(i^*,j^*) < b(i^*,j^*)$, then

$$a(i,j) \leq a(i^*,j^*) < b(i^*,j^*) \leq b(i,j)$$

and MaxSeg increases j.

If $a(i^*,j^*) \geq b(i^*,j^*)$, then, by the optimality of j^*, $a(i^*,j^*-1) < b(i^*,j^*-1)$ as otherwise (i^*,j^*-1) is also optimal. Consequently,

$$a(i,j) \leq a(i^*,j^*-1) < b(i^*,j^*-1) \leq b(i,j) \quad .$$

and MaxSeg increases j.

Assume $i < i^*$ and $j = j^*$.

If $a(i^*,j^*) \geq b(i^*,j^*)$, then

$$a(i,j) \geq a(i^*,j^*) \geq b(i^*,j^*) \geq b(i,j)$$

and MaxSeg increases i.

If $a(i^*,j^*) < b(i^*,j^*)$, then, by the optimality of i^*, $a(i^*-1,j^*) > b(i^*-1,j^*)$ as otherwise (i^*-1,j^*) is also optimal. Consequently,

$$a(i,j) \geq a(i^*-1,j^*) > b(i^*-1,j^*) \geq b(i,j) \quad .$$

and MaxSeg increases i. This proves the induction step.

The while-loop in MaxSeg(a,b) is executed at most $2m$ times. Each $a(i,j)$ or $b(i,j)$ requires $\mathcal{O}(n)$ time, proving the claim. □

Next, we prove the approximation result.

Proposition 5.2 *Let* $O = \min_{i \leq j} a(i,j) + b(i,j)$ *be the optimal error. Let* $\tau = \text{MaxSeg}(a,b)$. *Assume* $\epsilon > 0$. *Let* ρ *be the cost of the row returned by* Est$(2\tau, \epsilon\tau)$. *Then* $\rho \leq (1+\epsilon)O$.

Moreover, Est$(2\tau,\epsilon\tau)$ *runs in* $\mathcal{O}(mn/\epsilon)$ *time.*

Proof Let (i^*,j^*) be the indices yielding O. Similarly, let (i',j') be the indices yielding τ. Lemma 5.1 implies

$$\begin{aligned}
\tau &\leq \max(a(i^*,j^*), b(i^*,j^*)) \\
&\leq a(i^*,j^*) + b(i^*,j^*) \\
&= O \\
&\leq a(i',j') + b(i',j') \\
&\leq 2\max(a(i',j'), b(i',j')) = 2\tau \quad .
\end{aligned}$$

To summarize $\tau \leq O \leq 2\tau$.

Let σ be the smallest variable used by Est such that $\sigma \geq a(i^*,j^*)$. Such variable exist since $a(i^*,j^*) \leq O \leq 2\tau$. During this iteration, let j be the largest variable visited by Est when $i = i^*$. Note that since $a(i^*,j^*) \leq \sigma$, we must have $j \geq j^*$ as otherwise j is not maximal.

Since σ is minimal, we have $a(i^*,j^*) \geq \sigma - \epsilon\tau$. Consequently,

$$\begin{aligned}
\rho &\leq a(i^*,j) + b(i^*,j) \\
&\leq \sigma + b(i^*,j^*) \\
&\leq a(i^*,j^*) + \epsilon\tau + b(i^*,j^*) \\
&= O + \epsilon\tau \\
&\leq (1+\epsilon)O,
\end{aligned}$$

proving the first claim.

To prove the second claim note that the inner loop is executed $\mathcal{O}(m)$ times and the outer loop is executed $\mathcal{O}(u/\delta) = \mathcal{O}(1/\epsilon)$ times. □

The running time results in Lemma 5.1 and in Proposition 5.2 immediately imply the following result.

Proposition 5.3 *A single iteration of* GEst *runs in* $\mathcal{O}(mnk/\epsilon)$ *time.*

In order to improve the performance in practice, at the end of each iteration we optimize S using the for-loop given in IHill. This update does not change the running time analysis.

