Information Sciences 719 (2025) 122489

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Summarizing Boolean and fuzzy tensors with sub-tensors

Victor Henrique Silva Ribeiro, Loïc Cerf,*

Federal University of Minas Gerais, Avenida Antonio Carlos 6627, Belo Horizonte, 31270-901, Minas Gerais, Brazil

ARTICLE INFO

Dataset link: https://dcc.ufmg.br/~lcerf/en/prototypes.html#nclusterbox

Keywords:
Disjunctive box cluster model
Fuzzy tensor
Ordinary least squares
Hill climbing
Elbow method

ABSTRACT

The disjunctive box cluster model summarizes an n-way Boolean tensor with some of its sub-tensors and their densities, *i.e.*, the arithmetic means of their values. Mirkin and Kramarenko proposed that easy-to-interpret regression model, for $n \in \{2,3\}$, and hill climbing to discover good sub-tensors, according to ordinary least squares. This article generalizes Mirkin and Kramarenko's work: n-way *fuzzy* tensors are summarized. They encode to what extent n-ary predicates are satisfied. The article also details significant performance improvements to the sequential execution, its parallelization, better starting points for hill climbing, a selection of the discovered sub-tensors, their ranking in order of contribution to the model, and the use of the elbow method to truncate the ordered list. In-depth experiments using synthetic and real-world tensors compare the proposed method, NclusterBox, to Mirkin and Kramarenko's and to the state-of-the-art algorithms for matrix factorization using the max (rather than +) operator and for Boolean tensor factorization. NclusterBox summarizes synthetic and real-world fuzzy tensors more efficiently and, most importantly, more accurately.

1. Introduction

Many datasets of interest are n-way fuzzy tensors: they quantify within the interval $[0,1]$ the satisfactions of n-ary predicates, where $n \in \mathbb{N}$ is at least 2. For instance, Fig. 1 shows a 3-way fuzzy tensor. Every value in it quantifies to what extent "the Twitter[1] user u was influential when she posted messages mentioning the soccer team t during the week w". The 3-tuple (u,t,w) indexes the cell containing the truth value, here a degree of influence. Given retweeted messages posted by 170 670 users and mentioning 29 teams over 12 weeks, the numbers of retweets may be used as a proxy to define the influence degrees (see Fig. 2) and the resulting fuzzy tensor is of dimension $170\,670 \times 29 \times 12$. This article proposes a method that aims to efficiently discover an accurate and easy-to-interpret summary of such a large fuzzy tensor.

On a quad-core processor, 1.78 second is enough to summarize the aforementioned tensor, with 452 409 nonzero influence degrees. Table 1 shows that summary. It is pattern-based. A *pattern* is a set of n-tuples resulting from the Cartesian product of subsets of each of the dimensions of the tensor ($X_{\text{users}} \times X_{\text{teams}} \times X_{\text{weeks}}$ in Table 1), hence the specification of a sub-tensor. Beside each pattern, its *density* is reported. It is simply the arithmetic mean of the values in the related sub-tensor. For instance, the second row in Table 1 stands for 279 Twitter users mentioning the `Atlético-MG` and the `Cruzeiro` soccer teams over all 12 weeks. Those messages were

* Corresponding author.
E-mail addresses: victor.henrique@dcc.ufmg.br (V.H. Silva Ribeiro), lcerf@dcc.ufmg.br (L. Cerf).
URL: https://dcc.ufmg.br/~lcerf (L. Cerf).

[1] A microblogging site, now called X. *Retweeting* is reblogging a message.

https://doi.org/10.1016/j.ins.2025.122489
Received 25 March 2025; Received in revised form 3 July 2025; Accepted 4 July 2025

Available online 11 July 2025

	Atlético-MG	Bahia	Cruzeiro
	week 4		
ecbahia	0	1	0
ESPNagora	0.1	1	1
iBahia	0	1	0
impedimento	0	0.2	0
News_Raposa	1	0	1
	week 5		
ecbahia	0	1	0
ESPNagora	0	1	0.6
iBahia	0	1	0.1
impedimento	1	1	0.6
News_Raposa	1	0.3	1

Fig. 1. Two patterns, {ESPNagora, impedimento, News_Raposa} × {Atlético-MG, Bahia, Cruzeiro} × {week 4, week 5} and {ecbahia, ESPNagora, iBahia, impedimento} × {Bahia} × {week 4, week 5} that overlap.

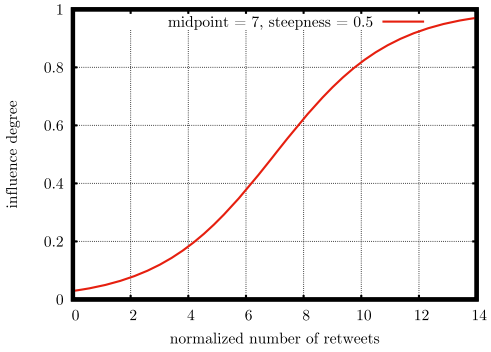

Fig. 2. Numbers of retweets are normalized so that summing those relating to any team gives the average number of retweets per team; the depicted logistic function turns them into influence degrees.

	Atlético-MG	Bahia	Cruzeiro
	week 4		
ecbahia	0.001	0.9	0.001
ESPNagora	0.6	0.9	0.6
iBahia	0.001	0.9	0.001
impedimento	0.6	0.9	0.6
News_Raposa	0.6	0.6	0.6
	week 5		
ecbahia	0.001	0.9	0.001
ESPNagora	0.6	0.9	0.6
iBahia	0.001	0.9	0.001
impedimento	0.6	0.9	0.6
News_Raposa	0.6	0.6	0.6

Fig. 3. Reconstruction of the fuzzy tensor in Fig. 1 from the summary composed of the two patterns (emphasized in Fig. 1) of densities $\frac{10.8}{18} = 0.6$ and $\frac{7.2}{8} = 0.9$, assuming the shown tensor is part of a larger tensor of density 0.001.

Table 1
NclusterBox's 7-pattern summary of the $170\,670 \times 29 \times 12$ fuzzy tensor.

$\lvert X_{\text{users}} \rvert$	X_{teams}	$\lvert X_{\text{weeks}} \rvert$	density
394	{Botafogo, Corinthians, Flamengo, Fluminense, Palmeiras, Ponte Preta, Santos, Vasco}	12	0.238
279	{Atlético-MG, Cruzeiro}	12	0.369
163	{Grêmio, Internacional}	12	0.458
170	{Bahia}	12	0.592
247	{Avaí, Figueirense}	11	0.349
133	{São Paulo}	12	0.597
148	{Coritiba}	12	0.552

rather influential: a 0.369 influence degree on average across the $279 \times 2 \times 12 = 6696$ combinations giving the 3-tuples of the pattern. In contrast, 0.001 is the average influence degree in the whole $170\,670 \times 29 \times 12$ tensor.

If Fig. 1 gives part of it, Fig. 3 shows how an analyst should envision that part summarized by the two patterns Fig. 1 emphasizes. At the intersection of several overlapping patterns of different densities (0.6 and 0.9, in the example), the analyst considers the greatest density (0.9, in the example), because the summary is a disjunction of patterns and max is the fuzzy logic operator corresponding to

OR in the Boolean logic. Outside the patterns of the summary, she considers the average degree across the whole tensor (0.001, in the example). That amounts to implicitly adding to the summary the pattern covering the whole tensor. The underlying regression model is named the *disjunctive box cluster model*. Mirkin and Kramarenko proposed it to summarize Boolean matrices and 3-way Boolean tensors [1]. They proposed as well hill climbing from several initial patterns to discover good patterns, according to ordinary least squares, to compose the summary.

After some basic definitions in Section 2 and related work in Section 3, Section 4 generalizes Mirkin and Kramarenko's algorithm to n-way fuzzy tensors. Section 5 details significant performance improvements, different starting points for hill climbing, a greedy selection and ordering of the discovered patterns, and suggests the elbow method to truncate the ordered list. Using synthetic and real-world tensors, Section 6 shows the proposed method more efficiently and more accurately summarizes them than the state-of-the-art algorithms for matrix factorization using the max operator and for Boolean tensor factorization. Finally, Section 7 concludes the article.

2. Basic definitions

Given n *dimensions* (*i.e.*, n finite sets, assumed disjoint without loss of generality) $D_1, \ldots, D_n$, a *fuzzy tensor* $\mathbf{T}$ maps any n-tuple $t \in \prod_{i=1}^{n} D_i$ (where $\prod$ denotes the Cartesian product) to a value $\mathbf{T}_t \in [0,1]$, called the *membership degree* of t. $\mathbf{1}$ denotes the tensor mapping every $t \in \prod_{i=1}^{n} D_i$ to 1. A set of n-tuples $X \subseteq \prod_{i=1}^{n} D_i$ is a *pattern* if and only if $\forall i \in \{1,\ldots,n\}$, $\exists X_i \subseteq D_i$ such that $X = \prod_{i=1}^{n} X_i$. The *area* of X is the number $|X|$ of n-tuples it includes. The arithmetic mean $\frac{\sum_{t \in X} \mathbf{T}_t}{|X|}$ of the membership degrees associated with those n-tuples is the *density* of X in $\mathbf{T}$. For example, the area of the blue-and-purple pattern in Fig. 1 is $3 \times 3 \times 2 = 18$ and its density is $\frac{10.8}{18} = 0.6$. The respective values for the red-and-purple pattern in Fig. 1 are $4 \times 1 \times 2 = 8$ and $\frac{7.2}{8} = 0.9$. Given two patterns X and Y, X is a *sub-pattern* of Y and Y is a *super-pattern* of X if and only if $X \subseteq Y$.

Given an n-tuple $t \in \prod_{i=1}^{n} D_i$ and $i \in \{1,\ldots,n\}$, t_i denotes the ith component of t, hence an element of D_i. Given $X = \prod_{i=1}^{n} X_i$ and, for some $i \in \{1,\ldots,n\}$, an element $e \in D_i$, the *slice* of X relating to e is $s(X,e) = (\prod_{j=1}^{i-1} X_j) \times \{e\} \times (\prod_{j=i+1}^{n} X_j)$. A slice is either *internal*, if $e \in X_i$, or *external*, if $e \in D_i \setminus X_i$. Removing an internal slice from X gives a sub-pattern of X. Adding an external slice to X gives a super-pattern of X.

For $n = 2$, the first operation applied to a pattern $X_1 \times X_2$ removes a row (if $e \in X_1$) or a column (if $e \in X_2$), whereas the second operation adds a row (if $e \in D_1 \setminus X_1$) or a column (if $e \in D_2 \setminus X_2$). For examples in the case $n = 3$, consider the red-and-purple pattern in Fig. 1, $X = \{\texttt{ecbahia}, \texttt{ESPNagora}, \texttt{iBahia}, \texttt{impedimento}\} \times \{\texttt{Bahia}\} \times \{\texttt{week 4}, \texttt{week 5}\}$. Its slice $s(X, \texttt{impedimento})$ is $\{\texttt{impedimento}\} \times \{\texttt{Bahia}\} \times \{\texttt{week 4}, \texttt{week 5}\}$. Since $\texttt{impedimento} \in \{\texttt{ecbahia}, \texttt{ESPNagora}, \texttt{iBahia}, \texttt{impedimento}\}$, that slice is internal. Removing it from X gives $\{\texttt{ecbahia}, \texttt{ESPNagora}, \texttt{iBahia}\} \times \{\texttt{Bahia}\} \times \{\texttt{week 4}, \texttt{week 5}\}$. In contrast, $s(X, \texttt{Cruzeiro})$ is an external slice of X, because $\texttt{Cruzeiro} \notin \{\texttt{Bahia}\}$. That slice is $\{\texttt{ecbahia}, \texttt{ESPNagora}, \texttt{iBahia}, \texttt{impedimento}\} \times \{\texttt{Cruzeiro}\} \times \{\texttt{week 4}, \texttt{week 5}\}$. Adding it to X gives $\{\texttt{ecbahia}, \texttt{ESPNagora}, \texttt{iBahia}, \texttt{impedimento}\} \times \{\texttt{Bahia}, \texttt{Cruzeiro}\} \times \{\texttt{week 4}, \texttt{week 5}\}$.

3. Related work

The method proposed in this article aims to summarize n-way fuzzy tensors. Fuzzy matrices (where $n = 2$) and n-way Boolean (aka 0/1) tensors (with $n \geq 3$) are special cases. An even more special case is that of 0/1 matrices, with an abundant literature that [2] surveys, unlike this section.

3.1. Complete algorithms

Given an n-way 0/1 tensor, several algorithms, *e.g.*, DATA-PEELER [3], list every sub-tensor that is *all-ones* and *closed*, *i.e.*, with no 0 and such that no strict super-pattern satisfies that property. Relaxations of that definition for $n = 3$ and methods to mine them are surveyed in [4]. That article recommends TriclusterBox (generalized in Section 4 of this article) despite its scalability issues (addressed in Subsections 5.1 and 5.2 of this article) and OAC-PRIME. Given a 3-way 0/1 tensor $\mathbf{T}$, OAC-PRIME enumerates every 3-tuple $t \in D_1 \times D_2 \times D_3$ such that $\mathbf{T}_t = 1$ and outputs the pattern $\{e \in D_1 \mid \mathbf{T}_{e,t_2,t_3} = 1\} \times \{e \in D_2 \mid \mathbf{T}_{t_1,e,t_3} = 1\} \times \{e \in D_3 \mid \mathbf{T}_{t_1,t_2,e} = 1\}$ if its density is high enough. Generalizing OAC-PRIME to n-way 0/1 tensors is straightforward [5]. Also, different criteria can trigger the involvement of an element in a pattern and the output of that pattern. In particular, given an n-way real-valued (possibly fuzzy) tensor $\mathbf{T}$ and $\delta \in \mathbb{R}_+$, the NOAC algorithm [6] enumerates every t such that $\mathbf{T}_t \neq 0$ and outputs the pattern $\prod_{i=1}^{n} \{e \in D_i \mid |\mathbf{T}_{t_1,\ldots,t_{i-1},e,t_{i+1},\ldots,t_n} - \mathbf{T}_t| < \delta\}$ if the variance of the associated real values is small enough. RMiner [7] generalizes the definition of a pattern to a multi-relational context: 0/1 tensors sharing dimensions. RMiner's patterns tolerate no 0. A-RMiner's patterns [8] do, by amalgamation of patterns that RMiner computes and that share a common core. Given prior beliefs, RMiner or A-RMiner deems a pattern interesting if the ratio between its information content, which grows with its area, and its description length, which grows with the number of elements it involves, is high.