We should point out that the problem relevant to minimizing $a(i,j) + b(i,j)$ is *segmentation*, where the goal is to partition a sequence into k segments minimizing some error function. The segmentation problem can be solved exactly in quadratic time (Bellman 1961) and approximated in polylogaritmic time (Guha et al. 2006; Tatti 2019).[3] However, we cannot use these results directly since $b(i,j)$ depends both on the beginning and on the end of the row.

6 Bottom-up algorithm

In this section we introduce our final algorithm. We start by considering an easier decomposition problem.

We say that a binary matrix S is segment matrix, if all the rows have consecutive ones property, are disjoint, and every column has at least 1. This definition leads to the following optimization problem.

Problem 6.1 (Seg) Given a matrix X of size $n \times m$ and an integer k, find a segment matrix S of size $k \times m$ and a matrix W of size $n \times k$ such that $\|X - WS\|_F$ is minimized.

We should point out that Seg is equivalent to segmenting X into k segments, each segment correspond to 1 s in a row of S, while minimizing L_2 cost. This problem can be solved with a dynamic program in $\mathcal{O}\left(km^2n\right)$ time (Bellman 1961).

Proposition 6.1 *Assume a C1P matrix S of size $k \times m$ and a weight matrix W of size $n \times k$. Let $k' = 2k - 1$. Then there is a segment matrix T of size $k' \times m$ and a weight matrix U of size $n \times k'$ such that $WS = UT$.*

Proof We claim that there are at most $k' - 1$ columns in S that are different than the column on their right. The claim is trivial for $k = 1$, and follows immediately by induction since a new row in S introduce at most 2 such columns.

Let $i_1, \ldots, i_{k'-1}$ be these columns. Write also $i_0 = 0$ and $i_{k'} = n$. We define T such that the jth row has 1 s between $i_{j-1} + 1$ and i_j, and 0 otherwise. Finally, we set U such that jth column is equal to $WS_{\cdot i_j}$. The proposition follows. □

[3] Here we assume that the cost of a single segment can be obtained in constant time.

The proposition leads to the following approach, for which the pseudo-code is given in Algorithm 7. Given X we find a decomposition with $2k-1$ components WS, where S is a segment matrix. In order to manipulate S we will represent these matrices as a sequence of pairs $I = ((a_i, b_i))$, where a_i and b_i indicate the column end points containing 1 s at the ith row.

In order to transform S to C1P-matrix with only k segments, we first add a constant row full of 1 s, that is, we add $(1, m)$ to I.

We then test each pair, say (a, b) and (a', b'), in I. We replace these pairs with a new pair of $(s, t) = (\min(a, a'), \max(b, b'))$, generating a new candidate. We test the new candidate by solving Dcmp-W. In addition, if I contains a pair that starts at $t+1$, we generate a new candidate by extending that pair to $\max(a, a')$, and test the new candidate. Similarly, if I contains a pair that ends at $s-1$, we generate a new candidate by extending that pair to $\min(b, b')$, and test the new canidate.

After the tests, we keep w best candidates, where w is a user parameter specifying the width of the beam search. As a tiebreaker we use the total number of 1 s. This procedure is repeated k times for each of the w candidates, after which only k rows remain in each I. We select the candidate with the lowest score as the final output.