`Multidupehack` [9] generalizes DATA-PEELER to fuzzy tensors: n bounds, $(\epsilon_1, \ldots, \epsilon_n) \in \mathbb{R}_+^n$, specify how much the sum of the membership degrees over any internal slice of a pattern can deviate from what would be obtained if that slice was all-ones. Formally, `multidupehack` outputs the pattern $X = \prod_{i=1}^{n} X_i$ only if $\forall i \in \{1,\ldots,n\}$, $\forall e \in X_i$, $\sum_{t \in X \text{ s.t. } t_i = e} (1 - \mathbf{T}_t) \leq \epsilon_i$. In that definition, and considering Zadeh's NOT operator, $1 - \mathbf{T}_t$ quantifies to what extent the n-tuple t is absent from the fuzzy n-ary relation $\mathbf{T}$ encodes. In this way, setting $\epsilon_i \in \mathbb{R}_+$ is specifying the maximal number of absent n-tuples in any slice of the pattern relating to an element

of X_i. Like DATA-PEELER, `multidupehack` can force the patterns to be closed and prune their search with additional constraints they must each satisfy, *e.g.*, the minimal area constraint. DCE [10] is another complete algorithm to mine patterns in fuzzy tensors. DCE's definition catches patterns that are not sub-patterns of any pattern planted in a synthetic tensor, even if little noise affects that tensor [9]. In contrast, the patterns `multidupehack` outputs do not go over the edges of the planted patterns, unless the tensor is very noisy. Moreover, `multidupehack` scales better. Yet, the number of patterns it outputs exponentially grows with the bounds $\epsilon_1, \ldots, \epsilon_n$, and so does the run time. As a consequence, given a reasonable time budget, `multidupehack` can only return many overlapping fragments of a large and noisy pattern to discover.

3.2. Heuristic algorithms

TRICLUSTER [11] mines patterns in 3-way 0/1 tensors. To reach larger and less-dense patterns, two patterns can be amalgamated, *i.e.*, merged into their smallest common super-pattern. TRICLUSTER does so if there is a large-enough ratio between the number of 3-tuples in either pattern and the number of 3-tuples in neither of them but in the super-pattern. That process ignores the membership degrees. ALPHA [12] defines the similarity between two patterns as the smallest average membership degree among all internal slices of the super-pattern. Using that similarity, ALPHA hierarchically amalgamates patterns and selects amalgamates that are both dense and far from their parents in the dendrogram. The difference between the similarity to the empty pattern $\emptyset$ and that to the parent pattern in the dendrogram formalizes the trade-off.

Several algorithms [13–20] factorize 0/1 tensors in an approximate way. As a consequence, the membership degrees in a fuzzy tensor must be rounded to 0/1 before applying any of those algorithms. The Boolean rank-r CANDECOMP/PARAFAC (CP) decomposition of an n-way 0/1 tensor $\mathbf{T}$ finds n 0/1 matrices $A^1, \ldots, A^n$ whose respective dimensions are $|D_1| \times r, \ldots, |D_n| \times r$. The search of those matrices aims to minimize the reconstruction error $\|\mathbf{T} - \max_{k=1}^{r} A^1_{:k} \otimes \cdots \otimes A^n_{:k}\|$, where $A^i_{:k}$ denotes the kth column of A^i, $\otimes$ is the outer product, $\|.\|$ is the Frobenius norm, and $\max_{k=1}^{r}$ returns a tensor where the membership degree of an n-tuple is 1 if it is 1 in at least one of r tensors, otherwise 0. In each of those r rank-1 tensors, the set of n-tuples with value 1 is a pattern. Minimizing the reconstruction error favors the discovery of r large and dense patterns. To seek a good Boolean rank-r CP decomposition of an n-way 0/1 tensor $\mathbf{T}$, GETF [20] exploits geometric properties of the pattern of density 1 maximizing the area. They lead to the identification of a *pattern fiber* (*i.e.*, a pattern $\prod_{i=1}^{n} X_i$ of area $|X_j|$ for some $j \in \{1, \ldots, n\}$). GETF turns it into a rank-1 tensor and subtracts it from $\mathbf{T}$ (considering $0 - 1 = 0$). That process is repeated r times or until adding the rank-1 tensor to the decomposition would not decrease the reconstruction error. To the best of our knowledge, GETF is state-of-the-art, in terms of reconstruction accuracy and of time complexity.

Technically, non-negative tensor decomposition applies to fuzzy tensors. With it, every rank-1 tensor assigns non-negative weights to all the elements of the n dimensions. The rank-1 tensors are summed in the standard way and cannot be interpreted as (fuzzy) patterns. Also, the values in the reconstructed tensor cannot be interpreted as membership degrees. Some may even exceed 1. The non-negative weights can be clustered (*e.g.*, with 2-means) [21] to end up with a Boolean decomposition, but many experiments (*e.g.*, in [14,16,17]) have shown that directly taking into consideration the Boolean nature of the tensor allows to more accurately summarize it.

The `Cancer` [22] and GRECOND$_L$ [23] matrix (not tensor) decomposition techniques substitute the standard addition for the max operator. Applying one of them to a fuzzy matrix, every resulting rank-1 matrix can be interpreted as a fuzzy pattern, where every element of every dimension is involved to some extent. `Cancer` can minimize the Frobenius norm and produces sparse factors, *i.e.*, many zero weights that ease the interpretation. GRECOND$_L$'s rank-1 matrices correspond to formal concepts [24]. By theory, they *undercover* the decomposed matrix, *i.e.*, any reconstructed *grade* (membership degree, if the scale is $[0, 1]$) is at most the input grade. By greedily expanding the formal concepts, GRECOND$_L$+ [25] allows overcovering and similar reconstruction errors are reached with smaller ranks. The available implementations of GRECOND$_L$ and GRECOND$_L$+ assume a finite scale, unlike $[0, 1]$, which is used in this article.

4. Generalizing BiclusterBox and TriclusterBox

The proposed method builds upon an existing algorithm [1] to summarize 0/1 matrices and 3-way 0/1 tensors. Its authors named it BiclusterBox (or BBox) when $n = 2$, TriclusterBox (or TBox) when $n = 3$. That section directly presents their generalization to n-way fuzzy tensors.

Summarizing an n-way fuzzy tensor $\mathbf{T}$ with a set of patterns $\mathcal{X}$ is seen as a regression problem. Under the convention $\max_{X \in \emptyset} \lambda_X = 0$, the *disjunctive box cluster model* models the membership degree of any n-tuple $t \in \prod_{i=1}^{n} D_i$:

$$\mathbf{T}_t = \lambda_0 + \max_{X \in \mathcal{X} \text{ s.t. } t \in X} \lambda_X + \epsilon \ , \tag{1}$$

where the intercept, λ_0, can be seen as a similarity shift, a parameter λ_X is associated with every pattern $X \in \mathcal{X}$, and ϵ is the error term. In that model, no parameter is fuzzy: λ_0 and every λ_X are standard real numbers and, as defined in Section 2, a pattern either includes a given n-tuple or not. Nevertheless, fuzzy logic plays a role in the definition of the disjunctive box cluster model. Indeed, it uses max, which is the Zadeh operator for the disjunction, *i.e.*, corresponding to OR in the Boolean logic.

Ordinary least squares are used, *i.e.*, the selection of $\mathcal{X}$ and the estimations of λ_0 and of every λ_X aim to minimize the residual sum of squares:

$$RSS_{\mathbf{T}}(\mathcal{X}) = \sum_{t \in \prod_{i=1}^{n} D_i} (\mathbf{T}_t - \hat{\mathbf{T}}_t)^2 \ . \tag{2}$$

Encoding, as explained in Subsection 3.2, the patterns of the summary $\mathcal{X}$ in the factors $A^1, \ldots, A^n$ of a rank-$|\mathcal{X}|$ Boolean CP decomposition, $RSS_{\mathbf{T}}(\mathcal{X})$ equals $\|\mathbf{T} - \hat{\lambda}_0 \mathbf{1} - \max_{k=1}^{|\mathcal{X}|} \hat{\lambda}_k A^1_{:k} \otimes \cdots \otimes A^n_{:k}\|^2$. Fixing in that expression the estimate of λ_0 to 0 and the estimate of every λ_X to 1, the square of the reconstruction error of the Boolean CP decomposition is obtained. In contrast, [1] proposes non-constant estimates of the parameters to further minimize $RSS_{\mathbf{T}}$. First, it sets the estimate of λ_0 to $\hat{\lambda}_0 = \frac{\sum_{t \in \prod_{i=1}^{n} D_i} \mathbf{T}_t}{|\prod_{i=1}^{n} D_i|}$. In this way, $RSS_{\mathbf{T}}(\emptyset) = \sum_{t \in \prod_{i=1}^{n} D_i} (\mathbf{T}_t - \lambda_0)^2$ is minimized. Then, any $\hat{\lambda}_X$ is $\frac{\sum_{t \in X} (\mathbf{T}_t - \hat{\lambda}_0)}{|X|}$, to minimize $RSS_{\mathbf{T}}(\{X\}) = \sum_{t \in X} (\mathbf{T}_t - \hat{\lambda}_0 - \lambda_X)^2 + \sum_{t \in (\prod_{i=1}^{n} D_i) \setminus X} (\mathbf{T}_t - \hat{\lambda}_0)^2$. Those estimates are easy to interpret. $\hat{\lambda}_0$ is the density of (the pattern $\prod_{i=1}^{n} D_i$ covering) the whole n-way fuzzy tensor and $\hat{\lambda}_0 + \hat{\lambda}_X = \frac{\sum_{t \in X} \mathbf{T}_t}{|X|}$ is the density of X.

BiclusterBox or TriclusterBox [1] builds the summary $\mathcal{X}$ of $\mathbf{T}$ one pattern at a time. Every such pattern is the ending point of a hill-climbing optimization. Its starting point involves one single element of D_1, which can be any of the n dimensions, their indexing being arbitrary. More precisely, X is an initial pattern of BiclusterBox or TriclusterBox if and only if:

$$\exists e \in D_1 \mid X = \{e\} \times \prod_{i=2}^{n} \left\{ f \in D_i \mid \exists t \in \prod_{j=1}^{n} D_j \text{ s.t. } \begin{cases} t_1 = e \\ t_i = f \\ \mathbf{T}_t > \hat{\lambda}_0 \end{cases} \right\} \ . \tag{3}$$

Every iteration of the hill-climbing optimization either adds/removes an external/internal slice to/from the current pattern X (see Section 2 for definitions and examples) or adds X to $\mathcal{X}$. In the latter case, a new optimization can start with an initial pattern involving a different element of D_1. At every iteration, the choice among the $1 + \sum_{i=1}^{n} |D_i|$ possible moves minimizes $RSS_{\mathbf{T}}(\{X\}) = \prod_{t \in \prod_{i=1}^{n} D_i} (\mathbf{T}_t - \hat{\lambda}_0)^2 - |X| \hat{\lambda}_X^2$, hence a maximization of the following function denoted by the letter g in [1] and in this article:

$$g(X) = |X| \hat{\lambda}_X^2 \ . \tag{4}$$

In the end, $\mathcal{X}$ contains at most $|D_1|$ patterns locally maximizing the area times the square of the density in $\mathbf{T} - \hat{\lambda}_0 \mathbf{1}$. That tensor is named the *shifted tensor* from now on.

Algorithms 1, 2, and 3 formalize BiclusterBox/TriclusterBox differently than [1], to ease the presentation, in Subsection 5.1, of performance improvements. In Algorithm 1, line 2 computes the shifted tensor and line 4 enumerates the initial patterns Equation (3) defines. Each of them is the starting point of a hill-climbing maximization of g. Line 5 launches it. As in Section 2, any pattern X in Algorithm 2 or 3 is a set of n-tuples, $\prod_{i=1}^{n} X_i$. The actual code uses n dynamic arrays: $X_1, \ldots, X_n$. Removing/adding a slice (lines 13/15 in Algorithm 2) then amounts to adding/removing an element to/from one of the n arrays, computing $|X|$ to multiplying their sizes, testing $s(X, e) \subseteq X$ (with $e \in D_i$) to checking if $e \in X_i$, and a single division, $\frac{|X|}{|X_i|}$, gives $|s(X, e)|$. Every iteration (lines 3–16 in Algorithm 2) of every hill-climbing maximization processes a pattern and considers the removal/addition of any single internal/external slice. Line 3 enumerates each of them. Line 4 sums the shifted membership degrees of its n-tuples so that `next_move` (Algorithm 3) can then compute the product of the area and the square of the density in the shifted tensor. The executions of the line 4 in Algorithm 2 dominate the run time of any iteration: discovering the next internal/external slice to remove/add from/to $X = \prod_{i=1}^{n} X_i$ takes $O(\sum_{j=1}^{n} |D_j| \prod_{k \in \{1,\ldots,n\} \setminus \{j\}} |X_k|)$ time.

Algorithm 1 BiclusterBox/TriclusterBox/NclusterBox.

Input: fuzzy tensor $\mathbf{T}$
Output: set of patterns summarizing $\mathbf{T}$
1: $\cup_{i=1}^{n} D_i \leftarrow \text{sort}(\cup_{i=1}^{n} D_i)$ using the partial order Equation (5) defines
2: $\mathbf{T} \leftarrow \mathbf{T} - \frac{\sum_{t \in \prod_{i=1}^{n} D_i} \mathbf{T}_t}{\prod_{i=1}^{n} |D_i|} \mathbf{1}$ ▷ similarity shift by $\hat{\lambda}_0$
3: $\mathcal{V} \leftarrow \emptyset$ ▷ $\mathcal{V}$ is a hash set of patterns
4: **for all** initial pattern X **do**
5: `hill-climb`$(\cup_{i=1}^{n} D_i, \mathbf{T}, X, \mathcal{V})$ ▷ $\mathcal{V}$ can grow
6: **end for**

In Algorithm 2, several sums associated with internal (respectively external) slices relating to elements of a same dimension may be minimal (respectively maximal) for that dimension, especially if the input tensor is 0/1. If the next move is to remove (respectively add) a slice of that dimension, they are tied if the only criterion is the maximization of g, defined in Equation (4). No other criterion is defined in [1]. However, it makes sense to break the tie by comparing the densities of the slices of $\prod_{j=1}^{n} D_j$ relating to the same elements. Indeed, the sparser (respectively denser), the more promising the removal (respectively addition) of the related slice of the pattern. To do so, in Algorithm 1, line 1 partially sorts $\cup_{i=1}^{n} D_i$ so that $\forall i \in \{1, \ldots, n\}$,

$$\forall (e, f) \in D_i \times D_i, \sum_{t \in s(\prod_{j=1}^{n} D_j, e)} \mathbf{T}_t < \sum_{t \in s(\prod_{j=1}^{n} D_j, f)} \mathbf{T}_t \Rightarrow e \text{ is before } f. \tag{5}$$

Algorithm 2 BiclusterBox/TriclusterBox's `hill-climb` procedure.

Input: $\cup_{i=1}^{n} D_i$, shifted tensor $\mathbf{T}$, pattern X, already output patterns $\mathcal{V}$
Output: a new pattern locally maximizing $g(X) = |X|\hat{\lambda}_X^2$ or none

$r \leftarrow \sum_{t \in X} \mathbf{T}_t$
while $X \notin \mathcal{V}$ **do**
  for all $e \in \cup_{i=1}^{n} D_i$ **do**
    $\delta_e \leftarrow \sum_{t \in s(X,e)} \mathbf{T}_t$ ▷ δ is an array
  end for
  $e_{\text{best}} \leftarrow$ `next_move`$(X, r, \cup_{i=1}^{n} D_i, \delta)$
  if $e_{\text{best}} =$ stop **then**
    output X ▷ X locally maximizes g
    $\mathcal{V} \leftarrow \mathcal{V} \cup \{X\}$ ▷ to never output X again
    return
  end if
  if $s(X, e_{\text{best}}) \subseteq X$ **then**
    $(X, r) \leftarrow (X \setminus s(X, e_{\text{best}}), r - \delta_{e_{\text{best}}})$ ▷ remove $s(X, e_{\text{best}})$ from X
  else
    $(X, r) \leftarrow (X \cup s(X, e_{\text{best}}), r + \delta_{e_{\text{best}}})$ ▷ add $s(X, e_{\text{best}})$ to X
  end if
end while

Algorithm 3 BiclusterBox/TriclusterBox/NclusterBox's `next_move`.

Input: pattern X, $r = \sum_{t \in X} \mathbf{T}_t$, $\cup_{i=1}^{n} D_i$, array δ such that $\forall e \in \cup_{i=1}^{n} D_i$, $\delta_e = \sum_{t \in s(X,e)} \mathbf{T}_t$
Return: either stop (if X locally maximizes g) or an element of $\cup_{i=1}^{n} D_i$ identifying the next internal/external slice to remove/add from/to X

1: $(e_{\text{best}}, g) \leftarrow (\text{stop}, \frac{r^2}{|X|})$ ▷ $\frac{r^2}{|X|} = |X|\hat{\lambda}_X^2$ since $\hat{\lambda}_X = \frac{r}{|X|}$
2: **for all** $e \in \cup_{i=1}^{n} D_i$ **do**
3: **if** $s(X,e) \subseteq X \wedge (r - \delta_e)^2 > (|X| - |s(X,e)|)g$ **then**
4: $(e_{\text{best}}, g) \leftarrow \left(e, \frac{(r-\delta_e)^2}{|X|-|s(X,e)|}\right)$ ▷ best move so far: remove $s(X,e)$ from X
5: **end if**
6: **if** $s(X,e) \not\subseteq X \wedge (r + \delta_e)^2 \geq (|X| + |s(X,e)|)g$ **then**
7: $(e_{\text{best}}, g) \leftarrow \left(e, \frac{(r+\delta_e)^2}{|X|+|s(X,e)|}\right)$ ▷ best move so far: add $s(X,e)$ to X
8: **end if**
9: **end for**
10: **return** e_{best}

Then, $>$ and $\geq$ in lines 3 and 6 of Algorithm 3 effectively implement the tie break. The hash set of the already output patterns $\mathcal{V}$ is absent from [1] too. In Algorithms 1 and 2, $\mathcal{V}$ prevents the output of duplicate patterns.

5. NclusterBox

5.1. Faster sequential execution

Storing in $\mathcal{V}$ not only the already output patterns but all the patterns visited so far decreases the run time. The output remains the same. Indeed, whenever Algorithm 2 reaches a pattern that a previous hill-climbing maximization visited, the same time-consuming moves are then taken until the same local maximum of g, defined in Equation (4). Algorithm 4 replaces Algorithm 2 and implements the improvement: the line 6 in Algorithm 4 is executed whatever the next move, unlike the line 9 in Algorithm 2.

At any iteration (lines 3–16 in Algorithm 2, corresponding to lines 6–22 in Algorithm 4) removing/adding an internal/external slice, every slice of the resulting pattern is either unchanged (if it relates to an element of the same dimension as that of the element defining the removed/added slice) or only loses/gains n-tuples of the removed/added slice. In Algorithm 4, lines 14–16 and 19–21 accordingly update the sums of the shifted membership degrees of the n-tuples belonging to every slice. For $n = 2$, after removing/adding a row (respectively column) in line 13/18, every column (respectively row) loses/gains one single 2-tuple whose shifted membership degree is subtracted/added to the associated sum. In general, if the removed/added slice of $X = \prod_{i=1}^{n} X_i$ relates to an element of the dimension D_i, updating the sum associated with an element of a different dimension D_j requires summing the shifted membership degrees of $\prod_{k \in \{1,\dots,n\} \setminus \{i,j\}} |X_k|$ n-tuples: those at the intersection of the slices relating to the two elements. That is why the time to update all the sums is $O(\sum_{j \in \{1,\dots,n\} \setminus \{i\}} |D_j| \prod_{k \in \{1,\dots,n\} \setminus \{i,j\}} |X_k|)$. It dominates the time to execute the iteration. In Section 4, the analog iteration of Algorithm 2 was found to require $O(\sum_{j=1}^{n} |D_j| \prod_{k \in \{1,\dots,n\} \setminus \{j\}} |X_k|)$ time. As a consequence, an iteration of Algorithm 4 can be said as fast as an iteration of Algorithm 2 processing a pattern in a tensor with the ith dimension missing (for example a matrix instead of a 3-way tensor), the dimension of the element relating to the removed/added slice.

5.2. Multithreading

The workload is embarrassingly parallel. Indeed, the executions of Algorithm 4 are almost independent from each other. The only common resources are $\mathcal{V}$, which contains the patterns visited so far, and the output stream. The proportion of the run time spent using $\mathcal{V}$, in lines 5 and 6, and the output stream, in line 9, is tiny. Two binary semaphores enforce exclusive accesses to those resources

Algorithm 4 NclusterBox's `hill-climb` procedure.

Input: $\cup_{i=1}^{n} D_i$, shifted tensor $\mathbf{T}$, pattern X, visited patterns $\mathcal{V}$
Output: a new pattern locally maximizing $g(X) = |X|\hat{\lambda}_X^2$ or none
$r \leftarrow \sum_{t \in X} \mathbf{T}_t$
for all $e \in \cup_{i=1}^{n} D_i$ **do**
  $\delta_e \leftarrow \sum_{t \in s(X,e)} \mathbf{T}_t$ ▷ δ is an array
end for
while $X \notin \mathcal{V}$ **do**
  $\mathcal{V} \leftarrow \mathcal{V} \cup \{X\}$ ▷ to abort whenever X is visited again
  $e_{\text{best}} \leftarrow$ `next_move`$(X, r, \cup_{i=1}^{n} D_i, \delta)$
  if $e_{\text{best}} =$ stop **then**
    output X ▷ X locally maximizes g
    return
  end if
  if $s(X, e_{\text{best}}) \subseteq X$ **then**
    $(X, r) \leftarrow (X \setminus s(X, e_{\text{best}}), r - \delta_{e_{\text{best}}})$ ▷ remove $s(X, e_{\text{best}})$ from X
    for all $e \in \cup_{i=1}^{n} \{D_i \mid e_{\text{best}} \notin D_i\}$ **do**
      $\delta_e \leftarrow \delta_e - \sum_{t \in s(s(X, e_{\text{best}}), e)} \mathbf{T}_t$
    end for
  else
    $(X, r) \leftarrow (X \cup S(X, e_{\text{best}}), r + \delta_{e_{\text{best}}})$ ▷ add $s(X, e_{\text{best}})$ to X
    for all $e \in \cup_{i=1}^{n} \{D_i \mid e_{\text{best}} \notin D_i\}$ **do**
      $\delta_e \leftarrow \delta_e + \sum_{t \in s(s(X, e_{\text{best}}), e)} \mathbf{T}_t$
    end for
  end if
end while

and a third semaphore to the initial patterns. No other coordination is needed to have a user-defined number of threads concurrently execute the lines 4 to 6 in Algorithm 1.

5.3. Single cells as initial patterns

With Equation (3) defining the initial patterns, there can be at most $|D_1|$ patterns in the summary. That may be too few to have Equation (1) accurately model the fuzzy tensor. A greater issue is that those initial patterns are large, often whole slices of $\prod_{i=1}^{n} D_i$, if $n \geq 3$. Starting from such a pattern, the hill climbing usually ends up with a pattern that is large and of low density. It locally maximizes g, of course, but denser sub-patterns that are not reached may provide greater local maxima. Even if they do not, together they usually make a more accurate summary, according to Equation (2), than the super-pattern of them all alone. Moreover, removing many slices of a large initial pattern takes much time (see end of Subsection 5.1).

To solve those issues, BiclusterBox/TriclusterBox's initial patterns are replaced by the densest m patterns of area 1, where $m \in \mathbb{N}$ is user-defined. After sorting in descending order the membership degrees of the n-way fuzzy tensor $\mathbf{T}$, if the mth membership degree equals the $(m+1)$th, what typically happens if $\mathbf{T}$ is 0/1, a subset of the tied patterns with that density is drawn. Every subset allowing to complete m initial patterns is equally likely. Formally, X is an initial pattern of NclusterBox only if:

$$\exists t \in \prod_{i=1}^{n} D_i \mid \begin{cases} X = \{t\} \\ |\{t' \in \prod_{i=1}^{n} D_i \mid \mathbf{T}_{t'} > \mathbf{T}_t\}| < m \end{cases} . \tag{6}$$

In the experiments of Section 6, using those initial patterns rather than those of BiclusterBox/TriclusterBox, defined in Equation (3), greatly improves the final summaries of the considered synthetic and real-world fuzzy tensors. Supplementary material presents results obtained with an intermediary definition, used in a preliminary work [26]. Its initial patterns are sub-fibers of the n-way fuzzy tensor, hence larger than single cells but smaller than BiclusterBox/TriclusterBox's if $n \geq 3$ (and the same if $n = 2$). The intermediary patterns lead to intermediary results in the same experiments. They therefore suggest that smaller initial patterns are better.

Nevertheless, no theoretical result proves that. The hill climbing procedure only guarantees that, whatever its starting point, it will end up with a pattern locally maximizing g, defined in Equation (4). Since the pattern area is literally a factor in g, Algorithm 3 usually prefers adding an external slice over removing an internal slice. Intuitively, the discovery of a small pattern locally maximizing g therefore looks more likely from a sub-pattern of it: no internal slice needs to be removed. However, the highest local maxima may correspond to large patterns. Their discovery would be favored if Algorithm 3 would consider the addition/removal of several slices at once. A previous work [27] has proposed such a loosened neighborhood. The resulting run times were unpredictable and often prohibitive. As a consequence, within a same time budget, far fewer searches of local maxima could be launched.

Even ignoring the computational costs, a more effective maximization of g is not necessarily desirable. First of all, given a fuzzy tensor $\mathbf{T}$, the sought summary of it needs not be composed of one single pattern. Having several patterns usually allows to further improve the fit, *i.e.*, to further minimize $RSS_{\mathbf{T}}$, defined in Equation (2). Repetitively finding the pattern X_{best} globally maximizing g is not only useless but also harmful for the accuracy of the summary: diversity is needed at the output of Algorithm 1. In fact, having X_{best} in the summary may even be sub-optimal. Several smaller and denser patterns overlapping with X_{best}, in particular sub-patterns of it, may altogether make a more accurate summary of the sub-tensor that X_{best} defines. To possibly select in the summary those overlapping patterns, relating to lower local maxima of g, they must be discovered in the first place.