Algorithm 7: Merge(X, k, w), beam search bottom-up method for decomposing X, the parameter w specifies the width of the beam search.

```
I ← solution of Seg with 2k − 1 segments;
prepend (1, m) to I;
beam ← {I};
foreach r = 0, ..., k − 1 do
    for I ∈ beam do
        for 1 ≤ i < j ≤ 2k − r do
            (a, b) ← ith interval in I;
            (a', b') ← jth interval in I;
            x ← (min(a, a'), max(b, b'));
            I' ← I with ith and jth pairs removed, and x added;
            construct S from I' and solve Dcmp-W(X, S);
            if there is ℓth interval in I' starting at max(b, b') + 1 then
                I'' ← extend the ℓth interval to max(a, a');
                construct S from I'' and solve Dcmp-W(X, S);
            if there is ℓth interval in I' ending at min(a, a') − 1 then
                I'' ← extend the ℓth interval to min(b, b');
                construct S from I'' and solve Dcmp-W(X, S);
    beam ← w best Is tested;
return S constructed from the best I in beam and the corresponding W;
```

Next, let us analyze the running time of Merge.

Proposition 6.2 *Merge runs in* $\mathcal{O}\big(knm^2 + wk^3(k^3 + mk^2 + nmk)\big)$ *time.*

Proof Finding initial segmentation requires $\mathcal{O}\big(knm^2\big)$ time.

During the merging phase we need to solve $\mathcal{O}\left(wk^3\right)$ DCMP-W problems for which we need $\mathcal{O}\left(k^3(k^3 + mk^2 + nmk)\right)$ time. □

We should point out that MERGE has a high dependency on k, due to the excessive comparisons when merging rows. Luckily, k is typically of moderate size. We leave developing more aggressive merging strategies for large k as a future line of work.

7 Related work

In this paper, we require that S has a very specific shape, making the neighboring columns of WS to look similar. Instead of having a hard constraint, previous approaches regularized S by punishing large changes between neighboring columns. Hsiang-Fu et al. (2016) regularized matrix decomposition with a score based on a Markov model, thus encouraging discovering temporal dependencies. Chen and Cichocki (2005) considered non-negative decompositions WS where the rows S are regularized by the error when compared against exponentially weighted moving average, thus encouraging smooth behavior of rows. In similar spirit, regularizations have been propposed where a column of S is compared to its neighboring columns, encouraging having similar values. (Rallapalli et al. 2010; Roughan et al. 2011; Xiong et al. 2010).

In related work, Tatti and Miettinen (2019) considered permuting and approximating binary matrix X with $W^T \circ S$ where W and S are both C1P-matrices and $\circ$ is a boolean multiplication. In their setup the order of column and rows is not fixed but one also needs to find a good permutation of rows and columns, as well as find W and S. Discovering such decomposition is closely related to discovering tilings, regions of 1 s in a binary matrix that are dense. Discovering such tilings have been proposed by Miettinen et al. (2008), minimizing the Frobenius norm. In addition, Gionis et al. (2004), Tatti and Vreeken (2012) proposed mining geometric tiles, that is, tiles that are column and row coherent, organized as trees while maximizing a likelihood-based score. Geerts et al. (2004) proposed mining covering binary data with k tiles while maximizing the number of 1 s covered by the tiles. Similarly, Henelius et al. (2016) proposed mining tiles from time series that were column-coherent while maximizing same objective. Kontonasios and De Bie (2010) proposed mining tiles maximizing a score based on a maximum entropy model. Xiang et al. (2008) considered representing the data exactly with tiles while minimizing their border length. Discovering tiles in a data stream was considered by Lam et al. (2014).

The aforementioned approaches are focused on modeling binary data. Similar to tilings, finding biclusters, submatrices in real-valued data that are coherent according to a given objective function have been proposed (Cheng and Church 2000; Madeira and Oliveira 2005; Zhang et al. 2005; Hartigan 1972). These approaches do not regularize or constraint biclusters based on the column/row order.

Our method resembles a problem of segmentation, where the goal is to partition the sequence into k segments such that segments are cohesive according to some

error function (Bellman 1961). In our case, the segments would be the neighboring columns in S having equal values. While the standard segmentation problem is solvable in polynomial time (Bellman 1961; Guha et al. 2006; Tatti 2019) our problem is **NP**-hard because the columns of W may participate in multiple segments. In similar fashion, Gionis and Mannila (2003) considered an **NP**-hard problem where k segments can only use $h < k$ different centroids. However, their approach cannot be applied to our problem due to the differences in the optimization problems.

8 Experimental evaluation

In this section we present our experimental evaluation.

8.1 Datasets

We used a series of synthetic datasets and 4 real-world datasets as benchmark datasets.

We generate two synthetic dataset series as follows. In the first set, each dataset is a size of 500×500 and has the form $W_{gen}S_{gen} + N$. Here S_{gen} is of size 5×500, the ith row has 1 s between $50i - 49$ and $550 - 50i$, W_{gen} has real values sampled uniformly between 1 and 2, and N is a matrix of size 500×500, consisting of Gaussian noise with 0 mean and σ^2 variance. We denote this dataset by *Mode* (σ), and vary σ.

In the second dataset series, each dataset has the form $W_{gen}S_{gen} + N$. Here, S_{gen} is of size $5 \times n$ with intervals of 1 s sampled uniformly. However, we reject cases where two segments share the same end point since the ground truth becomes ambiguous as the shorter row can be subtracted from the longer row. The ith column in W_{gen} has real values sampled uniformly between $1i$ and $2i$. Finally, N is a matrix of size $n \times m$, consisting of Gaussian noise with 0 mean and σ^2 variance. We denote this dataset by *Syn* (σ), and vary σ. Unless specified, we set $n = m = 500$.

The first real-world dataset, *Milan*, consists of monthly averages of maximum daily temperatures in Milan between the years 1763–2007.[4] The second dataset, *Power*, consists of hourly power consumption (variable `global_active_power`) of a single household over almost 4 years, a single time series representing a day.[5] The third dataset, *ECG* are heart beat data (Goldberger et al. 2000). We used MLII data of a single patient (id 106) from the MIT-BIH arrhythmia database,[6] containing both normal beats and abnormal beats with premature ventricular contraction. Each time series represent measurements

[4] https://www.ncdc.noaa.gov/.

[5] http://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption.

[6] https://physionet.org/content/mitdb/1.0.0/.

Table 2 Sizes of the datasets and results of the greedy algorithms

Dataset	n	m	GEst			GExact		
			E	T	R	E	T	R
Mode(1)	500	500	1.01	4.06	1	1.01	23.27	1
Syn(1)	500	500	1.01	4	1	1.01	46	2
ECG	2 027	253	5.01	30.33	8	5.01	236.54	21
Milan	245	12	1.95	0.1	2	1.95	0.03	2
Population	309	8	7.49	0.01	1	7.49	0.01	1
Power	1 417	24	1.28	0.24	1	1.28	0.08	1

Each matrix is a size of $n \times m$. E is the error of the decomposition normalized by the error of SVD decomposition, T is the execution time of an algorithm, R is the number of rounds

between −300 ms and 400 ms around each beat. The fourth dataset, *Population* consists of age distribution of municipalities in Finland.[7]

8.2 Setup

We implemented the 4 algorithms using Python and used a laptop with Intel Core i5 (2.3GHz) to conduct our experiments.[8]

We decomposed each dataset with 5 algorithms using $k = 5$. Since the 4 iterative algorithms require initial decomposition, we used the solution given by Merge as the starting point. To speed up computation of Merge, we used approximative segmentation algorithm by Guha et al. (2001) with $\epsilon = 0.05$. We set the beam search width to $w = 50$. Finally, we used $\epsilon = 0.1$ for GEst.

8.3 Results

Synthetic data: Our first goal is to test how well we can recover ground truth from synthetic data as a function of noise. We used Merge with $k = 5$, the other algorithms produced identical results. For comparison we used SVD decomposition in the following manner: we first demeaned every row and used these means as the first column in W (with the row being full of 1 s), we then applied SVD with 4 components to obtain the remaining part of the decomposition.