5.4. Ranking and selecting patterns

Using Equation (6), Algorithm 1 outputs at most m patterns, maybe too many for an easy interpretation. Moreover, the estimated fuzzy tensor $\hat{\mathbf{T}}$ they provide via Equation (1) may overfit $\mathbf{T}$. Ranking the patterns helps the analyst, who may interpret only the first ones. Given any set of candidate patterns, Algorithm 5 returns an ordered subset, the summary $\mathcal{X}$, by greedily minimizing the Bayesian Information Criterion (BIC) [28]. The BIC is a popular trade-off between minimizing of the residual sum of squares, in Equation (2), and minimizing the number of parameters, $1 + |\mathcal{X}|$. According to the BIC, a pattern Y is worth adding to $\mathcal{X}$ if $\frac{RSS_{\mathbf{T}}(\mathcal{X}\cup\{Y\})}{RSS_{\mathbf{T}}(\mathcal{X})} < |\prod_{i=1}^{n} D_i|^{-\frac{1}{|\prod_{i=1}^{n} D_i|}}$, where $|\prod_{i=1}^{n} D_i|$ appears because it is the number of observations.

Algorithm 5 Selection of an ordered summary, by stepwise regression.

Input: fuzzy tensor $\mathbf{T}$, set of candidate patterns $\mathcal{Y}$
Return: ordered subset of $\mathcal{Y}$ (truncating gives a coarser summary)

1: $\mathcal{X} \leftarrow \emptyset$ ▷ an ordered collection
2: **while** $\mathcal{Y} \neq \emptyset$ **do**
3: $Y \leftarrow \arg\min_{Y\in\mathcal{Y}} RSS_{\mathbf{T}}(\mathcal{X} \cup \{Y\})$ ▷ best addition to $\mathcal{X}$
4: **if** $RSS_{\mathbf{T}}(\mathcal{X} \cup \{Y\}) \geq |\prod_{i=1}^{n} D_i|^{-\frac{1}{|\prod_{i=1}^{n} D_i|}} \times RSS_{\mathbf{T}}(\mathcal{X})$ **then**
5: **return** $\mathcal{X}$ ▷ no pattern of $\mathcal{Y}$ decreases the BIC when added to $\mathcal{X}$
6: **end if**
7: **append** Y to $\mathcal{X}$
8: $\mathcal{Y} \leftarrow \mathcal{Y} \setminus \{Y\}$
9: $X \leftarrow \arg\min_{X\in\mathcal{X}} RSS_{\mathbf{T}}(\mathcal{X} \setminus \{X\})$ ▷ best removal from $\mathcal{X}$
10: **if** $RSS_{\mathbf{T}}(\mathcal{X}) \geq |\prod_{i=1}^{n} D_i|^{-\frac{1}{|\prod_{i=1}^{n} D_i|}} \times RSS_{\mathbf{T}}(\mathcal{X} \setminus \{X\})$ **then**
11: $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{Z \in \mathcal{X} \mid Z \text{ after } X \text{ in } \mathcal{X}\}$ ▷ X is out, forever
12: **truncate** $\mathcal{X}$ keeping the patterns before X
13: **end if**
14: **end while**
15: **return** $\mathcal{X}$ ▷ no candidate pattern left

At every iteration, given the initially empty (line 1) collection of patterns $\mathcal{X}$ shortlisted so far, line 3 identifies the candidate pattern Y whose addition to $\mathcal{X}$ would improve the fit the most. If the BIC would decrease, the test in line 4 fails, line 7 appends Y to $\mathcal{X}$, line 8 removes it from the candidates, and line 9 identifies, among the previously added patterns, the pattern X that contributes the least to the fit. Line 10 decides if the presence of X worsens the current summary (greater BIC). If so, the selection essentially starts over without that pattern. However, the execution would be identical until right before X was appended. That is why line 11 reinserts among the candidates only the patterns that were appended after X and line 12 keeps in the summary the patterns before X, a *truncation*. Either line 15 returns the final summary, if no candidate pattern is left, or line 5, if adding any remaining one to the summary would not decrease the BIC.

Using Equation (2) for every evaluation of $RSS_{\mathbf{T}}$ in Algorithm 5 would require much time. The implemented computation of $RSS_{\mathbf{T}}(\mathcal{X})$, in lines 4 and 10, is actually incremental. It takes $O(\prod_{i=1}^{n} |D_i|)$ time to get the residual sum of squares of the empty summary, $RSS_{\mathbf{T}}(\emptyset) = \sum_{t\in\prod_{i=1}^{n} D_i} (\mathbf{T}_t - \hat{\lambda}_0)^2$. Whenever line 7 shortlists a pattern Y, $RSS_{\mathbf{T}}(\mathcal{X} \cup \{Y\}) - RSS_{\mathbf{T}}(\mathcal{X})$, whose efficient computation will be presented soon, is added to $RSS_{\mathbf{T}}(\mathcal{X})$ to obtain $RSS_{\mathbf{T}}(\mathcal{X} \cup \{Y\})$, which is appended to an array. Whenever line 12 truncates the summary, it truncates that array at the same position. In this way, the array always explicitly defines the function mapping $k \in \{1, \ldots, |\mathcal{X}|\}$ to $RSS_{\mathbf{T}}$(set of the first k patterns of $\mathcal{X}$). Visually analyzing its curve at the end of the execution may reveal an elbow: every pattern after improves the fit much less than every pattern before. That has a practical interest: it makes sense to stop interpreting the ordered summary at the elbow.

For all $t \in \prod_{i=1}^{n} D_i$, the actual implementation of Algorithm 5 stores in main memory not only $\mathbf{T}_t$ but also keeps updated, during the selection of $\mathcal{X}$, the estimation Equation (1) gives, $\hat{\mathbf{T}}_t = \hat{\lambda}_0 + \max_{X\in\mathcal{X} \text{ s.t. } t\in X} \hat{\lambda}_X$, and the estimation Equation (1) would give if, among the patterns of $\mathcal{X}$ that include t, (one of) the densest would be removed, $\hat{\mathbf{T}}'_t = \hat{\lambda}_0 + \max_{X\in\mathcal{X}\setminus\{Y\} \text{ s.t. } t\in X} \hat{\lambda}_X$, where $Y \in \arg\max_{Z\in\mathcal{X} \text{ s.t. } t\in Z} \hat{\lambda}_Z$. To do so, $\hat{\mathbf{T}}$ and $\hat{\mathbf{T}}'$ are initialized to $\hat{\lambda}_0\mathbf{1}$. Whenever line 7 shortlists a pattern Y, every $t \in Y$ is enumerated and $\hat{\lambda}_0 + \hat{\lambda}_Y$ is compared to $\hat{\mathbf{T}}_t$ and $\hat{\mathbf{T}}'_t$. Both are updated if $\hat{\mathbf{T}}_t < \hat{\lambda}_0 + \hat{\lambda}_Y$. Only $\hat{\mathbf{T}}'$ is updated if $\hat{\mathbf{T}}'_t < \hat{\lambda}_0 + \hat{\lambda}_Y \leq \hat{\mathbf{T}}_t$. None of them is updated if $\hat{\lambda}_0 + \hat{\lambda}_Y \leq \hat{\mathbf{T}}'_t$.

In line 3 of Algorithm 5, the pattern $\arg\min_{Y\in\mathcal{Y}} RSS_{\mathbf{T}}(\mathcal{X}\cup\{Y\})$ is $\arg\min_{Y\in\mathcal{Y}} RSS_{\mathbf{T}}(\mathcal{X}\cup\{Y\}) - RSS_{\mathbf{T}}(\mathcal{X})$ too, because $RSS_{\mathbf{T}}(\mathcal{X})$ is constant. For any pattern Y and any n-tuple $t \notin Y$, Equation (1) provides the same estimate $\hat{\mathbf{T}}_t$ whether the summary is $\mathcal{X}$ or $\mathcal{X}\cup\{Y\}$. As a consequence, summing over all $t \in Y$ such that $\hat{\lambda}_0 + \hat{\lambda}_Y > \hat{\mathbf{T}}_t$ every term $(\mathbf{T}_t - \hat{\lambda}_0 - \hat{\lambda}_Y)^2 - (\mathbf{T}_t - \hat{\mathbf{T}}_t)^2$ gives the difference $RSS_{\mathbf{T}}(\mathcal{X}\cup\{Y\}) - RSS_{\mathbf{T}}(\mathcal{X})$. In this way, line 3 not only identifies the best candidate to add to the summary, but it does so computing $RSS_{\mathbf{T}}(\mathcal{X}\cup\{Y\}) - RSS_{\mathbf{T}}(\mathcal{X})$. Adding that difference to $RSS_{\mathbf{T}}(\mathcal{X})$ gives $RSS_{\mathbf{T}}(\mathcal{X}\cup\{Y\})$, that line 4 tests. Similar observations hold for lines 9 and 10. Here, to compute the variation of $RSS_{\mathbf{T}}$ that the removal of X from $\mathcal{X}$ would entail, $\hat{\mathbf{T}}'$ is needed too. Indeed, $RSS_{\mathbf{T}}(\mathcal{X}) - RSS_{\mathbf{T}}(\mathcal{X}\setminus\{X\})$ is the sum of every $(\mathbf{T}_t - \hat{\mathbf{T}}'_t)^2 - (\mathbf{T}_t - \hat{\mathbf{T}}_t)^2$ for all $t \in X$ such that $\hat{\mathbf{T}}_t = \hat{\lambda}_0 + \hat{\lambda}_X$.

Storing and incrementally updating the variation of $RSS_{\mathbf{T}}$ that is entailed by the addition of any candidate (respectively shortlisted) pattern $P = \prod_{i=1}^{n} P_i$ to $\mathcal{X}$ (respectively $\mathcal{X}\setminus\{P\}$) further lowers the time complexity. That variation is initially $-|P|\hat{\lambda}_P^2$. When line 7 appends $Y = \prod_{i=1}^{n} Y_i$, only the n-tuples of $P \cap Y$ are enumerated to update the variation, summing terms as specified earlier. Computing $P \cap Y = \prod_{i=1}^{n} P_i \cap Y_i$ requires $O(\sum_{i=1}^{n} |P_i| + |Y_i|)$ time, P_i and Y_i being sorted, for all $i \in \{1, \ldots, n\}$. In this way, an iteration

of Algorithm 5 that removes no pattern from $\mathcal{X}$ takes $O(\sum_{P\in\mathcal{P}}(|P\cap Y|+\sum_{i=1}^{n}|P_i|+|Y_i|))$ time, where $\mathcal{P}$ is the current set of patterns, the candidates and those of $\mathcal{X}$, and Y is the last pattern of $\mathcal{X}$.

For an iteration that removes from the summary $\mathcal{X}$ the pattern X defined in line 9, the time requirement increases by that needed to reset all $\hat{\mathbf{T}}_t$ and $\hat{\mathbf{T}}'_t$ for $t\in\cup_{Y\in\mathcal{X}}Y$ (any other $\hat{\mathbf{T}}_t$ or $\hat{\mathbf{T}}'_t$ is still $\hat{\lambda}_0$) and all the variations of $RSS_{\mathbf{T}}$ associated with all the patterns (but X) to the values they had right before X was shortlisted. That is done from their initial values, taking the same steps as those previously taken to append to $\mathcal{X}$ the patterns before X. In the worst case, the whole execution of Algorithm 5 processing the set of input candidates $\mathcal{Y}$ eliminates $O(|\mathcal{Y}|)$ of them and every elimination happens after the last remaining candidate is shortlisted. That is why Algorithm 5 requires at most $O(\prod_{i=1}^{n}|D_i|+|\mathcal{Y}|\sum_{(Y,P)\in\mathcal{Y}^2}(|P\cap Y|+\sum_{i=1}^{n}|P_i|+|Y_i|))$ time.

The ability to eliminate patterns particularly matters if the candidates include a large pattern and denser sub-patterns of it. Algorithm 5 may shortlist the larger pattern before (some of) its sub-patterns, because it explains more memberships degrees. Nevertheless, after shortlisting (more) sub-patterns, to better model the membership degrees associated with their n-tuples, the contribution of the larger pattern to the fit may be detrimental. Indeed, the presence of that pattern in the summary only makes a difference outside its now-shortlisted denser sub-patterns. There, the membership degrees are smaller than the density of the large pattern. Its elimination not only improves the fit but also eases the selection of more of its sub-patterns. They complement those that were previously shortlisted and, together, accurately summarize the sub-tensor related to the eliminated pattern.

6. Experimental study

The proposed method is named NclusterBox. It is Algorithm 1 enumerating the initial patterns Equation (6) defines and calling Algorithm 4. When Equation (3) and Algorithm 2 are used instead, the method is named BiclusterBox if $n=2$, or TriclusterBox if $n=3$, *i.e.*, the names that Mirkin and Kramarenko coined [1] apply to their work generalized to *fuzzy* data. Given the same inputs, Algorithms 2 and 4 always output the same pattern, or lack thereof. As a consequence, in the following experiments, the choice of initial patterns *entirely* explains the differences between the summaries computed with NclusterBox and those obtained with BiclusterBox/TriclusterBox.

When $n=2$, Cancer [22] is a competitor. Like NclusterBox, Cancer uses the max operator to aggregate the rank-1 matrices of its decomposition and applies to fuzzy matrices. In particular, the matrix reconstructed from its factorization is always fuzzy, whereas values can exceed 1 if a standard non-negative factorization technique is used. When compared to four such techniques in applications where it makes sense to use max rather than +, Cancer has demonstrated advantages [22]. Every rank-1 matrix of its decomposition assigns a weight to each element (row or column). Such a rank-1 matrix is harder to interpret than a pattern (defined from two sets, of rows and columns) and its density. Nevertheless, Cancer produces sparse factors. That eases the conversion of its rank-1 matrices to (possibly overlapping) patterns. In the following experiments, it is necessary for comparison purpose: to get patterns, an element is kept if and only if its weight exceeds the density of the matrix.

When $n\geq 3$, besides TriclusterBox, the closest works deal with the Boolean CP decomposition. GETF [20] is state-of-the-art. It serves as a competitor. Since it can only process 0/1 tensors, the input membership degrees are rounded. The rank-1 tensors of the Boolean CP decomposition are directly mapped to patterns, which may overlap. The related model of the 0/1 tensor $\mathbf{T}$ is very simple: $\hat{\mathbf{T}}_t=0$ if t is in no output pattern, $\hat{\mathbf{T}}_t=1$ otherwise. That simplicity is not really an advantage over the disjunctive box cluster model. Indeed, the estimates of its additional parameters are the densities of the patterns, hence useful pieces of information that are easy to interpret. To assess how detrimental the rounding, NclusterBox is given the 0/1 tensor too. In that case, the algorithm is named NclusterBox01. Its initial patterns are those Equation (6) defines from the *fuzzy* tensor, hence exactly those NclusterBox processes. In this way, the rounding *entirely* explains the differences between the summaries computed with NclusterBox and those obtained with NclusterBox01.

Whatever the competitor, its default settings are used and Algorithm 5 post-processes the output patterns, seen as a disjunctive box cluster model of the *fuzzy* tensor, even if GETF or NclusterBox01 provides the patterns. Unless the effect of the hyper-parameter is studied, $m=1000$ in Equation (6). Eight threads are used (see Subsection 5.2) because the quad-core processor executing NclusterBox and its competitors supports hyper-threading. It is an Intel Xeon E3-1241 v3 processor with a base frequency of 3.5 GHz and 8 MB of cache. 16 GB of RAM are available. A GNU/Linux system runs the experiments. The scripts to reproduce them all and the C++ implementation of NclusterBox are published under the terms of the GNU GPLv3+ license on https://dcc.ufmg.br/~lcerf/en/prototypes.html#nclusterbox.

6.1. Results and insights on synthetic tensors

With equal probability of appearance, ten 25×25 (respectively $5\times5\times5$) sub-tensors of a 1000×1000 (respectively $100\times100\times100$) zero tensor are randomly drawn. They may overlap. Their values are set to 1. The resulting tensor is 0/1. Considering every 0 or 1 as the output of a Bernoulli variable with parameter p, the probability of a 1, *inverse transform sampling* adds noise to the tensor. The posterior distribution of p is the beta distribution of parameters α and β. They have a meaningful interpretation: after observing, for a same tuple, $\alpha-1$ membership degrees at 1 and $\beta-1$ at 0, the beta distribution is the distribution of p. The number of incorrect observations ($\beta-1$ when noising a 0, $\alpha-1$ when noising a 1) is here fixed to 0 and the number of correct observations ($\alpha-1$ when noising a 0, $\beta-1$ when noising a 1) controls the level of noise. Fig. 4 shows the inverses of cumulative beta distributions. They are used to noise a 0 (left) or a 1 (right): an abscissa within the interval $[0,1)$ is randomly drawn (any abscissa is equally likely) and the related ordinate, on the curve providing the desired level of noise, is the noisy version of the 0 or the 1. For GETF and NclusterBox01, every value is then rounded and the process from the initial 0/1 tensor equates to switching a 0 for a 1 or a 1 for a 0 with probability

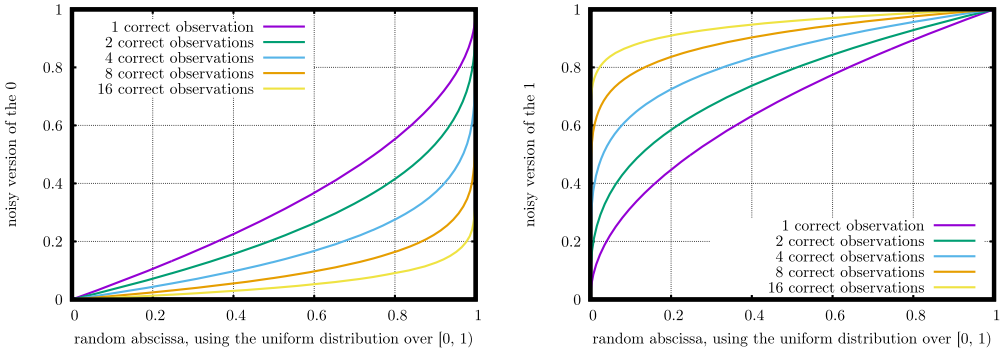

Fig. 4. Inverses of cumulative beta distributions, to noise a 0 (left) or a 1 (right).

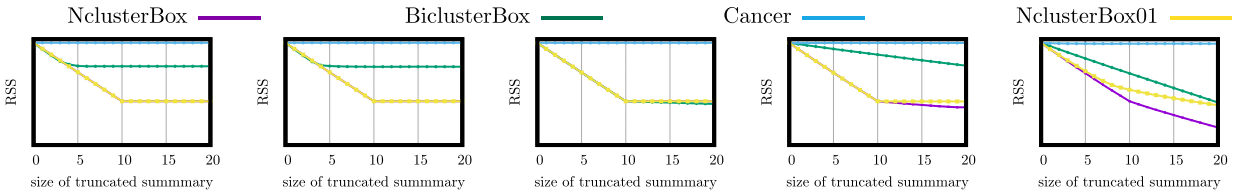

Fig. 5. Residual sums of squares of the truncated summaries of 1000×1000 matrices, which are increasingly noisy from left to right: 16, 8, 4, 2, and 1 correct observation(s).

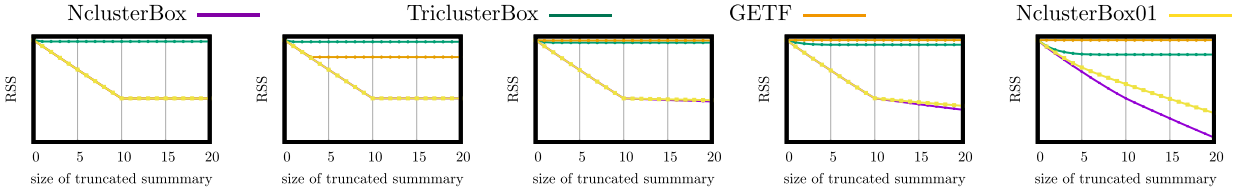

Fig. 6. Residual sums of squares of the truncated summaries of $100 \times 100 \times 100$ tensors, which are increasingly noisy from left to right: 16, 8, 4, 2, and 1 correct observation(s).

25% (1 correct observation), 12.5% (2 correct observations), 3.125% (4 correct observations), 0.1953125% (8 correct observations) or ≈0.0007629% (16 correct observations). Indeed, in the considered settings, those are the probabilities that inverse transform sampling turns a 0 (respectively 1) to a value greater (respectively less) than 0.5. For each of the five levels of noise, thirty fuzzy matrices/tensors are synthesized. Every reported result is an average over them.

Figs. 5 and 6 plot, in function of $k \in \{0, \dots, 20\}$, the residual sums of squares, defined in Equation (2), of the summaries composed of the first k patterns that Algorithm 5 returns, given the patterns at the output of a competitor. The main reason to look at such a curve is to detect an elbow, where the interpretation of the related summary could stop, every pattern after the elbow improving the fit much less than every pattern before. For that detection, the range visualized on the y-axis needs to be appropriate. Here, it is defined such that the residual sums of squares of the summary with no pattern, $RSS_{\mathbf{T}}(\emptyset) = \sum_{t \in \prod_{i=1}^{n} D_i} (\mathbf{T}_t - \hat{\lambda}_0)^2$, and that of the 10-pattern summary obtained with NclusterBox are always at the same graphic positions.

The curves related to NclusterBox clearly bend at $k = 10$. In the least noisy settings, they coincide with those of NclusterBox01, which starts from the same initial patterns. In those settings, even given rounded membership degrees, NclusterBox outputs only ten patterns, the ten planted patterns, and the residual sums of squares are constant from $k = 10$ onward. In the noisier settings, NclusterBox finds more patterns locally maximizing the function g, defined in Equation (4). Algorithm 5 selects more than ten patterns, by stepwise regression. Such techniques being prone to overfitting [29], it was expected. Anyway, ranking in descending order of contribution to the fit remains valuable and, here, the elbow method correctly truncates the summary. In the noisiest settings, the elbows start to disappear, especially in Fig. 6. They completely disappear if the membership degrees are rounded.

Cancer and GETF require as an input the maximal rank of the factorization. They are given the number of planted patterns, 10. Despite that unfair advantage, the residual sums of squares Cancer and GETF achieve are close to that of the summary with no pattern, unless GETF decomposes the least-noisy tensor. In that case, GETF perfectly recovers the ten planted patterns: its curve coincides with NclusterBox(01)'s. Anyway, the patterns to be discovered being known, it makes little sense to assess the competitors by interpreting residual sums of squares. First of all, although Cancer and GETF aim to minimize Equation (2), they estimate every membership degree differently than NclusterBox, which relies on the disjunctive ox cluster model, in Equation (1). As the beginning of this section remembered, Cancer uses a more complex model and GETF a simpler one. Yet, they apply to the recovery of patterns planted in tensors where noise is then added. Also, the goodness of fit does not tell the whole story. The residual sums of squares achieved with

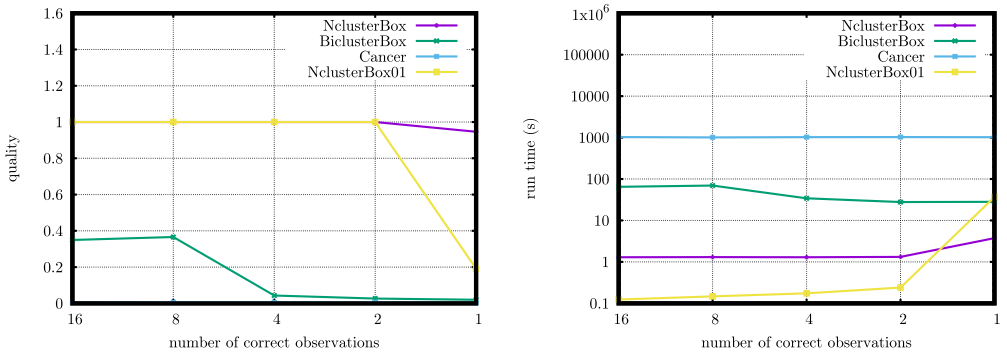

Fig. 7. Quality of the top-10 patterns (left) and run time (right) including the selection and ranking (Algorithm 5); the 1000×1000 matrices are increasingly noisy from left to right on the x-axis.

the patterns BiclusterBox outputs make that point: in the middle sub-figure (corresponding to 4 correct observations) of Fig. 5, the curves for BiclusterBox and NclusterBox are close, but the related summaries have very different qualities.

Given $\mathcal{P}$, the planted patterns, and $\mathcal{X}$, the patterns discovered in the related noisy matrix/tensor and that Algorithm 5 ranks in the top-10, the Jaccard index $\frac{|P \cap X|}{|P \cup X|}$ measures the similarity between a planted pattern $P \in \mathcal{P}$ and a pattern $X \in \mathcal{X}$. The *quality* of $\mathcal{X}$ is defined as:

$$\frac{\left|\bigcup_{P \in \mathcal{P}} \left(P \cap \arg\max_{X \in \mathcal{X}} \frac{|P \cap X|}{|P \cup X|} \right)\right|}{\left|\bigcup_{P \in \mathcal{P}} P \cup \bigcup_{X \in \mathcal{X}} X\right|} . \tag{7}$$

The quality is a proportion of tuples. At the numerator, the true positive tuples belong to both a planted pattern and to the most similar pattern of $\mathcal{X}$. At the denominator, the tuples are the union of all the patterns of $\mathcal{P}$ or $\mathcal{X}$. In this way, the quality measure penalizes both the absence from $\mathcal{X}$ of patterns that are similar to the planted ones and the presence in $\mathcal{X}$ of patterns including tuples out of the planted patterns.

Fig. 7 shows, in function of the level of noise, the quality of the discovered summaries of the fuzzy matrices (left) and the times to discover them (right). Cancer takes approximately 17 minutes per decomposition. For the least noisy matrices (16 correct observations), NclusterBox is 793 times faster. For the noisiest matrices (1 correct observation), it is 273 times faster. Between those two settings, NclusterBox requires 2.45 additional seconds. Nevertheless, NclusterBox itself (Algorithms 1 and 4) is not to be blamed for the increase. Algorithm 5 is mainly responsible for it. With 16 correct observations, NclusterBox's hill-climbing optimizations lead to only the ten patterns to be discovered and Algorithm 5 selects them all in approximately one millisecond. In contrast, with 1 correct observation, NclusterBox finds more than 100 patterns locally maximizing g. Algorithm 5 first shortlists a super-pattern of several planted patterns. However, after selecting sub-patterns, which are similar to the planted patterns, lines 9 and 10 deem the first-inserted pattern as detrimental to the summary and lines 11 and 12 remove that pattern. The selection starts over with another large pattern, which is later identified as detrimental, and so on until the elimination of all the too-large patterns. In the end, the few seconds that stepwise regression takes are worth it: despite the high level of noise, the quality reached with NclusterBox is 0.947. It is 1 when less noise (2 correct observations or more) affects the matrix: the first ten patterns returned to the analyst are exactly the ten planted patterns. The ability to eliminate previously shortlisted patterns clearly makes a significant difference. A preliminary work [26] lacking that capability could not list the ten 100×100 patterns planted in a 1000×1000 matrix, even if no noise is added.