In Figs. 1a, 2a we show the error, normalized by nm, of SVD and Merge as a function of the variance of the noise σ. As expected, the cost increases as the noise increases. Moreover, SVD has a smaller cost as it does not have C1P requirements on S. In Figs. 1b, 2b, we see that the errors of the disovered decompositions are always slightly smaller than the error of the ground truth. This is largely due to the matrix W optimized to the deviations in X due to the noise.

Next, we compare how well the recovered S matches the ground truth S_{gen}. To that end, we computed L_1 distance between S discovered by Merge, here we took into account all possible row permutations of S. Note that S discovered by SVD

[7] https://stat.fi/en/statistics/vaerak.
[8] The code is available at https://version.helsinki.fi/dacs/column-coherent-decomposition.

Table 3 Results for MERGE, IHILL, and IEXACT

Dataset	Merge		IHill			IExact		
	E	*T*	*E*	*T*	*R*	*E*	*T*	*R*
Mode(1)	1.01	34.37	1.01	0.05	1	1.01	0.194	1
Syn(1)	1.01	34.47	1.01	0.05	1	1.01	0.194	1
ECG	5.17	24.01	5.01	0.206	4	4.95	0.471	4
Milan	1.95	0.92	1.95	0.004	2	1.95	0.006	1
Population	7.49	0.63	7.49	0.003	1	7.49	0.005	1
Power	1.28	1.97	1.28	0.005	1	1.28	0.011	1

E is the error of the decomposition normalized by the error of SVD decomposition, *T* is the execution time of an algorithm, *R* is the number of rounds

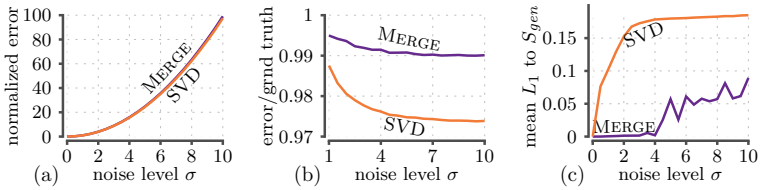


Fig. 1 Results Merge and SVD decompositions for *Mode*(σ) as a function of noise σ,averages of 10 runs. **a** cost of the decomposition, normalized by *nm*. **b** cost of the decomposition, normalized by the cost of the ground truth decomposition. **c** L_1 distance between S produced by Merge and the ground truth S_{gen}, normalized by *mk*

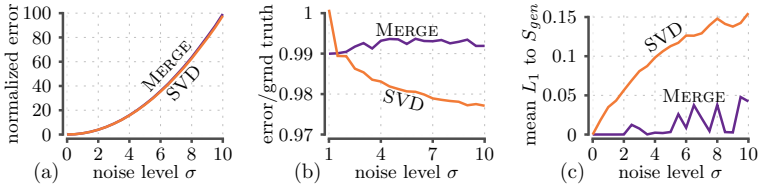


Fig. 2 Results Merge and SVD decompositions for *Syn*(σ) as a function of noise σ,averages of 10 runs. **a** cost of the decomposition, normalized by *nm*. **b** cost of the decomposition, normalized by the cost of the ground truth decomposition. **c** L_1 distance between S produced by Merge and the ground truth S_{gen}, normalized by *mk*

has orthogonal rows, which does not hold for S_{gen}. Therefore, in order to make the comparison more fair, when evaluating SVD, we projected S_{gen} to the subspace spanned by S, and computed L_1 distance between the projection and S_{gen}.

In Figs. 1c, 2c we show the L_1 distance, normalized by *mk*, of SVD and MERGE as a function of the variance of the noise σ. We see that in both cases the distance increases as a function of σ. However, MERGE is more resilient to the noise and outperforms SVD.