Rounding the fuzzy matrix before running NclusterBox, the quality remains 1 if the probability of a wrong rounded membership degree is 12.5% or less (see first paragraph of this subsection). At 25%, every local maximization of g tends to end up with a different pattern. The subsequent selection, with Algorithm 5, requires much time and the quality of the first ten patterns is only 0.191. In that context, it can be concluded that directly handling fuzzy data is valuable. BiclusterBox and Cancer do so. Yet, whatever the level of noise in these experiments, they never discover the planted patterns. Cancer always outputs very large patterns with low densities. The resulting quality is at most 0.007 and the related curve in Fig. 7 is hidden behind the x-axis. With 8 or more correct observations, BiclusterBox mostly discovers super-patterns of unions of a few planted patterns. As with the patterns NclusterBox outputs, Algorithm 5 eliminates the largest ones. Unfortunately, the remaining patterns that are returned are still too large, because no smaller pattern, closer to a planted pattern, is candidate. With 2 or fewer correct observations, BiclusterBox's hill climbing always ends up with a sub-pattern of the initial pattern (that Equation (3) defines), involving one single element of D_1 and too many elements of D_2. With 4 correct observations, BiclusterBox outputs a mix of the two types of patterns that have just been described. None of them matches up with the planted patterns and the resulting quality is always below 0.366.

Analogously to Fig. 7, Fig. 8 plots the results obtained with the 3-way synthetic tensors. The planted patterns are smaller than those in the previous matrices. Their individual area is $5 \times 5 \times 5 = 125$. That reduction is necessary to reach the limits of NclusterBox. In the noisiest setting (1 correct observation), the first ten patterns Algorithm 5 selects from the output of NclusterBox have a 0.877 quality. In presence of less noise (2 correct observations or more), the 10-pattern summary is perfect. The above interpretation of the performances of NclusterBox, NclusterBox01, and BiclusterBox in Fig. 7 essentially applies when the 3-way synthetic tensors replace the synthetic matrices. The main difference is that, while BiclusterBox is renamed TriclusterBox, it only discovers super-patterns of

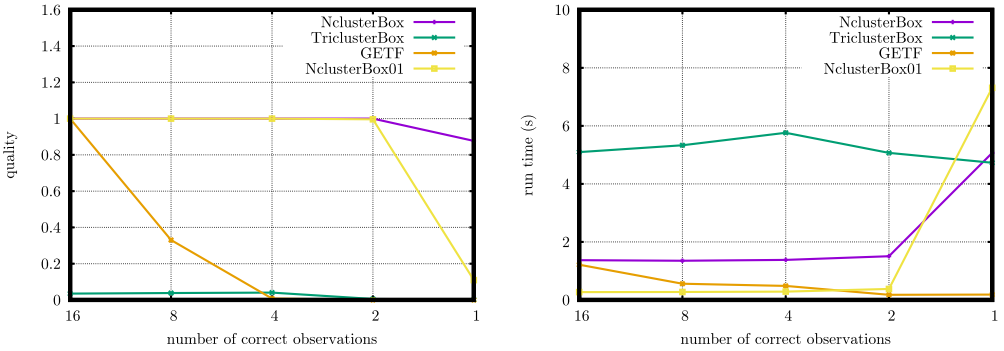

Fig. 8. Quality of the top-10 patterns (left) and run time (right) including the selection and ranking (Algorithm 5); the $100 \times 100 \times 100$ tensors are increasingly noisy from left to right on the x-axis.

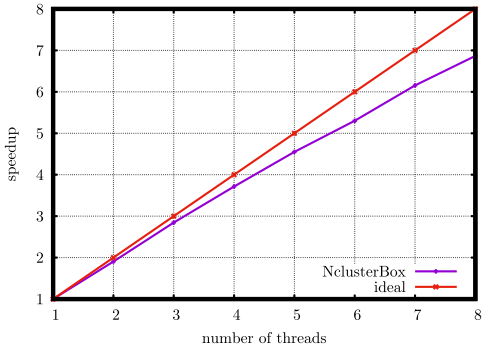

Fig. 9. Speedup on the $170\,670 \times 29 \times 12$ influence tensor; the 0.91 second to sequentially read the tensor from the disk and shift it (lines 1 and 2 in Algorithm 1) is ignored.

unions of several planted patterns. The resulting quality stays below 0.041, whatever the level of noise. The new competitor, GETF, is fast, but usually slower than NclusterBox01, which processes the same rounded tensors. GETF recovers the patterns planted in a 0/1 tensor where every membership degree has a ≈0.0007629% probability of being wrong. If that probability becomes 0.1953125%, the quality drops to 0.331. That low level of noise is indeed enough to have GETF return fewer than the ten patterns it is asked to compute. When the probability is 12.5% or more, GETF outputs nothing.

6.2. Results and insights on real-world tensors

6.2.1. Influence of Twitter users, a $170\,670 \times 29 \times 12$ fuzzy tensor

Section 1 introduced the application considered here: the analysis of the influence of 170 670 Twitter users (dimension D_{users}) retweeted at least once after mentioning a Brazilian soccer team among 29 (dimension D_{teams}) over 12 weeks (dimension D_{weeks}), from 13 January 2014 to 6 April 2014. NclusterBox takes 98 MB of RAM and 1.78 second to find 34 patterns locally maximizing g. In 0.002 second, Algorithm 5 selects 30 of them. Disabling the performance improvements that Subsection 5.1 details, but still using eight threads, the run time is multiplied by 62. Multithreading, as explained in Subsection 5.2, is effective too. Fig. 9 depicts the speedup. It is close to perfect up to four threads, corresponding to the number of physical cores, and still improving up to eight threads, thanks to hyper-threading. Using eight threads is 6.87 times faster than using one.

For all $k \in \{0, \ldots, 30\}$, Fig. 10 shows the residual sums of squares of the summaries containing the first k patterns Algorithm 5 returns when it is given the output of a competitor. The curve for NclusterBox clearly bends at $k = 7$ patterns. Table 1 lists them. Their densities, in the last column, contrast with $\hat{\lambda}_0 = 0.001$. The first pattern is large: it includes 37 824 3-tuples. It involves the four teams based in Rio de Janeiro and four major teams in São Paulo. Most of the commentators focus on those two Brazilian states. Every subsequent pattern deals with teams in one single state: Minas Gerais (2nd pattern), Rio Grande do Sul (3rd), Bahia (4th), Santa Catarina (5th), São Paulo (6th), and Paraná (7th). That makes sense: who posts influential messages about her favorite team is often retweeted as well when she posts about its rival(s), in the same state. Every pattern involves sport journalists and supporters of the teams appearing in the pattern. The seven patterns do not overlap. In particular, the `São Paulo` Futebol Clube appears in the 6th pattern but not in the 1st one. Beyond the 7th pattern, patterns start to overlap. For instance, the 9th pattern supports the discovery of 178 users who were highly influential (a 0.573 influence degree, on average) when they wrote about the `Atlético-MG` soccer team over all twelve weeks. The 2nd pattern, of density 0.369, involves 167 of those users. Equation (1) invites the analyst reaching the 9th pattern to adopt the significantly greater density, 0.573, at the intersection of the two patterns.

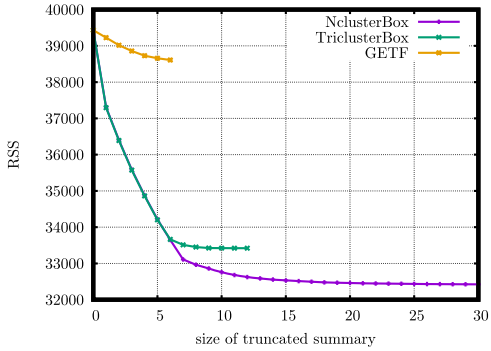

Fig. 10. Residual sums of squares of the truncated summaries of the $170\,670 \times 29 \times 12$ influence tensor.

Table 2
NclusterBox01's 7-pattern summary of the $170\,670 \times 29 \times 12$ influence tensor.

$\lvert X_{\text{users}}\rvert$	X_{teams}	$\lvert X_{\text{weeks}}\rvert$	density
276	{Botafogo, Corinthians, Flamengo, Fluminense, Palmeiras, Ponte Preta, Santos, Vasco}	12	0.281
200	{Atlético-MG, Cruzeiro}	12	0.427
124	{Grêmio, Internacional}	12	0.520
117	{Bahia}	12	0.701
188	{Avaí, Figueirense}	11	0.397
91	{São Paulo}	12	0.707
121	{Coritiba}	12	0.604

Table 3
TriclusterBox's 6-pattern summary of the $170\,670 \times 29 \times 12$ influence tensor.

$\lvert X_{\text{users}}\rvert$	X_{teams}	$\lvert X_{\text{weeks}}\rvert$	density
430	{Botafogo, Corinthians, Flamengo, Fluminense, Palmeiras, Ponte Preta, Santos, São Paulo, Vasco}	12	0.216
287	{Atlético-MG, Cruzeiro}	12	0.364
163	{Grêmio, Internacional}	12	0.458
170	{Bahia}	12	0.592
247	{Avaí, Figueirense}	11	0.349
148	{Coritiba}	12	0.552

In Fig. 10, no curve relates to NclusterBox01 because it would be visually indistinguishable from that of NclusterBox. Nevertheless, with NclusterBox01, the truncated summaries are consistently worse. At the elbow, the residual sum of squares is 33 279, whereas it is 33 110 using NclusterBox. Table 2 lists NclusterBox01's top-7 patterns. They are those in Table 1 but with always proper subsets of users, among the most influential, hence the higher densities (in the *fuzzy* tensor): rounding prevents the discovery of less influential users, who make the summary obtained with NclusterBox more accurate. In fact, rounding directly discards 156 663 users, 91.8%. Indeed, the influence degree of every 3-tuple involving one of them is below 0.5, hence 0 after rounding. Processing the resulting $14\,007 \times 29 \times 12$ Boolean tensor only takes NclusterBox01 0.25 second and 4 MB of RAM.

Setting $D_1 = D_{\text{teams}}$ in Equation (3) defines $|D_{\text{teams}}| = 29$ initial patterns. They are large. They involve all twelve weeks, except for one pattern, and, on average, 9542 users. That is mainly why TriclusterBox requires much time: 2 hours and 18 minutes, 4655 times the 1.78 second NclusterBox takes. TriclusterBox finds 21 patterns. Algorithm 5 keeps 12 of them. In Fig. 10, the curve for TriclusterBox bends at 6 patterns. Table 3 lists them. They involve more users and are less dense than those in Table 1. They are otherwise similar and so are the fits up to the 6th pattern. The major difference is that the 1st pattern in Table 3 "combines" the 1st and 6th patterns in Table 1: the São Paulo Futebol Clube is together with teams based in the same state or in Rio de Janeiro. NclusterBox actually outputs that combined pattern minus four user slices and Algorithm 5 selects it first. However, Algorithm 5 later selects the 6th pattern in Table 1 and, even later, the 1st pattern there. At the latter iteration, lines 9 and 10 figure out the pattern selected first is detrimental and the selection starts over without it. Fig. 10 clearly shows that having in the summary the two patterns with very different densities, 0.238 and 0.597, better fits the tensor that what can be achieved from the output of TriclusterBox, with no pattern involving only the teams appearing in the first pattern in Table 1 or only the São Paulo team.

Setting $D_1 = D_{\text{users}}$ in Equation (3) gives TriclusterBox $|D_{\text{users}}| = 170\,670$ opportunities to discover such patterns. It does not: it takes 2 days, 22 hours and 17 minutes (142 000 times NclusterBox's 1.78 second) and the first six patterns that Algorithm 5 returns are those in Table 3 minus 4 and 8 user slices removed from the 1st and 2nd patterns. Finally, with $D_1 = D_{\text{weeks}}$, TriclusterBox requires 2 hours and 42 minutes and Algorithm 5 keeps only one pattern. Involving 678 users, 14 teams and all 12 weeks, it is too large and its density, 0.133, is too low to be of much interest.

Asked to compute seven patterns in the rounded tensor, GETF takes 89.7 seconds and 4.81 GB of RAM, respectively 354 and 1216 times more than NclusterBox01, which processes the same rounded tensor, or 49 and 50 times more than NclusterBox. Considering

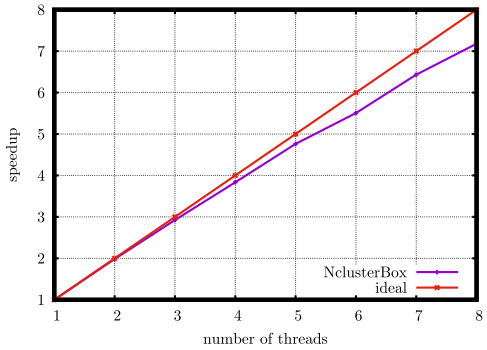

Fig. 11. Speedup on the $170\,670 \times 29$ influence matrix; the time to sequentially read the matrix from the disk and shift it (lines 1 and 2 in Algorithm 1) is ignored.

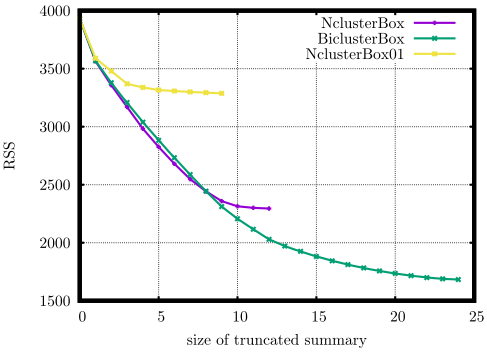

Fig. 12. Residual sums of squares of the truncated summaries of the $170\,670 \times 29$ influence matrix.

the seven patterns as patterns in the fuzzy tensor, Algorithm 5 selects and ranks six of them. Fig. 10 shows the residual sums of squares of the truncated summaries are much worse using GETF than using NclusterBox. Every pattern GETF outputs has a high density (at least 0.740) and a small area (at most 378). Yet, some of the patterns are hard to interpret. For instance, the pattern ranked first involves only six users and five weeks but eleven teams based in seven different states.