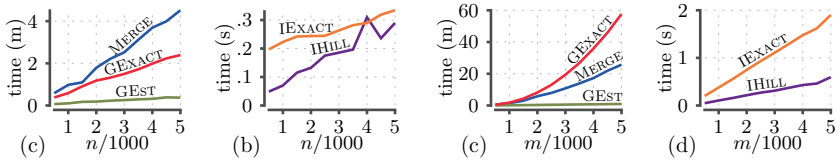


Fig. 3 **a**, **b** Running time as a function of n for $Syn(1)$ while $m = 500$. **c**, **d** Running time as a function of m for $Syn(1)$ while $n = 500$. The running times are per iteration, except for MERGE. Note that the time scales differ, and the x-axes are scaled with 1000

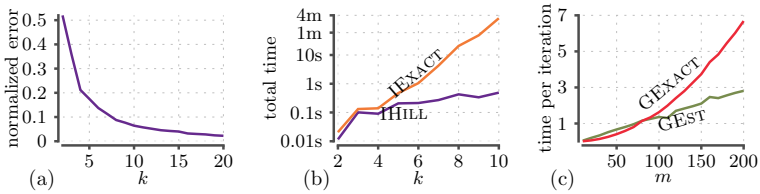


Fig. 4 **a** Error, normalized by the error of 1-decomposition, as a function for k using *ECG* and IHILL. **b** Computation time for decomposing *ECG* as a function of k. Note that the y-scale is logarithmic. **c** Time per single iteration for decomposing m first columns of *ECG* ($k = 5$)

Real-world data: Next let us look out our benchmark datasets. In Tables 2 and 3 we show the errors, running time, and number of required iterations. In order to normalize the errors we used the error of an SVD decomposition with 5 components.

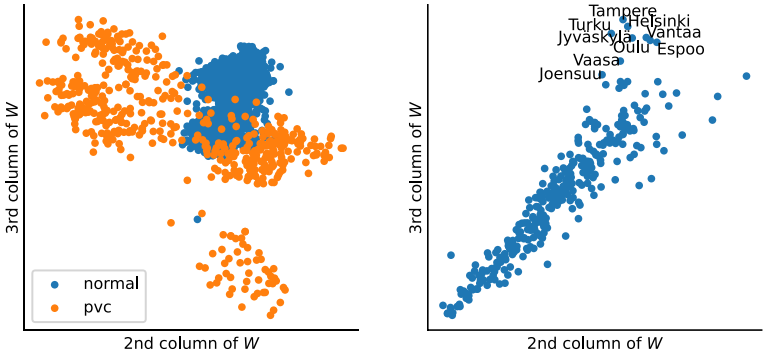
Our first observation is that MERGE typically finds a good solution that the iterative algorithms cannot improve. However, in *ECG* MERGE finds the worst decomposition, further improved by every iterative algorithm, with IEXACT yielding the smallest cost.

The error values are between 1 and -7.5, that is, produced errors are up to 8 times larger than the errors of SVD decomposition. This is expected, as our decomposition is significantly more restricted when compared to SVD. The largest error ratio of 7.5. was achieved for *Population*. However, in this case both decompositions are accurate: SVD achieves an L_2 error of 8×10^{-5} for an average row, while MERGE achieves an L_2 error of 6×10^{-5} for an average row. Recall that a single row is a histogram summing to 1.

Next, we demonstrate how error behaves as a function of k. A typical example is shown in Fig. 4a for *ECG* data. Here, the error drops quickly as k increases: an error for $k = 20$ is 2% of the error for $k = 1$.

Running time: In our experiments, IHILL was the fastest and GEXACT or MERGE were the slowest, mostly due to the $\mathcal{O}(m^2)$ term. We should stress that the running times for the iterative algorithms do not include the running time needed to compute the initial solution.

Figure 3 shows the running time of the algorithms as we increase either n or m. The behavious is as expected. The algorithm scale linearly with n, and GEXACT and MERGE scale quadratically with m while the remaining algorithms scaler linearly with m. The reported times in Fig. 3 are per iteration, the number of iterations required were between 1 and 3.