6.2.2. *Influence of Twitter users, a* $170\,670 \times 29$ *fuzzy matrix*

A $170\,670 \times 29$ matrix is built from the normalized numbers of retweets per pair (user, team), over all twelve weeks. The logistic function of midpoint $7 \times 12 = 84$ and steepness $\frac{0.5}{12} \approx 0.042$ turns those numbers into influence degrees. Its curve is exactly that of Fig. 2, but with the x-axis ranging from 0 to 168 (instead of 14). NclusterBox only requires 78 MB of RAM and, partly thanks to multithreading (speedup in Fig. 11), 1.25 second to discover 15 patterns in the resulting fuzzy matrix of density $\hat{\lambda}_0 = 0.003$. Algorithm 5 selects and ranks 12 of them. Disabling the improvements detailed in Subsection 5.1, the run time becomes 30.5 seconds, 24 times more.

Noise seems to affect the fuzzy matrix more than it affected the previous 3-way fuzzy tensor. First of all, no sharp elbow is visible in Fig. 12, which is analogous to Fig. 10. Then, among the top-10 patterns obtained with NclusterBox and listed in Table 4, the 2nd, 4th and 9th patterns each involve teams based in different states. For the 2nd pattern, the reason is a few messages mentioning "Atlético", actually referring to `Atlético-MG`, but registered as referring to `Atlético-GO`. Since only 1619 retweets (out of 3 167 511) relate to the latter team, a single retweet becomes, after normalization, a 0.334 influence degree. The normalization explains as well the high densities of the last four patterns, where only minor teams appear. That said, all the patterns in Table 4 tend to be denser than in Table 1. They deal with more users too. That is because, in the fuzzy matrix, a single much retweeted message is enough for a user to be considered influential about the mentioned team, whereas, in the 3-way fuzzy tensor, she must be influential over (most of) the weeks the pattern involves.

NclusterBox01 takes 0.11 second and 4 MB of RAM to find 9 patterns in the rounded matrix. As with the 3-way tensor, they leave out the less influential users. For instance, the patterns ranked first and second are those in Table 4 but with subsets of their users (153 and 78 of them, respectively) and higher densities (0.501 and 0.690, respectively). The related fit is much worse, as Fig. 12 shows.

On the contrary, a rather accurate summary of the fuzzy matrix can be composed from patterns BiclusterBox discovers. Setting $D_1 = D_{\text{teams}}$ in Equation (3), the entailed 29 hill-climbing searches take 3 minutes and 51 seconds, 185 times more than NclusterBox's 1.25 second. They lead to 25 patterns. Algorithm 5 selects and ranks 24 of them. Table 5 lists the top-10. Its 2nd, 5th, 9th and 10th patterns are in Table 4 too. According to Fig. 12, if an analyst has the time to interpret eight or fewer patterns, she had better read the more accurate summary obtained with NclusterBox. For instance, the 1st pattern in Table 5 is not only subjectively worse than the first pattern in Table 4, for involving the `Bahia` soccer team together with teams based in Rio de Janeiro and São Paulo. It is also worse with

Table 4
NclusterBox's 10-pattern summary of the $170\,670 \times 29$ influence matrix.

$\lvert X_{\text{users}} \rvert$	X_{teams}	density
265	{Botafogo, Corinthians, Flamengo, Fluminense Palmeiras, Ponte Preta, Portuguesa, Santos, Vasco}	0.395
455	{Atlético-MG, Atlético-GO, Cruzeiro}	0.391
267	{Avaí, Figueirense}	0.596
336	{Bahia, Vitória}	0.531
190	{Grêmio, Internacional}	0.646
197	{Coritiba}	0.861
135	{Guarani}	0.994
106	{América-MG}	1.000
219	{Náutico, Sport}	0.441
60	{Ceará}	0.864

Table 5
BiclusterBox's 10-pattern summary of the $170\,670 \times 29$ influence matrix.

$\lvert X_{\text{users}} \rvert$	X_{teams}	density
258	{Botafogo, Corinthians, Flamengo, Fluminense, Palmeiras, Ponte Preta, Portuguesa, Santos, São Paulo, Vasco, Bahia}	0.361
267	{Avaí, Figueirense}	0.596
319	{Atlético-MG, Cruzeiro}	0.520
431	{Atlético-GO}	0.629
190	{Grêmio, Internacional}	0.646
232	{Bahia}	0.867
205	{Coritiba}	0.844
242	{Vitória}	0.772
135	{Guarani}	0.994
106	{América-MG}	1.000

regard to the maximized function, in Equation (4): at the first pattern in Table 5, evaluating g gives $2838 \times (0.361 - 0.003)^2 \approx 365$, which is smaller than $2385 \times (0.395 - 0.003)^2 \approx 368$, obtained with the first pattern in Table 4. Starting with the large patterns Equation (3) defines, hill climbing does not reach the highest local maxima. However, it reaches more distinct local maxima than starting with the single-cell patterns of Equation (6). Given a larger set of larger candidate patterns, Algorithm 5 can select a summary that, through Equation (1), less often uses $\hat{\lambda}_0$ as an estimate for the influence degree. That is why, if nine or more patterns can be interpreted, a more accurate summary can be composed from the output of BiclusterBox than from that of NclusterBox. It is worth pointing out that the implementation of NclusterBox can actually use any initial patterns. If it processes those Equation (3) defines, it obviously outputs the same patterns as BiclusterBox. It does so more efficiently though, in 5.51 seconds. That is 42 times faster than without the performance improvements that Subsections 5.1 and 5.2 detail. In practice, Cancer cannot factorize the $170\,670 \times 29$ influence matrix: of 139 iterations, the first eight take 5 days, 10 hours and 37 minutes.

6.2.3. Contacts in a primary school, a $242 \times 242 \times 20$ Boolean tensor

On Thursday, 1 October 2009, and on Friday, 2 October 2009, Stehlé et al. collected in a primary school face-to-face contacts between 242 individuals (dimension $D_{\text{individuals}}$), 232 children and 10 teachers [30]. Each day, they recorded timestamped contacts over ten consecutive one-hour intervals, from 8 am–9 am to 5 pm–6 pm, hence 20 intervals (dimension D_{hours}) over the two days. For each 3-tuple $(i_1, i_2, h) \in D_{\text{individuals}} \times D_{\text{individuals}} \times D_{\text{hours}}$, the membership degree $\mathbf{T}_{i_1,i_2,h}$ is 1 if i_1 and i_2 were in contact during h or if $i_1 = i_2$. It is 0 otherwise. In this way, $\mathbf{T}$ is a $242 \times 242 \times 20$ Boolean tensor. In average over thirty executions, it takes NclusterBox 0.097 second and 44 MB of RAM to summarize $\mathbf{T}$. As Subsection 5.3 explains, every execution uses a different set of 1000 initial patterns, subset of the 57 542 single-cell patterns of density 1. This subsection will show the thirty resulting summaries are similar to the deterministic summary obtained with $m = 57\,542$ in Equation (6). Despite that high value for m, NclusterBox requires only 1.13 second. It discovers 78 patterns. Algorithm 5 selects and ranks 35 of them, in 0.006 second.

For all $k \in \{0, \ldots, 35\}$, Fig. 13 shows the residual sums of squares of the top-k-pattern summaries. The curve clearly bends at $k = 13$ patterns. They are in Table 6. The symmetry of the face-to-face contacts in the tensor, *i.e.*, $\forall (i_1, i_2, h) \in D_{\text{individuals}} \times D_{\text{individuals}} \times D_{\text{hours}}$, $\mathbf{T}_{i_1,i_2,h} = \mathbf{T}_{i_2,i_1,h}$, is insufficient to have any $X_1 \times X_2 \times X_3 \in 2^{D_{\text{individuals}}} \times 2^{D_{\text{individuals}}} \times 2^{D_{\text{hours}}}$ at the output of NclusterBox necessarily satisfy $X_1 = X_2$. Yet, in this experiment, that property holds, hence the unique set of individuals Table 6 shows for each pattern. There are ten classes in the school: 1A, 1B, 2A, 2B, 3A, 3B, 4A, 4B, 5A, and 5B. A clear mapping exists between them and the first nine and 11th patterns in Table 6. Indeed, each of those patterns exclusively deals with individuals assigned to a single class. Its teacher always appears and so do, on average, 95% of its children participating in the study. Taking into account only those participating during both days, that proportion becomes 98%. Each of the ten patterns also involves between 8 and 15 hours, in both mornings and both afternoons, except for the classes 4A and 4B associated with no hour during Friday afternoon.[2] The four lunch-break hours, midday-2 pm on both days,[3] never appear. On the contrary, they are the only hours appearing in the other three patterns in Table 6.

[2] "This is probably due to different class schedules (*e.g.*, a class being absent during half a day because of sport activities)." [30].

[3] "The school day runs from 8.30 am to 4.30 pm, with a lunch break from 12 pm to 2 pm, and two breaks of 20–25 min around 10.30 am and 3.30 pm." [30].

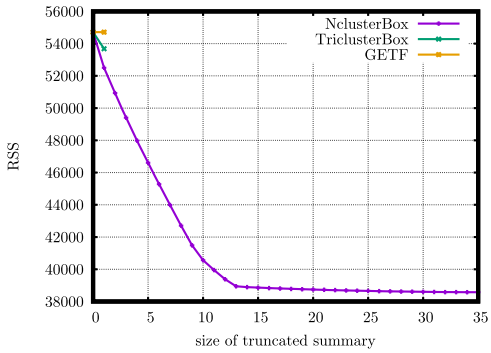

Fig. 13. Residual sums of squares of the truncated summaries of the $242 \times 242 \times 20$ contact tensor.

Table 6
NclusterBox's 13-pattern summary of the $242 \times 242 \times 20$ contact tensor using 57 542 initial patterns and *recovery* rates using 1000 initial patterns.

$X_{\text{individuals}}$	X_{hours}	density	recovery
25/25 in 1B, 1 teacher	Thu.: 9 am–midday, 2 pm–5 pm Fri.: 8 am–midday, 2 pm–5 pm	0.551	1
21/22 in 3B, 1 teacher	Thu.: 9 am–midday, 2 pm–5 pm Fri.: 8 am–midday, 2 pm–5 pm	0.548	1
23/23 in 3A, 1 teacher	Thu.: 10 am–midday, 3 pm–5 pm Fri.: 8 am–midday, 2 pm–5 pm	0.540	1
21/22 in 5A, 1 teacher	Thu.: 8 am–9 am, 10 am–midday, 2 pm–6 pm Fri.: 8 am–midday, 2 pm–6 pm	0.492	1
24/26 in 2B, 1 teacher	Thu.: 9 am–midday, 2 pm–4 pm Fri.: 9 am–midday, 2 pm–3 pm	0.543	1
22/23 in 2A, 1 teacher	Thu.: 8 am–9 am, 10 am–midday, 4 pm–5 pm Fri.: 8 am–midday, 2 pm–5 pm	0.527	1
21/21 in 4A, 1 teacher	Thu.: 9 am–midday, 2 pm–5 pm Fri.: 8 am–midday	0.566	0.988
20/24 in 5B, 1 teacher	Thu.: 9 am–midday, 2 pm–5 pm Fri.: 8 am–midday, 2 pm–5 pm	0.522	1
22/23 in 1A, 1 teacher	Thu.: 9 am–midday, 2 pm–5 pm Fri.: 8 am–midday, 2 pm–5 pm	0.470	0.994
85 children in all classes	Thu.: midday–2 pm Fri.: midday–2 pm	0.228	0.184
21/23 in 4B, 1 teacher	Thu.: 10 am–midday, 2 pm–5 pm Fri.: 9 am–midday	0.447	1
55 children in 1A, 2A, 2B, 3A, 3B	Fri.: midday–2 pm	0.430	0.905
52 children in 1A, 1B, 2A, 2B, 3A	Thu.: midday–2 pm	0.429	0.936

They deal with the cross-class subset of children eating at the common canteen[4] and have densities that are smaller than those of the ten class patterns, yet significantly greater than $\hat{\lambda}_0 = 0.049$. The 10th pattern in Table 6 involves all four lunch-break hours. The 12th and 13th patterns, only those on Friday and Thursday, respectively. Those two patterns overlap with the 10th pattern and have greater densities, to be considered at the intersections, according to Equation (1).

The last column of Table 6 informs how often and accurately thirty executions of NclusterBox with $m = 1000$ in Equation (6) discover each pattern. Given the thirty summaries, $\mathcal{X}_1, \ldots, \mathcal{X}_{30}$, that Algorithm 5 returns, the *recovery* of a pattern P, in Table 6, is defined as $\frac{1}{30}\sum_{\ell=1}^{30} \max_{X \in \mathcal{X}_\ell} \frac{|P \cap X|}{|P \cup X|}$, *i.e.*, on average over the thirty summaries, how similar the most similar pattern. That average is 1 for eight of the ten class patterns: they are always found, unmodified. The remaining two class patterns are always found too. Nevertheless, in four summaries, one child of 4A is missing, and, in two summaries, one child of 1A. Besides that, the class patterns are identical to those obtained with $m = 57\,542$. Three of the thirty summaries include the 12th pattern in Table 6. Its recovery, 0.905, is high because, in 25 additional summaries, there is a pattern that involves the same two hours and the same 55 children except one, who is missing. The remaining two summaries have no pattern specifically dealing with the lunch-break on Friday. In the same way, two of the thirty summaries include no pattern specifically dealing with the lunch-break on Thursday. In the other 28 summaries, that pattern is exactly the last one in Table 6. Finally, the least dense pattern in the table is the least recovered: three of the thirty executions of NclusterBox with $m = 1000$ output the 10th pattern, unmodified. Overall, $m = 1000$ looks enough to rather accurately summarize the contact tensor. A greater value may allow NclusterBox to reach slightly different patterns (*e.g.*, with one

[4] "Some children were going back home to have lunch." [30].