(a) *ECG* (b) *Population*

Fig. 5 Scatter plots based on W of *ECG* and *Population*. Here, GExact with $k = 3$ was used

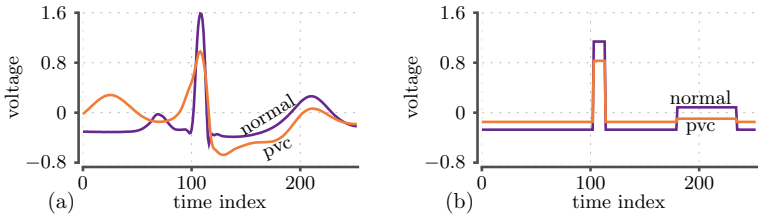


Fig. 6 **a** Averages of normal and abnormal ECG signals. **b** Averages of the reconstructed (*WS*) normal and abnormal ECG signals using GExact and $k = 3$

We see in Table 2 that GEst yields as good decomposition for *ECG* as GExact, despite solving the subproblem only approximately. On the other hand, GEst is faster than GExact.

Next, let us compare the running times of the two iterative algorithms. Recall that IExact is exponential w.r.t. k while IHill remains polynomial. This effect is shown in Fig. 4b with *ECG* data. Here IExact slows down considerably as k increases when compared to IHill (note that y-axis is logarithmic).

Similarly, GExact requires quadratic time w.r.t. m while GEst is quasilinear. This effect is shown in Fig. 4c when decomposing m first columns of *ECG* data: GExact slows down faster than GEst as the number of columns increases.

Examples: Finally, as an application we consider scatter plots of *ECG* and *Population* datasets, shown in Fig. 6. Here, we used GExact with $k = 3$ and used 2nd and 3rd columns of W as coordinates of each row in X. We did not plot the first column W as this is a bias term. In Fig. 5a we see that the normal beats and the abnormal beats yield different scatter plots, suggesting that the

decomposition was able to find discriminative features. These features are shown in Fig. 6b, compared to the data averages given in Fig. 6a.

The y-axis in Fig. 5b differentiates large university cities in Finland from the more rural municipalities.

9 Concluding remarks

We proposed a matrix decomposition problem that promotes having neighboring columns to have similar values. More specifically, we propose approximating X with WS, where S is a binary matrix such that 1 s on each row of S form a contiguous segment. We showed that the problem is inapproximable and proposed 5 algorithms whose computational complexity is summarized in Table 1.

Of the 5 algorithms, Merge produces good results that typically cannot be improved by the iterative algorithms. When improvement is possible the greedy algorithms GExact and GEst is a good choice: GExact when the number of columns is small and GEst when the number of columns is large.

We should point out that the consecutive ones requirement is strict. This is by design, since it allows us to summarize each row of S with just two numbers. An immediate extension would be to allow S to have ℓ segments of ones, per row; proposed algorithms can be extended to handle such a case. However, we can achieve a more flexible decomposition with the current approach by multiplying k with ℓ.

Future lines of work include additional regularization and relaxing the constraints for S, for example, requiring that S must be consistent with a PQ-tree (Booth and Lueker 1976).

Funding Open Access funding provided by University of Helsinki including Helsinki University Central Hospital.

Declarations

Conflict of interest Nikolaj Tatti is an action editor for DMKD and a member of editorial board for ECML PKDD JT 2023.

Research involving human participants and/or animals None

Informed consent Not applicable

References

Abboud A, Bringmann K, Hermelin D, Shabtay D (2022) Seth-based lower bounds for subset sum and bicriteria path. ACM Trans Algorithms 18(1):1–22

Bellman R (1961) On the approximation of curves by line segments using dynamic programming. Commun ACM 4(6):284–284

Booth KS, Lueker GS (1976) Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. J Comput Syst Sci 13(3):335–379

Chen Z, Cichocki A (2005) Nonnegative matrix factorization with temporal smoothness and/or spatial decorrelation constraints. Laboratory for Advanced Brain Signal Processing, RIKEN, Tech. Rep, 68

Cheng Y, Church GM (2000) Biclustering of expression data. In ISMB 8:93–103

Fomin FV, Golovach PA, Panolan F (2020) Parameterized low-rank binary matrix approximation. Data Min Knowl Disc 34:478–532

Geerts F, Goethals B, Mielikäinen T (2004) Tiling databases. In DS, pp 278–289

Gionis A, Mannila H (2003) Finding recurrent sources in sequences. In RECOMB, pp 123–130

Gionis A, Mannila H, Seppänen JK (2004) Geometric and combinatorial tiles in 0–1 data. In PKDD, pp 173–184

Goldberger Ary L, Amaral LAN, Glass L, Hausdorff JM, Ivanov PC, Mark RG, Mietus JE, Moody GB, Peng C-K, Eugene Stanley H (2000) Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals. Circulation 101(23):e215–e220

Guha S, Koudas N, Shim K (2001) Data-streams and histograms. In Proceedings of the thirty-third annual ACM symposium on Theory of computing, pp 471–475

Guha S, Koudas N, Shim K (2006) Approximation and streaming algorithms for histogram construction problems. TODS 31(1):396–438

Gupta A, Karypis G, Kumar V (1997) Highly scalable parallel algorithms for sparse matrix factorization. TPDS 8(5):502–520

Hartigan JA (1972) Direct clustering of a data matrix. JASA 67(337):123–129

Henelius A, Karlsson I, Papapetrou P, Ukkonen A, Puolamäki K (2016) Semigeometric tiling of event sequences. In ECML PKDD, pp 329–344

Impagliazzo R, Paturi R (2001) On the complexity of k-sat. J Comput Syst Sci 62(2):367–375

Kontonasios K-N, De Bie T (2010) An information-theoretic approach to finding informative noisy tiles in binary databases. In SDM, pp 153–164

Lam HT, Pei W, Prado A, Jeudy B, Fromont É (2014) Mining top-k largest tiles in a data stream. In ECML PKDD, pp 82–97. Springer

Madeira SC , Oliveira AL (2005) A linear time biclustering algorithm for time series gene expression data. In International workshop on algorithms in bioinformatics, pp 39–52. Springer

Miettinen P, Mielikäinen T, Gionis A, Das G, Mannila H (2008) The discrete basis problem. TKDE 20(10):1348–1362

Rallapalli S, Qiu L, Zhang Y, Chen Y-C (2010). Exploiting temporal stability and low-rank structure for localization in mobile networks. In MobiCon, pp 161–172

Roughan M, Zhang Y, Willinger W, Qiu L (2011) Spatio-temporal compressive sensing and internet traffic matrices (extended version). ToN 20(3):662–676

Tatti N (2019) Strongly polynomial efficient approximation scheme for segmentation. IPL 142:1–8

Tatti N, Miettinen P (2019) Boolean matrix factorization meets consecutive ones property. In SDM, pp 729–737

Tatti N, Vreeken J (2012) Discovering descriptive tile trees–by mining optimal geometric subtiles. In ECML PKDD 7523:9–24

Wang Y-X, Zhang Y-J (2012) Nonnegative matrix factorization: a comprehensive review. TKDE 25(6):1336–1353

Xiang Y, Jin R, Fuhry D, Dragan FF (2008) Succinct summarization of transactional databases: an overlapped hyperrectangle scheme. In KDD, pp 758–766

Xiong L, Chen X, Huang T-K, Schneider J, Carbonell JG (2010) Temporal collaborative filtering with bayesian probabilistic tensor factorization. In SDM, pp 211–222

Yu H-F, Rao N, Dhillon IS (2016) Temporal regularized matrix factorization for high-dimensional time series prediction. NIPS, vol 29

Zhang Y, Zha H, Chu C-H (2005) A time-series biclustering algorithm for revealing co-regulated genes. In ITCC 1:32–37