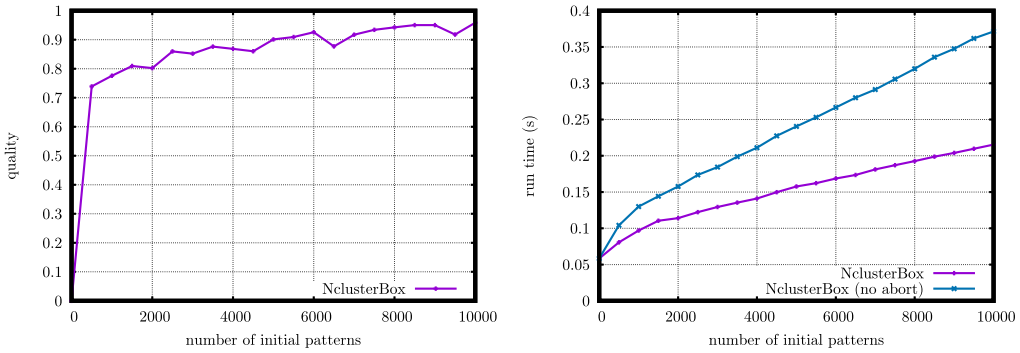

Fig. 14. In function of the number of initial patterns, m in Equation (6), similarity of the top-13 patterns to those in Table 6 obtained with $m = 57\,542$ (left) and run time (right) including the selection and ranking (Algorithm 5).

additional child in both subsets of $D_{\text{individuals}}$) relating to a higher local maximum of g. More rarely, a significantly different pattern of low density may be output thanks to the greater m.

To further study how the number of initial patterns impacts the quality of the summary and the run time, the summary in Table 6 is considered a ground truth and different numbers of initial patterns are enumerated by increment of 500, from 0 to 10 000. For each number m, NclusterBox runs thirty times. Fig. 14 plots (on the left) the average quality of the thirty resulting summaries and (on the right) the average time NclusterBox and Algorithm 5 together take. Denoting $\mathcal{P}$ the set of 13 patterns in Table 6 and $\mathcal{X}$ the first 13 patterns in a summary, Equation (7) defines the quality. Even with $m = 500$, it is rather high, 0.739. Of course, it tends to increase with m, however less and less. The run time increases less and less too, as m increases. The beginning of Subsection 5.1 explains why: Algorithm 4 aborts the maximization of g whenever it reaches a previously visited pattern. The greater m, the more often it happens and, when it does, the earlier in the hill-climbing maximization. Disabling that single improvement, the run time linearly increases with m, as Fig. 14 shows. Accepting that overhead may make sense. Indeed, it comes with a reduction of the memory usage: besides the shifted tensor, only patterns locally maximizing g are stored. With the contact tensor though, and $m = 57\,542$, NclusterBox requires only 193 MB of RAM, which are still more than for any other experiment in this article.

Setting, for GETF, the maximal rank of the decomposition to 13 and, for TriclusterBox, $D_1 = D_{\text{hours}}$ in Equation (3), either algorithm outputs one single pattern. GETF requires 0.46 second and 414 MB of RAM; 5.46 seconds and 12 MB for TriclusterBox. The pattern GETF discovers involves all the elements of all the dimensions, except one child of only one of the two dimensions $D_{\text{individuals}}$. The density of that pattern is essentially $\hat{\lambda}_0 = 0.049$. The pattern TriclusterBox outputs is better. Nevertheless, it is still worse than the first pattern in Table 6, according to Fig. 13. Its density is 0.151. The same individuals appear in the two subsets of $D_{\text{individuals}}$, but they are too numerous. Five teachers and 107 children they teach to: 24 of the 25 children in the 1B class, 21/23 in 2A, 23/23 in 3A, 21/22 in 3B, and 18/22 in 5A. The pattern associates them with only eight hours: 10 am–11 am and 4 pm–5 pm on Thursday; 9 am–12 pm and 2 pm–5 pm on Friday.

7. Conclusion

For simply modeling a tensor with a set of sub-tensors associated with their densities, the disjunctive box cluster model is easy to interpret. Even more so if, as this article has proposed, the sub-tensors composing the summary are sorted in descending order of contribution to the fit and a selection is made, by stepwise regression and the elbow method. Although the literature on Boolean matrix/tensor factorization has mostly ignored it, BiclusterBox/TriclusterBox [1] may effectively summarize such datasets. This article has extended its scope to fuzzy matrices/tensors, much improved the quality of the returned summaries thanks to single-cell patterns as starting points for the hill climbing, and lowered its time requirements, often by orders of magnitude. In-depth experiments have shown the resulting algorithm, NclusterBox, clearly exceeds the performances of the state-of-the-art methods for matrix factorization using the max operator (so that no value in the reconstructed matrix can exceed 1) and for Boolean tensor factorization, both in terms of accuracy and efficiency.

For example, NclusterBox has perfectly recovered overlapping 25×25 (respectively $5 \times 5 \times 5$) all-ones sub-tensors planted in a 1000×1000 (respectively $100 \times 100 \times 100$) zero tensor even if every 0 or 1 has been switched with a 12.5% probability. In contrast, competitors fail at a 0.2% probability. If the tensor is fuzzy rather than Boolean, NclusterBox's recovery remains perfect at even higher levels of noise, hence the usefulness of the generalization. Real-world applications have confirmed the superiority of NclusterBox. Although the absence of ground truths turns the analyses necessarily more subjective, NclusterBox has clearly exceeded the state-of-the-art algorithms. It has output sensible patterns of influence in retweet data and, given timestamped face-to-face contacts in a primary school, it has almost perfectly found the classes and their schedules. Moreover, discovering those patterns has usually taken less than a hundredth of the time competitors have required.

The empirical comparison to BiclusterBox and TriclusterBox has demonstrated that starting every hill climbing search with an individual cell eventually leads to building better summaries. Although a large sub-tensor can make an excellent single-pattern summary and is harder to reach, the proposed post-processing procedure has often advantageously replaced it with several smaller and denser sub-tensors that become easier to discover. That procedure is a bidirectional stepwise regression. Experiments have shown

it tends to overfit the fuzzy tensor. Nevertheless, the ranking it gives has proven more useful than the selection it makes. The data scientist is invited to interpret the sub-tensors of the summary in the order they are listed. She may stop at an elbow that, even in real-world applications, has usually appeared in the curve of the residual sum of squares along the ranking. Although a remarkably good summary of a dynamic graph, seen as a 3-way 0/1 tensor, has been obtained, further gains are expected if the disjunctive box cluster model is modified to target that specific context.

CRediT authorship contribution statement

Victor Henrique Silva Ribeiro: Writing – review & editing, Investigation, Data curation. **Loïc Cerf:** Writing – review & editing, Writing – original draft, Visualization, Supervision, Software, Methodology, Funding acquisition, Conceptualization.

Funding source

Victor's scientific initiation is funded by the FAPEMIG/PRPq/UFMG (PROBIC 04/2023).

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Victor Henrique Silva Ribeiro reports financial support was provided by Minas Gerais State Foundation of Support to the Research. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.ins.2025.122489.

Data availability

The source code of NclusterBox, the data and the scripts to reproduce all the experiments are published under the terms of the GNU GPLv3+ on https://dcc.ufmg.br/~lcerf/en/prototypes.html#nclusterbox

References

[1] B.G. Mirkin, A.V. Kramarenko, Approximate bicluster and tricluster boxes in the analysis of binary data, in: Proceedings of the 13th International Conference on Rough Sets, Fuzzy Sets, Data Mining and Granular Computing (RSFDGrC 2011), 2011, pp. 248–256.
[2] P. Miettinen, S. Neumann, Recent developments in Boolean matrix factorization, in: Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI 2020), 2020, pp. 4922–4928.
[3] L. Cerf, J. Besson, C. Robardet, J.-F. Boulicaut, Closed patterns meet *n*-ary relations, ACM Trans. Knowl. Discov. Data 3 (1) (2009) 1–36, https://doi.org/10.1145/1497577.1497580.
[4] D.I. Ignatov, D.V. Gnatyshak, S.O. Kuznetsov, B.G. Mirkin, Triadic formal concept analysis and triclustering: searching for optimal patterns, Mach. Learn. 101 (1) (2015) 271–302, https://doi.org/10.1007/s10994-015-5487-y.
[5] D.I. Ignatov, A. Semenov, D. Komissarova, D.V. Gnatyshak, Multimodal clustering for community detection, in: Formal Concept Analysis of Social Networks, 2017, pp. 59–96.
[6] D.A. Egurnov, D.I. Ignatov, Triclusters of close values for the analysis of 3D data, Autom. Remote Control 86 (6) (2022) 894–902, https://doi.org/10.1134/S0005117922060078.
[7] E. Spyropoulou, T.D. Bie, M. Boley, Interesting pattern mining in multi-relational data, Data Min. Knowl. Discov. 28 (3) (2014) 808–849, https://doi.org/10.1007/s10618-013-0319-9.
[8] E. Spyropoulou, T.D. Bie, Mining approximate multi-relational patterns, in: Proceedings of the 2014 IEEE International Conference on Data Science and Advanced Analytics (DSAA 2014), 2014, pp. 477–483.
[9] L. Cerf, W. Meira Jr., Complete discovery of high-quality patterns in large numerical tensors, in: Proceedings of the 30th IEEE International Conference on Data Engineering (ICDE 2014), 2014, pp. 448–459.
[10] E. Georgii, K. Tsuda, B. Schölkopf, Multi-way set enumeration in weight tensors, Mach. Learn. 82 (2) (2011) 123–155, https://doi.org/10.1007/s10994-010-5210-y.
[11] L. Zhao, M.J. Zaki, triCluster: an effective algorithm for mining coherent clusters in 3D microarray data, in: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2005), 2005, pp. 694–705.
[12] L. Cerf, P.-N. Mougel, J.-F. Boulicaut, Agglomerating local patterns hierarchically with ALPHA, in: Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM 2009), 2009, pp. 1753–1756.
[13] I. Leenen, I. Van Mechelen, P. De Boeck, S. Rosenberg, INDCLAS: a three-way hierarchical classes model, Psychometrika 64 (1) (1999) 9–24, https://doi.org/10.1007/BF02294316.
[14] P. Miettinen, Boolean tensor factorizations, in: Proceedings of the 11th IEEE International Conference on Data Mining (ICDM 2011), 2011, pp. 447–456.
[15] R. Bělohlávek, C.V. Glodeanu, V. Vychodil, Optimal factorization of three-way binary data using triadic concepts, Order 30 (2) (2013) 437–454, https://doi.org/10.1007/s11083-012-9254-4.
[16] D. Erdős, P. Miettinen, Walk 'n' merge: a scalable algorithm for Boolean tensor factorization, in: Proceedings of the 13th IEEE International Conference on Data Mining (ICDM 2013), 2013, pp. 1037–1042.
[17] S. Metzler, P. Miettinen, Clustering Boolean tensors, Data Min. Knowl. Discov. 29 (5) (2015) 1343–1373, https://doi.org/10.1007/s10618-015-0420-3.
[18] T. Rukat, C.C. Holmes, C. Yau, Probabilistic Boolean tensor decomposition, in: Proceedings of the 35th International Conference on Machine Learning (ICML 2018), 2018, pp. 4410–4419.

[19] N. Park, S. Oh, U. Kang, Fast and scalable method for distributed Boolean tensor factorization, VLDB J. 28 (4) (2019) 549–574, https://doi.org/10.1007/s00778-019-00538-z.
[20] C. Wan, W. Chang, T. Zhao, S. Cao, C. Zhang, Geometric all-way Boolean tensor decomposition, in: Advances in Neural Information Processing Systems (NeurIPS 2020), 2020, pp. 2848–2857.
[21] L. Gauvin, A. Panisson, C. Cattuto, Detecting the community structure and activity patterns of temporal networks: a non-negative tensor factorization approach, PLoS ONE 9 (1) (2014), https://doi.org/10.1371/journal.pone.0086028.
[22] S. Karaev, P. Miettinen, Cancer: another algorithm for subtropical matrix factorization, in: Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2016), 2016, pp. 576–592.
[23] R. Bělohlávek, V. Vychodil, Factor analysis of incidence data via novel decomposition of matrices, in: Proceedings of the 7th International Conference on Formal Concept Analysis (ICFCA 2009), 2009, pp. 83–97.
[24] R. Bělohlávek, Concept lattices and order in fuzzy logic, Ann. Pure Appl. Log. 128 (1–3) (2004) 277–298, https://doi.org/10.1016/J.APAL.2003.01.001.
[25] R. Bělohlávek, M. Trnečková, Factorization of matrices with grades with overcovering, IEEE Trans. Fuzzy Syst. 32 (4) (2024) 1641–1652, https://doi.org/10.1109/TFUZZ.2023.3330760.
[26] V.H.S. Ribeiro, L. Cerf, Summarizing fuzzy tensors with sub-tensors, in: Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing (SAC 2023), 2023, pp. 362–364.
[27] L. Maciel, J. Alves, V.F. dos Santos, L. Cerf, Climbing the hill with ILP to grow patterns in fuzzy tensors, Int. J. Comput. Intell. Syst. 13 (1) (2020) 1036–1047, https://doi.org/10.2991/IJCIS.D.200715.002.
[28] G. Schwarz, Estimating the dimension of a model, Ann. Stat. 6 (2) (1978) 461–464, https://doi.org/10.1214/aos/1176344136.
[29] G. Smith, Step away from stepwise, J. Big Data 5 (1) (2018), https://doi.org/10.1186/s40537-018-0143-6.
[30] J. Stehlé, N. Voirin, A. Barrat, C. Cattuto, L. Isella, J.-F. Pinton, M. Quaggiotto, W.V. den Broeck, C. Régis, B. Lina, P. Vanhems, High-resolution measurements of face-to-face contact patterns in a primary school, PLoS ONE 6 (8) (2011), https://doi.org/10.1371/journal.pone.0023176.